



User Guide and Reference Manual

September 2024



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.

Scientific Toolworks, Inc.
444 E Tabernacle Suite #B101
St George, UT 84770

Copyright © 2024 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., 444 E Tabernacle Suite #B101, St George, UT 84770.

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

Understand by Scientific Toolworks, Inc. is certified for use as a support tool for all projects requiring ISO 26262, IEC 61508, and EN 50128 compliance.

Reference Build: Understand 6.5 Build 1201 (9/24)

Contents

Chapter 1	Introduction	
	What is Understand?	14
	Languages Supported	15
	Installing Understand	16
	Certifications	16
	For Those Who Don't Like to Read Manuals	17
Chapter 2	Parts and Terminology	
	Using Understand Windows	19
	Understand Terminology	20
	Parts	21
	Starting Understand	22
	Other Ways to Run Understand	23
	Context Menus Are Everywhere	24
	Quickly Find Things in Your Source	26
	Entity Filter	26
	Entity Locator	27
	Instant Search	27
	Find in Files	28
	Favorites	28
	Information Browser	29
	Source Editor	30
	Architecture Browser	31
	Graphical Views	32
	APIs for Custom Reporting	33
Chapter 3	Configuring Your Project	
	About Understand Projects	35
	Project Storage	35
	Creating a New Project	37
	Add Source Code Page	38
	Open Existing Page	38
	CMake Compilation Page	38
	Import Settings and Preview Import Pages	38
	Build Project Page	38
	Build Wrapper Page	39
	Select Languages Page	39
	Background Checks Page	40
	Set Location Page	41
	Creating Projects for Git Repositories	42
	Using Build Watcher to Create Projects	43
	Creating Projects from CMake Projects	44

Creating Projects from Visual Studio Projects	45
Creating Projects from Apple Xcode Projects	45
Project Configuration Dialog	46
Languages Category	48
Files Category	49
Adding Directories	50
Adding Files	51
Removing Directories and Files	51
Setting Overrides	52
File Type Options	54
File Options.	55
Report Options	56
Reports > Selected Category	58
Imports Options	59
History Options	61
Search Options	62
Dependencies Options	62
Metrics Options	63
Portability Options	66
Ada Options	68
Ada > Macros Category	70
Assembly Options	72
Visual Basic (.NET) Options	74
C/C++ Options	75
Strict C/C++ Mode Options	76
Fuzzy C/C++ Mode Options	79
C++ > Includes Category	81
C++ > Includes > Automatic Includes Category	83
C++ > Includes > System Includes Category	83
C++ > Includes > Frameworks Category	84
C++ > Includes > Ignore Category	84
C++ > Includes > Replacement Text	84
C++ > Macros Category	85
C++ > Macros > Undefines Category	86
C# Options	87
Fortran Options	89
Fortran>Includes Category	91
Other Fortran Categories	91
Java Options	92
Java > Class Path Category	93
JOVIAL Options	94
Jovial > !Copy Category	95
Pascal Options	96
Pascal > Macros Category	97

Pascal > Standard Library Paths Category	97
Pascal > Search Paths Category	97
Python Options	98
Python > Imports Category	99
VHDL Options	99
Web Options	100
Analyzing the Code	102
Improving the Analysis	104
Using the Missing Header Files Tool	105
Using the Undefined Macros Tool	106

Chapter 4

Setting Options

Understand Options Dialog	108
General Category	109
User Interface Category	111
User Interface > Lists Category	112
User Interface > Alerts Category	113
User Interface > Windows Category	114
User Interface > Application Styles Category	115
Key Bindings Category	116
CodeCheck Category	118
Analyze Category	119
Dependency Category	120
Editor Category	121
Editor > Advanced Category	123
Editor > Macros Category	127
Editor > Styles Category	127
Editor > Navigation Category	128
Editor > External Editor Category	129
Graphs Category	130
Annotations Category	132
User Tools Category	133

Chapter 5

Exploring Your Codebase

PLEASE RIGHT-CLICK	135
Various Windows Explained...	136
Entity Filter	137
Using the Filter Field	138
Customizing the Display	138
Root Entity Types	139
Information Browser	139
Drilling Down a Relationship	140
Displaying More or Less Information	141
Searching the Information Browser	141
Syncing the Information Browser	142

Visiting Source Code	142
Visiting References	143
Viewing Metrics	143
Saving and Printing Information Browser Text	144
Entity History	144
Project Browser	145
Exploring a Hierarchy	147
Dependency Browser	148
What are Dependencies?	150
Exporting Dependencies	152
Exporting Dependencies to a CSV File	153
Exporting Dependencies to a CSV Matrix File	154
Exporting Dependencies to Cytoscape	154
Favorites	155
Creating a Favorite Entity	155
Creating a Favorite View	156
Using a Favorites Group	156
Creating a Plain Text Favorite	158

Chapter 6

Searching Your Source

Searching: An Overview	160
Instant Search	161
Find in Files	163
Search Results	165
Replace in Files	166
Entity Locator	168
Resizing Columns	168
Long versus Short Names	168
Column Headers	169
Choosing Columns	169
Filtering the List	170
Finding Windows	172
Source Visiting History	173
View Menu Commands	174
Displaying Toolbars	175
Session Browser	175
Searching in a File	176
Find Next and Previous	176
Find & Replace	176
Contextual Information	177

Chapter 7

Editing Your Source

Source Editor	179
File Icons	180
Scope List	180

Toolbar	181
Status Line	181
Selecting and Copying Text	182
Browse Mode	182
Context Menu	183
Hover Text	184
Saving Source Code	184
Refactoring Tools	185
Renaming Entities	186
Renaming Files	187
Inlining Functions	189
Extracting Functions	190
Inlining Scope	191
Inline Temp	192
Extract Temp	193
Other Editing Features	194
Previewer	195
Bracket Matching	195
Folding and Hiding	196
Changing the Source Code Font Size	196
Changing Case	196
Commenting and Uncommenting	196
Splitting the Editor Window	197
Indentation	197
Line Wrapping	198
Insert and Overtyping Modes	198
Sorting Lines Alphabetically	198
Keyboard Commands	198
Recording, Playing, and Saving Macros	198
Creating and Opening Files	199
Bookmarking	199
Managing Source Editor Tabs	201
Spell Checking	202
Annotations	203
Viewing Annotations	204
Sharing Annotations	204
Searching for Annotations	204
Adding Annotations	205
Editing Annotations	205
Tagging Annotations	206
Attaching Files	206
Deleting Annotations	207
Managing Orphan Annotations	207
Using Annotations to Ignore Violations	207

	Printing Source Views	208
Chapter 8	Architecting Your Codebase	
	About Architectures	210
	Browsing Architectures	211
	Enabling Built-In Architectures	212
	Context Menus for Architectures	212
	Creating and Editing Custom Architectures	214
	Creating Architecture Nodes	215
	Editing Architecture Nodes	215
	Adding Files and Entities to Nodes	215
	Complete Coverage for Architectures	216
	Managing Orphan Architecture Nodes	216
	Using the Architecture Designer	217
	Sharing Architectures	219
	Using Automatic Architectures	219
	Viewing Architecture Graphs	220
	Dependency Graphs	220
	Architecture Structure Views	221
	Checking Dependencies	222
	Viewing Architecture Metrics	223
Chapter 9	Using Reports	
	Configuring Reports	225
	Customizing Report Colors	226
	Generating Reports	226
	Viewing Reports	227
	An Overview of Report Categories	228
	Augment with Plugins	229
	Cross-Reference Reports	230
	Data Dictionary Report	230
	Program Unit Cross-Reference Report	231
	File Contents Report	231
	Object Cross-Reference Report	232
	Type Cross-Reference Report	232
	Macro Cross-Reference	233
	Include File Cross-Reference	233
	Exception Cross-Reference Report	233
	Structure Reports	234
	Declaration Tree	234
	Extend Tree	235
	Invocation Tree Report	235
	Simple Invocation Tree Report	235
	With Tree Report	236
	Simple With Tree Report	236

Generic Instantiation Report	236
Renames Report	236
Imports Report	236
Quality Reports	237
Program Unit Complexity Report	237
Fortran Extension Usage Report	238
Implicitly Declared Objects Report	238
Uninitialized Items	238
Unused Objects and Functions	238
Unused Objects Report	239
Unused Types Report	239
Unused Program Units Report	240
Uses Not Needed Report	240
Withs Not Needed Report	240
Metrics Reports	241
Project Metrics Report	241
Class Metrics Report	242
Class OO Metrics Report	242
Program Unit Metrics Report	243
File Metrics Report	243
File Average Metrics Report	244
Importing Report Plugins	244

Chapter 10

Using Metrics

About Metrics	246
Home	247
Metrics Browser	248
Metric Definitions	249
Exporting Metrics	250
Metrics in the Information Browser	252
Metrics Treemap	253
Metric Plugins	256

Chapter 11

Using Graphical Views

Opening Graphs	258
Choosing Graph Variants	261
Controlling Graphical Views	263
Graph Toolbar	263
Graph Display Options	265
Mouse Controls	266
Customization Panel Controls	266
Classic Context Menu Controls	268
Controlling Graph Styles	271
Viewing the Legend	271
Using the Legend: Summary	271

Modifying Styles in the Legend	272
Adding Node Styles	272
Adding Edge Styles	273
Shading Nodes or Edges Using a Metric	274
Removing Node and Edge Styles	275
Saving, Importing, and Exporting Styles	275
Using Context Menus for Graphs	277
Saving and Exporting Graphical Views	285
Printing Graphical Views	286
Graphical View Printing	286
Importing Graphical View Plugins	287

Chapter 12

Using CodeCheck

About CodeCheck	289
Running a CodeCheck	290
Re-Analyzing the Project	291
Configuring Checks	291
Running Configurations in the Background	294
Running Configurations on the Command Line	294
Sharing Configurations	295
Selecting Files to Check	295
Viewing Results	296
Viewing Results by File	297
Viewing Results by Checks	298
Viewing Results in the Locator	299
Viewing Results by Treemap	299
Viewing the Results Log	301
Viewing Results in Violation Browser	302
Viewing Violation Metrics	304
Viewing SARIF Files	304
Ignoring or Excluding Violations	305
Ignoring Violations in CodeCheck	305
Ignoring Violations in the Source Editor	307
Adding Notes About Ignored Violations	308
Using Baseline Ignore Settings	308
Adding Automatic Ignores to Code	309
Exporting CodeCheck Results	310
Writing CodeCheck Scripts	312
Installing Custom Scripts	314

Chapter 13

Comparing Source Code

Comparing Files and Folders	316
Comparing Entities	319
Comparing Text	320
Comparing Projects	320

	Comparison Graphs	323
	Comparison Treemaps	324
	Exploring Git History	326
	Exploring Differences	328
Chapter 14	Running Tools and External Commands	
	Configuring Tools	331
	Variables	333
	Adding Tools to the Context Menus	338
	Adding Tools to the Tools Menu	339
	Adding Tools to the Toolbar	340
	Importing and Exporting Tool Commands	341
	Running External Commands	342
	Installing and Running Plugins	343
	APIs	343
	Interactive Reports	344
	Plugin Graphs	344
	CodeCheck Scripts	344
	Metric Plugins	344
	Architecture Plugins	345
Chapter 15	Command Line Processing	
	Using the und Command Line	347
	Getting Help on Und.	349
	Creating a New Project	349
	Converting a Legacy Project	349
	Adding Files to a Project	349
	Removing Items from a Project	351
	Getting Information about a Project and the License . . .	351
	Modifying Project Settings	352
	Importing into a Project	352
	Exporting from a Project	352
	Analyzing a Project	353
	Generating Reports	353
	Generating Metrics	353
	Using CodeCheck	354
	Running Perl Scripts	354
	Creating a List of Files	354
	Using the understand Command Line	355
	Using Buildspy to Build Understand Projects	356
Chapter 16	Quick Reference	
	File Menu	358
	Edit Menu	359
	Search Menu	359

View Menu	360
Project Menu	360
Architectures Menu	361
Reports Menu	361
Metrics Menu	361
Graphs Menu	361
Checks Menu	362
Annotations Menu	362
Tools Menu	362
Compare Menu	363
Window Menu	363
Help Menu	364

This chapter introduces the *Understand* software.

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

This chapter contains the following sections:

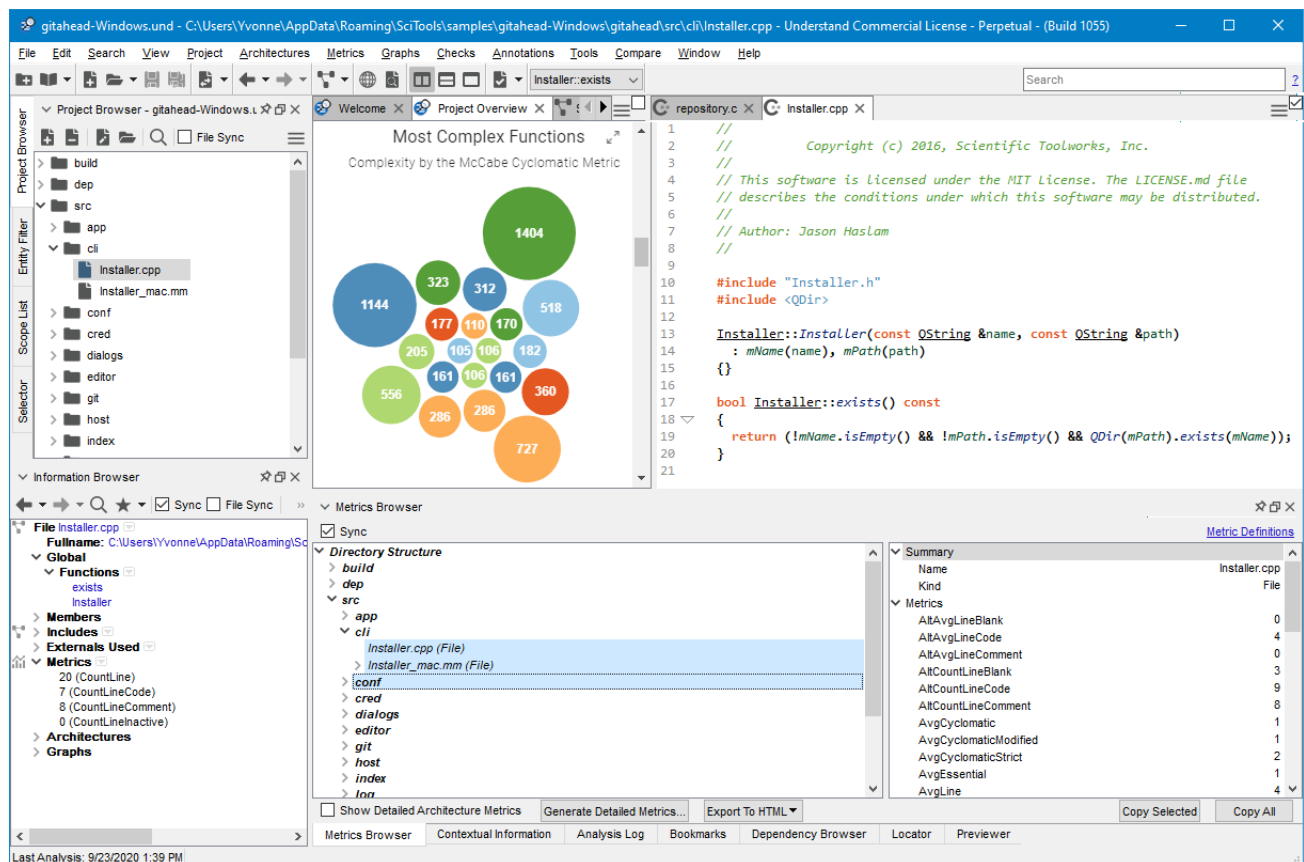
Section	Page
What is Understand?	14
Languages Supported	15
Installing Understand	16
For Those Who Don't Like to Read Manuals	17

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, Fortran, Java, JOVIAL, Pascal/Delphi, 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 for project analysis:

- **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 Studio .NET 2002 through Visual Studio 2022 for Visual Basic.
- **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, C++17, and C++20 are supported. CUDA files (.cu and .cuh) are also supported. *Understand* supports Visual Studio 97 through Visual Studio 2022 for C/C++.
- **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 11. *Understand* supports Visual Studio .NET 2002 through Visual Studio 2022 for C#.
- **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.
- **Java:** *Understand* supports Java through Java 18.
- **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.
- **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.

Installing Understand

Understand requires about 200-300 MB of installation space, depending on which OS it is installed on.

For larger code bases, the more RAM the better. We recommend 1 GB / million lines of code. *Understand* will run with less RAM, but it is much slower.

Understand is available for use on Windows, Linux, and MacOS. You can install an older version of *Understand* if you need support for a particular OS version. The current version supports the following OS releases:

- Windows 64-bit (Windows 10 and later)
- Linux 64-bit (CentOS & RHEL 8.4, Ubuntu 20.04, and later)
- MacOS (Mojave 10.14 and later)

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

For information about licensing, including using Offline mode, see the [Licensing category](#) on our support website. Choose **Help > Licensing** to see your license information.

Access to certain *Understand* features is controlled by the type of license you have. For example, these features include HTML Reports, Bug Hunter checks, and CodeCheck reports. If a feature described in this document is not available in your *Understand* installation, please contact support at support@scitools.com to add that feature to your license.

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

Certifications

Understand has achieved certification for ISO 26262, IEC 61508, and EN 50128 compliance from TÜV SÜD, a globally recognized testing and certification organization. This certification demonstrates our commitment to delivering a robust and reliable code analysis solution.

If you have specific compliance requirements, we offer a dedicated compliance package that includes the necessary certification to integrate *Understand* seamlessly into ISO 26262-compliant systems.

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.
- Choose **Help > Show Suggestions** from the menus.
- Visit support.scitools.com for knowledge base topics that explain useful features.
- Visit 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, you can click the **Chat** icon in the *Understand* Welcome tab and use the Welcome bot to send us your question.



For more advanced users, try these information sources:

- Choose **Help > About Understand** to see which build you are currently running.
- Visit 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 [support site](#) for knowledge base topics.
- Plugin and API examples, templates, and documentation are available in the `plugins` directory in your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>.

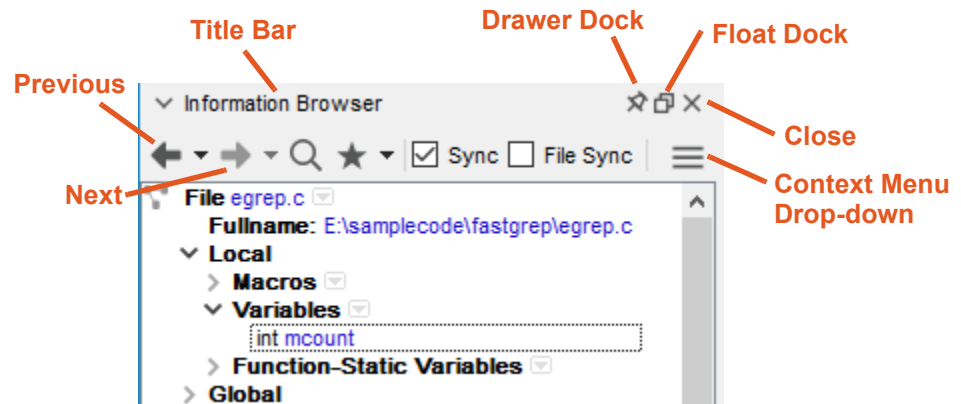
This chapter helps you put *Understand* to good use quickly and easily by describing the basic windows in *Understand*.





This chapter contains the following sections:

Section	Page
Using Understand Windows	19
Understand Terminology	20
Starting Understand	22
Context Menus Are Everywhere	24
Quickly Find Things in Your Source	26
Information Browser	29
Source Editor	30
Architecture Browser	31
Graphical Views	32
APIs for Custom Reporting	33

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 175](#)).

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 can be 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 35](#) 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.

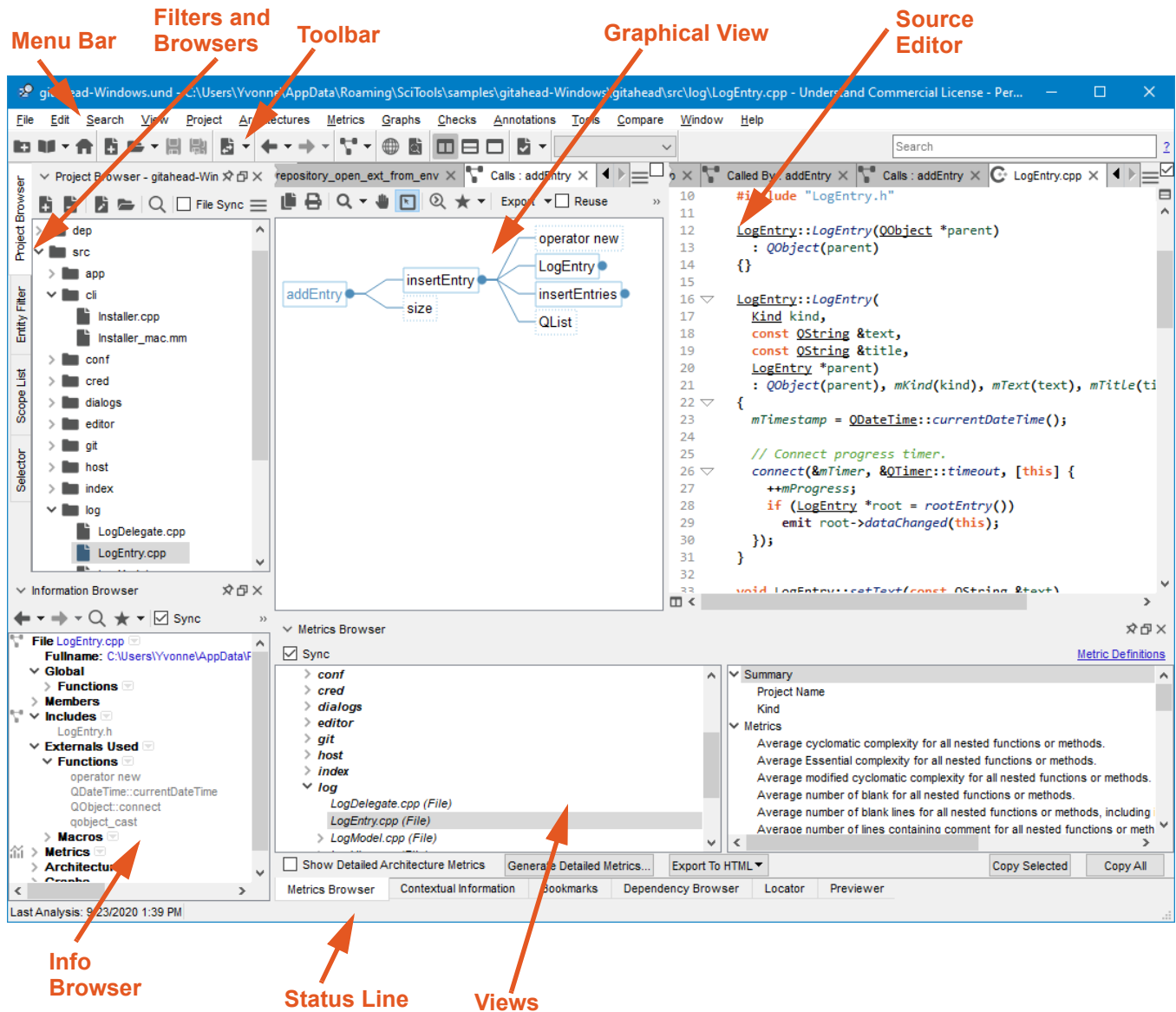
Project: The set of source code you have analyzed and the settings and parameters chosen. An .und project directory contains everything needed to share your project with others on your team.

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

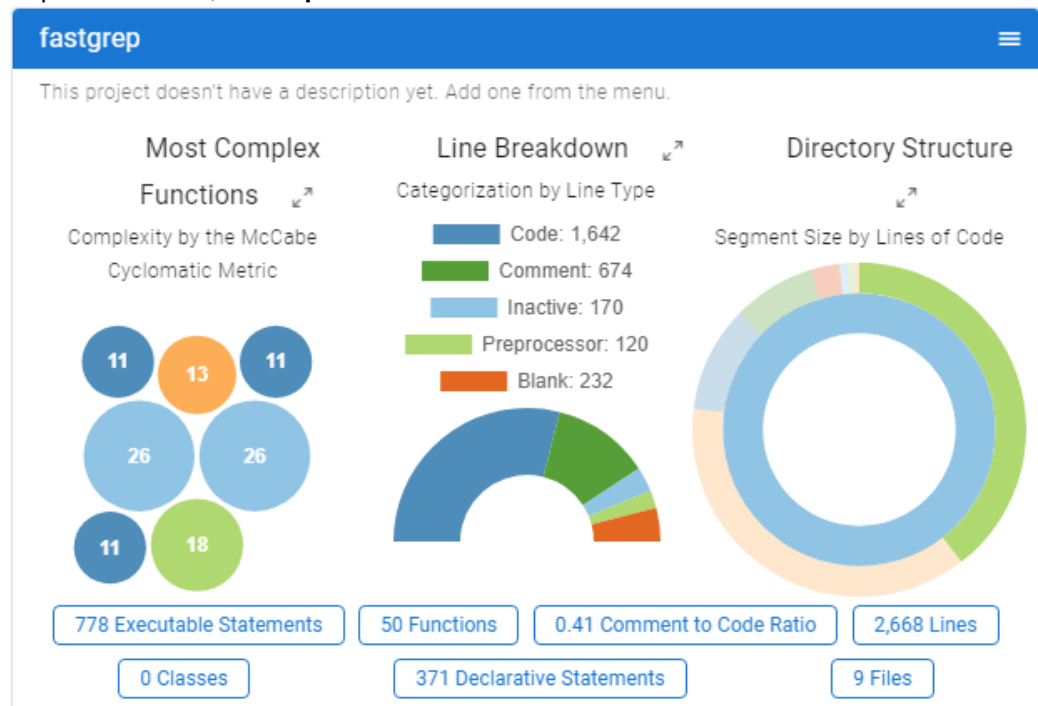
When you install *Understand* on Windows, a command to launch the software is added to your Windows **Start** menu.

When you start *Understand*, you see the **Welcome** page in the Understand window. To begin creating a new project, click **New** (see *Creating a New Project* on [page 37](#)).



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 or use the Open Project icon and drop-down list in the main toolbar to open a project.

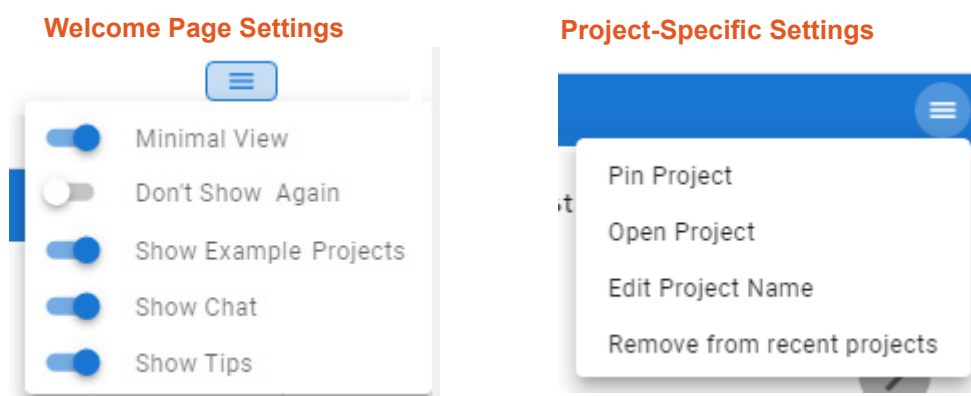
If you want to open a project that was created with an earlier version of *Understand* and convert it to the new project storage format, choose **File > Open > Legacy .udb Project**. Browse to select the file with a .udb extension that was used by earlier versions to contain projects instead of a directory. You are asked whether you want the new project stored in the same location as the source code or in a different location. Choosing the same location as the source code makes projects easier to share.

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

You can open one 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 changes in each file individually. To learn where the project files are saved and how you can store them in a source-code control system, see *Project Storage* on [page 35](#).

To show or hide parts of the Welcome page, use the drop-down menu for the Welcome page. The Minimal View displays the projects on smaller cards with one summary graphic at a time. The chat icon can be used to send a question to our support team.

To control the list of projects, use the drop-down menu for an individual project. You can pin the projects you use most to the top of the list. Editing the project name 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.

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

Other Ways to Run Understand

For information on running *Understand* from the command line, see [Chapter 15, 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

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category* on [page 109](#)).

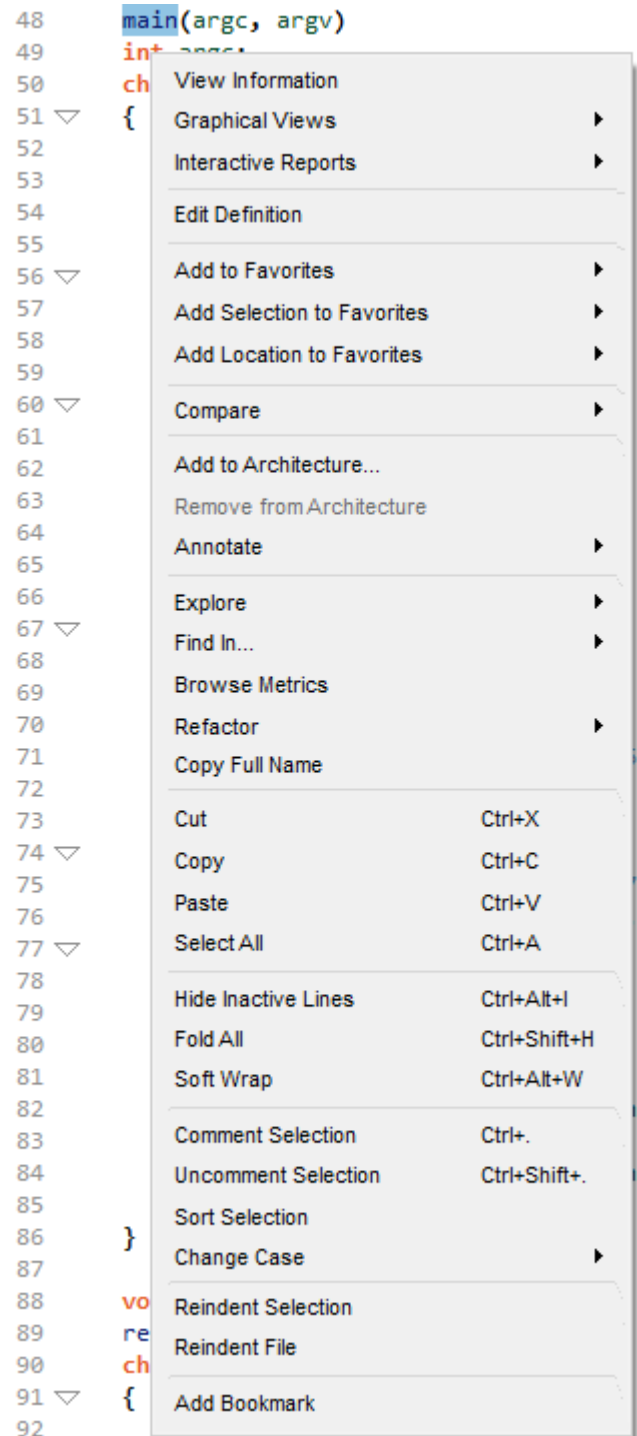
Context Menus Are Everywhere

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.

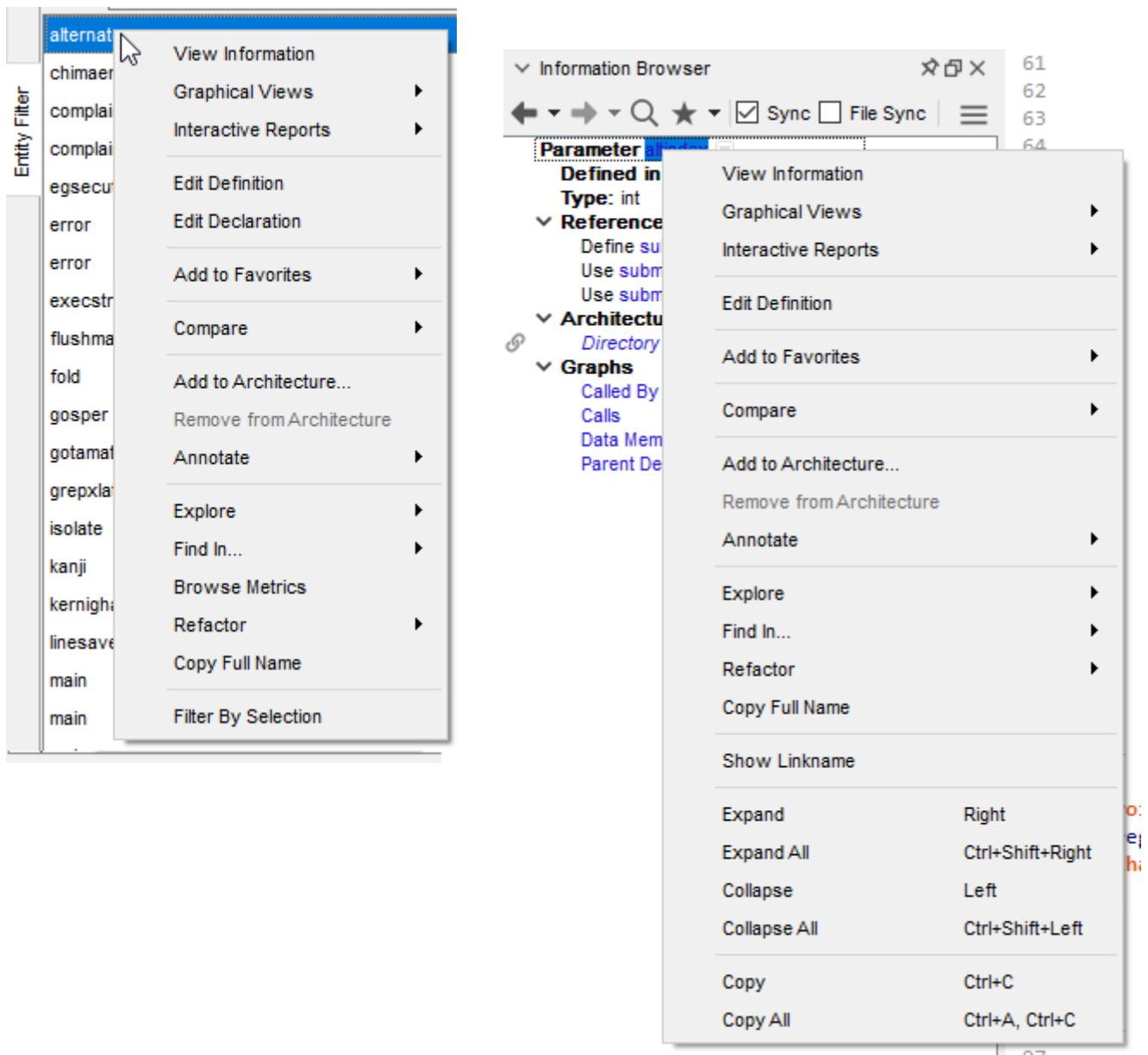
For example, if you right-click on an entity in the Source Editor, you see the list of commands shown here.

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

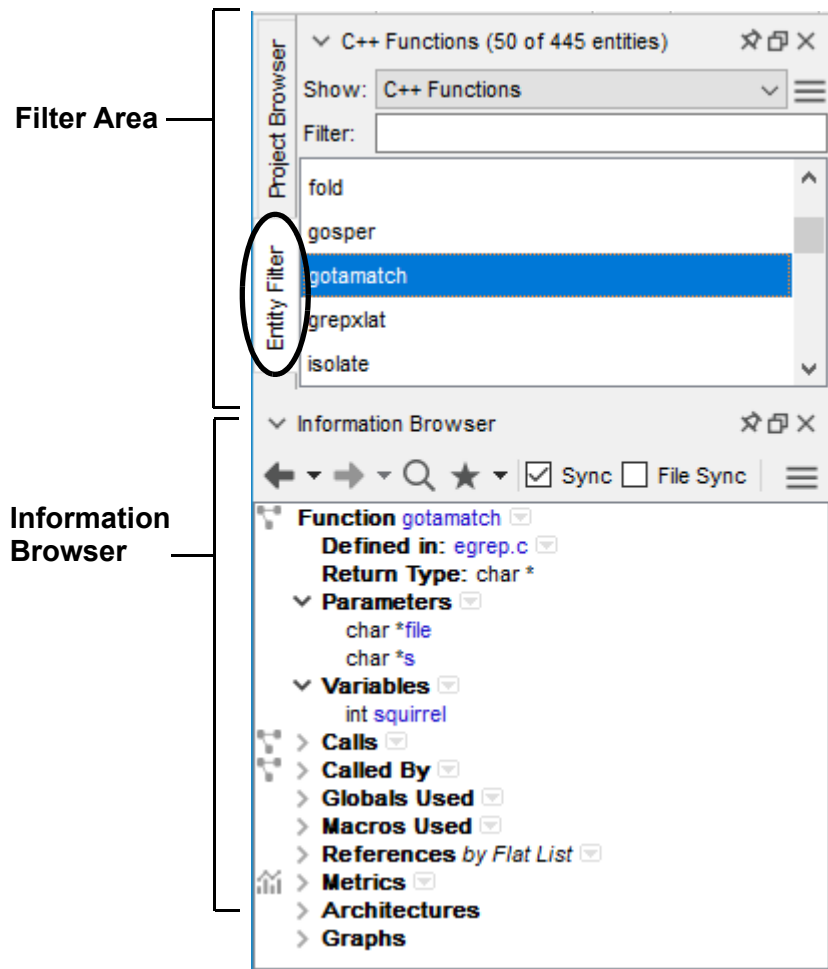
Understand provides several ways to quickly locate items of interest in your source code. These features include the Filter Area, the Entity Locator, and the Find in Files dialog.

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*.

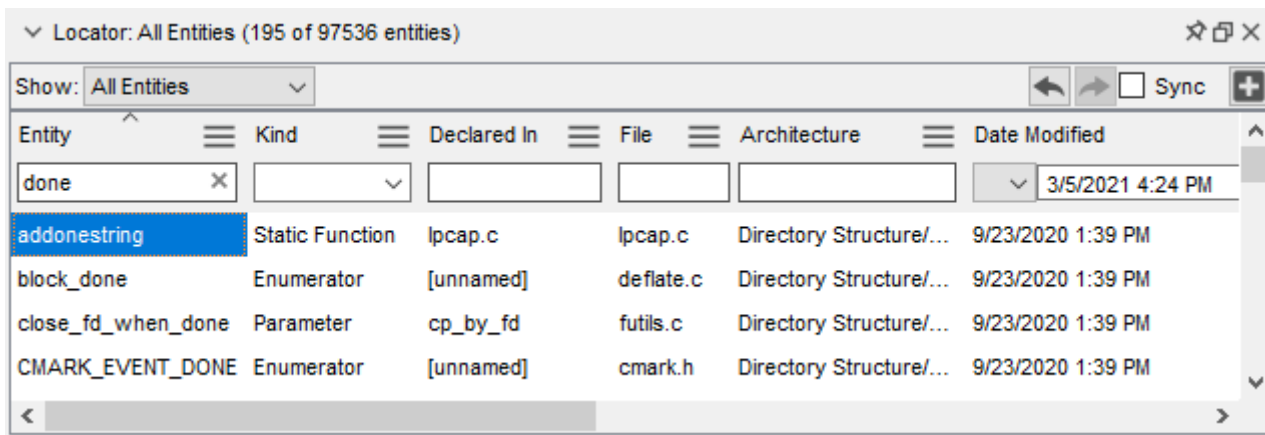


For details, see *Entity Filter* on [page 137](#) and *Information Browser* on [page 139](#).

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.

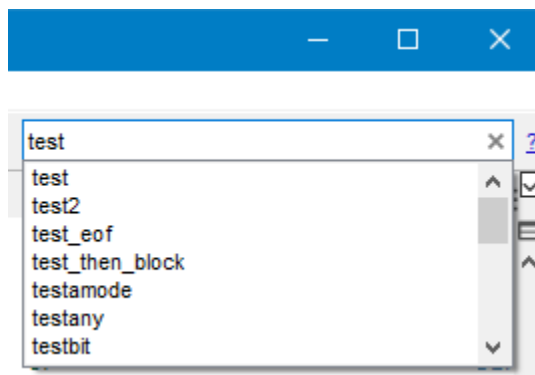
For details, see *Entity Locator* on [page 168](#).

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 161](#).

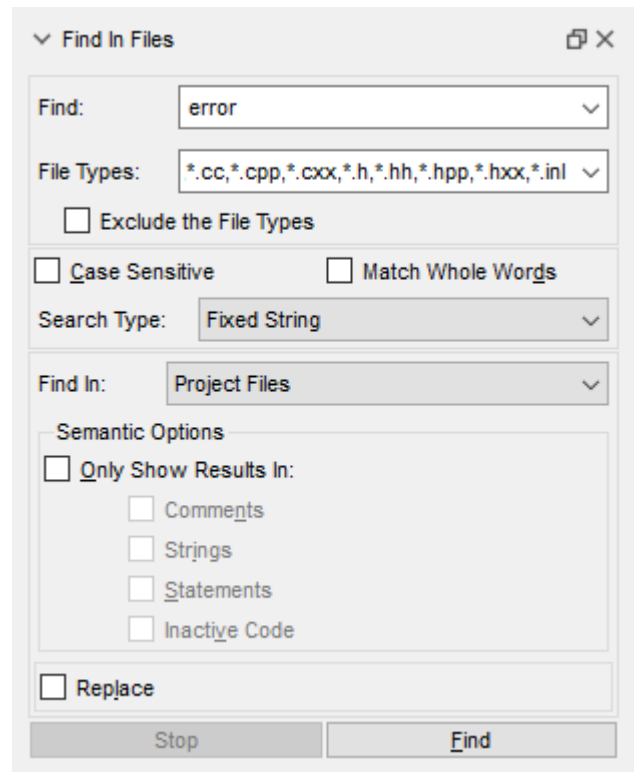
Find in Files

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.

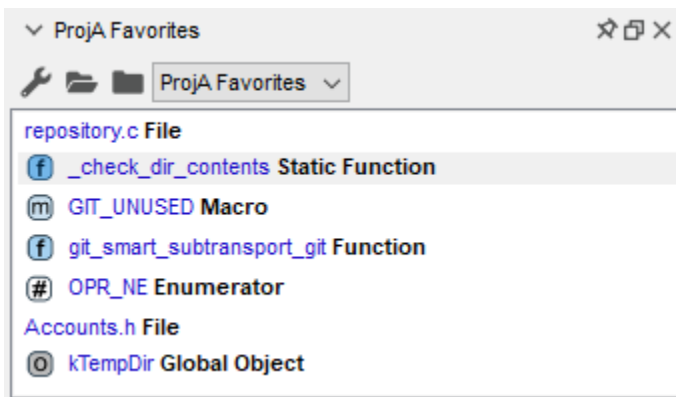
See *Find in Files* on [page 163](#) for more information.



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 155](#) for more information.

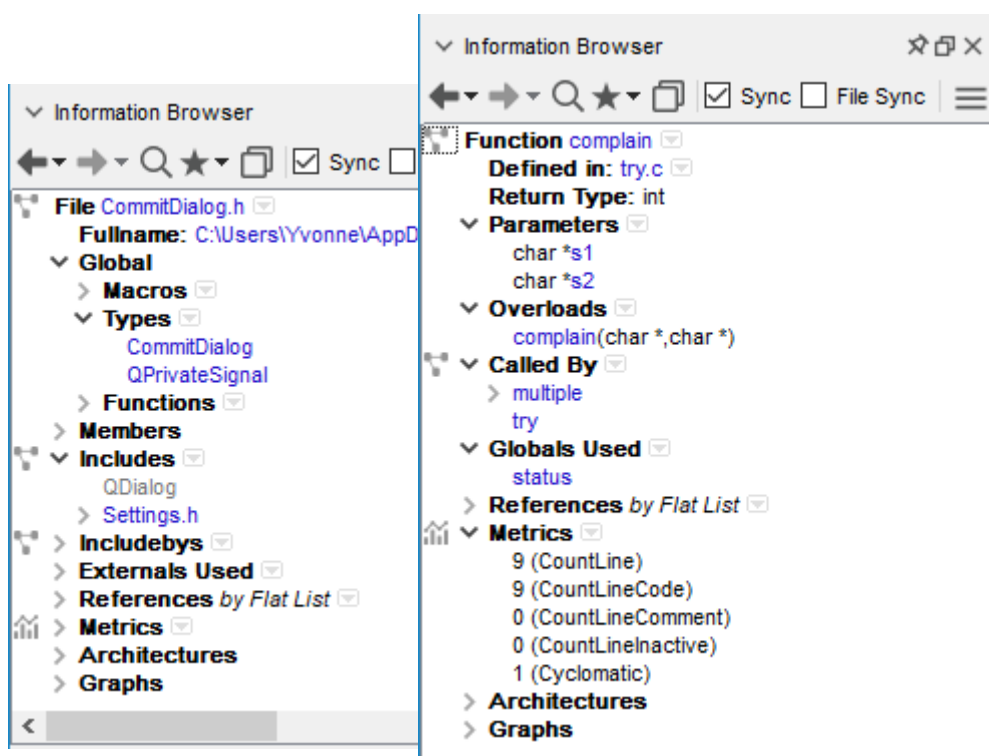
Information Browser

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:

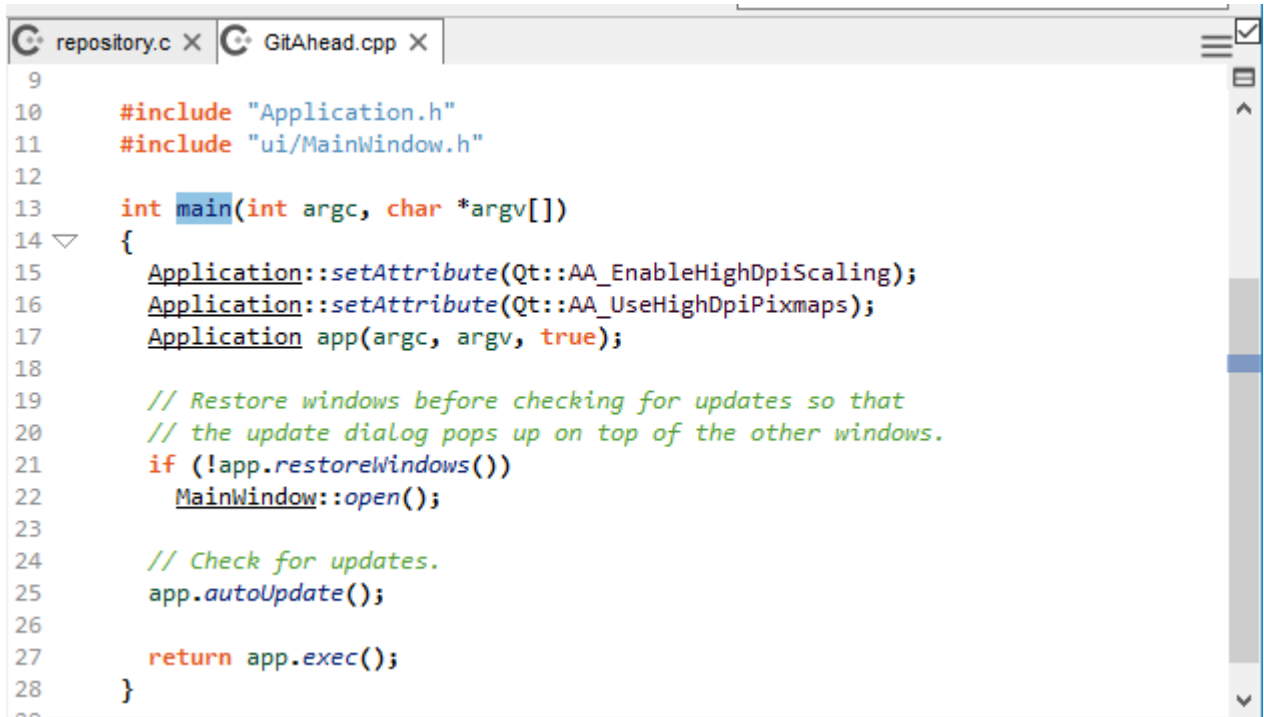


For details, see *Information Browser* on [page 139](#).

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.



```
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 }
```

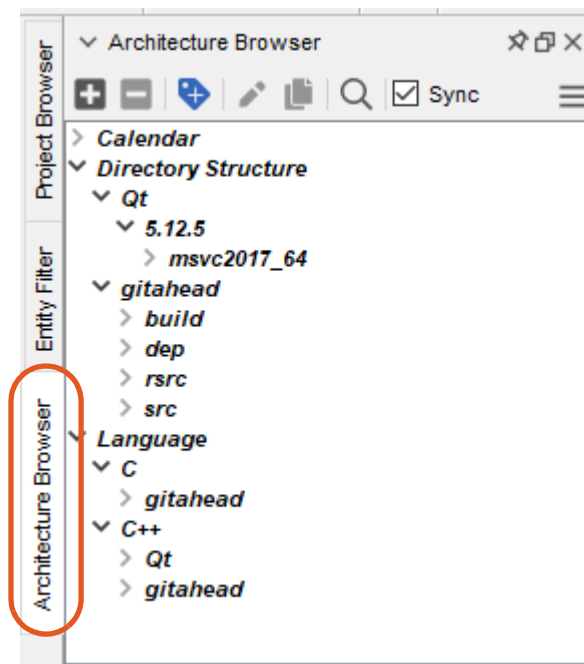
As with any other place in *Understand*, a context menu is available throughout the editor. To learn about something just right-click on it to see what information is available.

For details, see *Source Editor* on [page 179](#).

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.

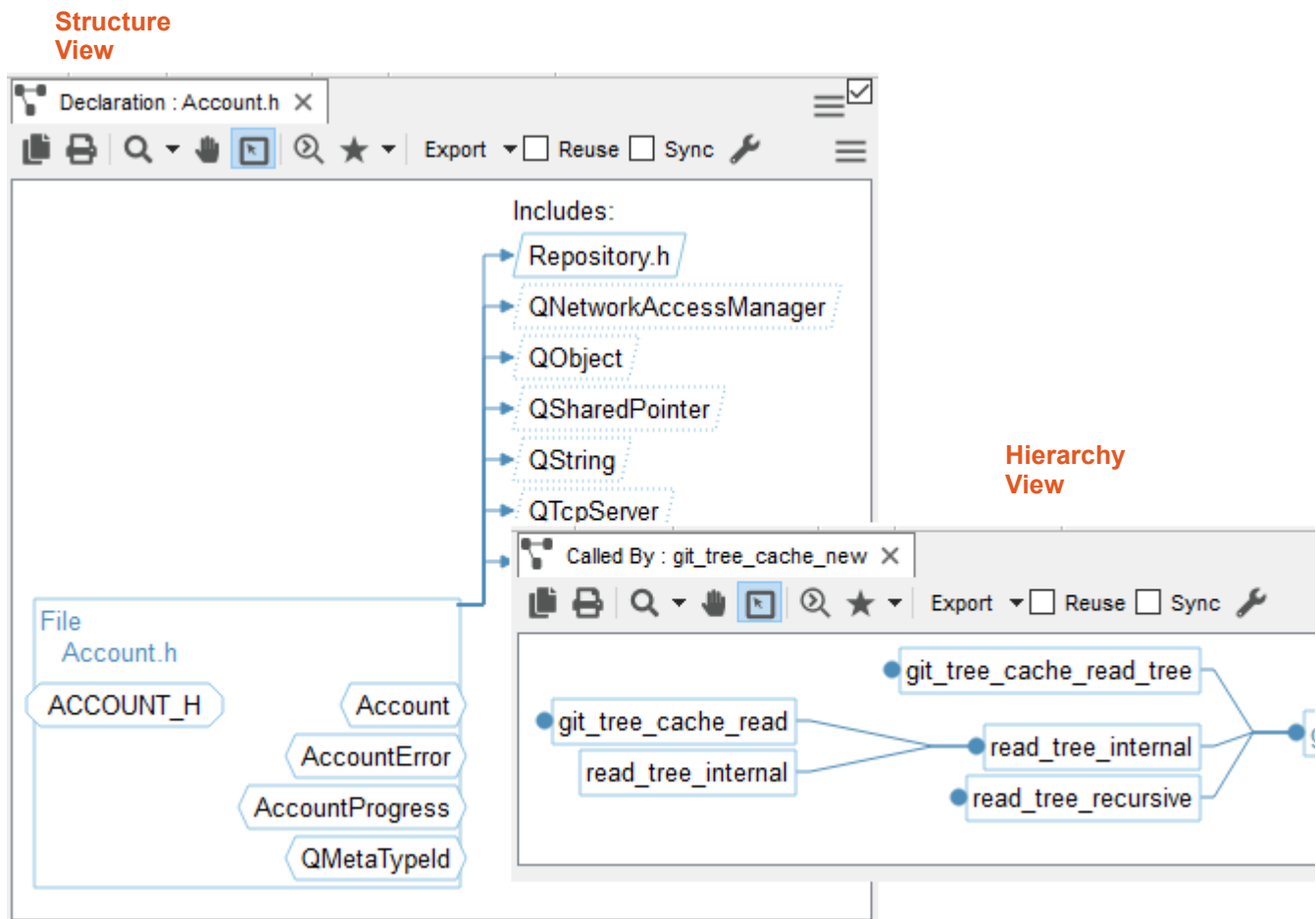
For details, see *About Architectures* on [page 210](#).

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 257](#).

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.

See *APIs* on [page 343](#) for more about the available APIs.

This manual doesn't provide detailed documentation for the APIs. Examples, templates, and documentation are available in the `plugins` directory in your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>. Find documentation for the APIs in the following locations:

- **Python API:** Choose **Help > Python API Documentation** or see the `doc/manuals/python` subdirectory of the *Understand* installation.
- **Java API:** See the `doc/manuals/java` subdirectory of the *Understand* installation.
- **Perl API:** Search our Support website for "Perl".

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	35
Creating a New Project	37
Project Configuration Dialog	46
Languages Category	48
Files Category	49
File Type Options	54
File Options	55
Report Options	56
Imports Options	59
History Options	61
Search Options	62
Dependencies Options	62
Metrics Options	63
Portability Options	66
Ada Options	68
Assembly Options	72
Visual Basic (.NET) Options	74
C/C++ Options	75
C# Options	87
Fortran Options	89
Java Options	92
JOVIAL Options	94
Pascal Options	96
Python Options	98
VHDL Options	99
Web Options	100
Analyzing the Code	102

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 are using Windows, you can right-click on a `<project>.und` folder and choose **Open with Understand** to open the project in *Understand*.

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 by default. 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\Roaming\SciTools\Db`
- **Linux/Unix:** `~/.local/share/Scitools/Db`
- **MacOS:** `~/Library/Application Support/Scitools/Db`

The location of these files can be changed using the **Tools > Options** command and changing the location of the **Settings Folder** in the General category (see *General Category* on [page 109](#)).

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. If you are using the New Project Wizard to create a project, check the **Store analysis data in project folder** box in the **Set Location** page of the wizard to force all project data to be stored in a “local” folder within the project folder.

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.

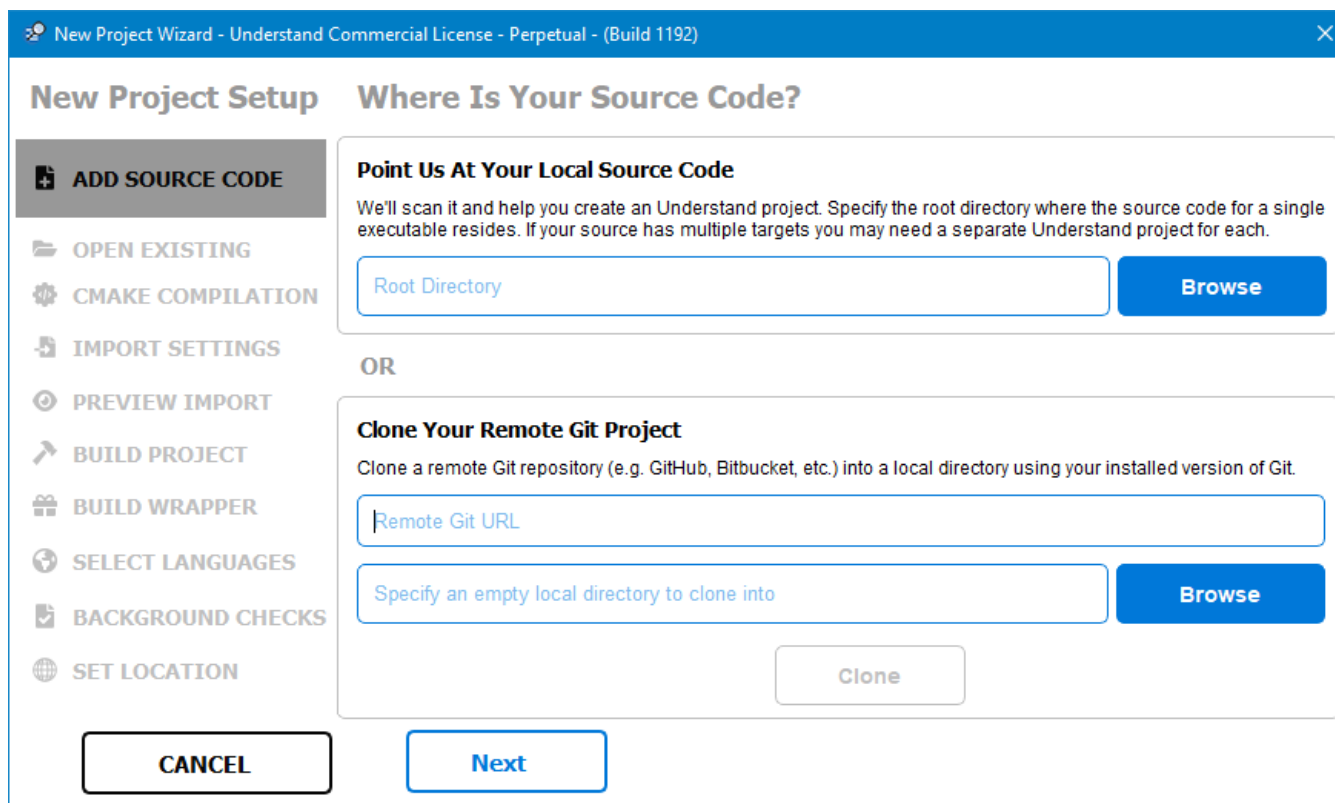
Sample projects that you download and open from the Welcome page are typically stored in the following location:

- **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.

Click the **New** link in the Welcome page you see when you start *Understand*. Or choose **File > New > Project** from the menus or use the  New Project icon in the main toolbar. By default, this opens the New Project Wizard.



Alternately, you may have disabled the option to run this wizard (see [page 111](#)), 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 *Project Configuration Dialog* on [page 46](#).

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 356](#).

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

Add Source Code Page

In the **Add Source Code** page of the wizard, you can specify the location of your source code using either a local or network directory location or a Git repository to clone to a local directory.

- **Point Us At Your Local Source Code:** Click **Browse** and select the top-level directory where your source code is located. You can select only one root directory here. After the project is created, you can exclude portions of that directory or add additional directories. If you already have a cloned Git repository in a local directory, you can select it here, and additional options related to Git will be provided by the New Project Wizard; see *Creating Projects for Git Repositories* on [page 42](#).
- **Clone Your Remote Git Project:** If you use Git to manage your source code but do not have a local clone of the repository that you want *Understand* to analyze, paste the URL for your Git project repository and browse for a local directory into which you want to clone the Git repository for this *Understand* project. See *Creating Projects for Git Repositories* on [page 42](#) for additional steps. Then click **Clone**.

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

The other prompts that occur after this point depend on the contents found in your source directory. Answer the prompts as appropriate and click **Next** or **Skip**.

Open Existing Page

If an *Understand* project already exists within this location, you see prompts asking whether you want to open that project in the **Open Existing** page of the wizard.

CMake Compilation Page

If the source files use CMake, you are asked if you want to import information from the CMake compilation database file in the **CMake Compilation** page of the wizard. See *Creating Projects from CMake Projects* on [page 44](#).

Import Settings and Preview Import Pages

- If a Visual Studio project exists within the location, you are asked if you want to import it to create a more accurate configuration in the **Import Settings** page of the wizard. See *Creating Projects from Visual Studio Projects* on [page 45](#).
- If an Apple Xcode project exists within the location, you are asked if you want to import it to create a more accurate configuration in the **Import Settings** page of the wizard. See *Creating Projects from Apple Xcode Projects* on [page 45](#).
- If your source code directory contains a project file created by Green Hills (*.gpj), Renesas (*.rcpe), or Keil Uvision (*.uvprojx) software, *Understand* asks if you want settings from that project file to be imported. Click the **Import** button in the **Import Settings** page of the wizard to see a preview of the files and settings to be imported. If there are multiple project configurations in the project file, you can select the configuration to import.

Build Project Page

If build files exist in the directory, the **Build Project** page of the wizard lets you build your project with *Understand* monitoring the build. This allows *Understand* to create a more accurate project by watching include paths and macros each file sends to the compiler. Run your build in a separate terminal window or other application while this page of the wizard is shown. See *Using Build Watcher to Create Projects* on [page 43](#).

Build Wrapper Page

If you have source code that uses gcc/g++, the **Build Wrapper** page explains how Buildspy can create *Understand* projects from your source code by gathering information about files, includes, and macros.

See *Using Buildspy to Build Understand Projects* on [page 356](#) for steps to add Buildspy to an existing *Understand* project. For more information about using Buildspy, use the `buildspy -help` command and see the Buildspy topic on the [Support website](#).

Commercial License - Perpetual - (Build 1192) ×

Other Import Options

Wrappers can capture project information from your existing build system:

BuildSpy

Buildspy, a command line wrapper bundled with Understand, empowers gcc/g++ users to effortlessly create Understand projects while building. Designed for any compiler with gcc-like syntax, it seamlessly extracts file lists, includes, and macros, hastening project setup and enhancing precision. Explore the Buildspy segment in Chapter 14 of the Understand help manual or [this support article](#) for setup guidance.

Back

Next

Select Languages Page

The **Select Languages** page of the wizard detects languages used in your source code. You can disable languages using the checkboxes on the left. (If you want to enable another language later, you can do that in the Project Configuration settings.)

Summary of Project Languages

Languages & Compilers

Select the languages you want to include.

LANGUAGE	FILES	COMPILER
<input checked="" type="checkbox"/> C++	70	MSVC ▾
<input checked="" type="checkbox"/> Web	2	PHP 5.3 ▾

For most languages, you can select the version of the compiler to use as the basis for analyzing the code from the drop-down list on the right. Then click **Next**.

Background Checks Page

In the **Background Checks** page, you can enable a set of CodeCheck checks to be run in the background whenever you analyze the project. The recommended checks are listed in this New Project Wizard page. For additional checks and more information, see [Chapter 12, Using CodeCheck](#). See *Analyzing the Code* on [page 102](#) for information about background processing.

Summary of Project Checks

Background Checks

To continuously ensure code quality, we recommend selecting these checks to run in the background each time you analyze the project.

- ▼ ☐ Recommended
 - > ☐ Compiler Warnings
 - ☐ Assert Side Effect
 - ☐ Branch Clone
 - ☐ Copy Constructor Init
 - ☐ Delete Null Pointer
 - ☐ Infinte Loop
 - ☐ Macro Side Effects
 - ☐ Missing Null Terminator
 - ☐ Redundant Condition
 - ☐ Special Member Functions
 - ☐ Unnecessary Friends
 - ☐ Unused Entities

MISRA, AUTOSAR, HIS, SEI-CERT and more are also supported and can be selected at any time by going to Checks->Select Checks after project creation.

CodeChecks can be run a single time or set to run in the background.

Back

Next

Set Location Page

- 1 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, use the **Set Location** page of the wizard to browse to select another location. See [page 35](#) for more about how projects are stored.

Set Location & Advanced Settings

Project Location

Where would you like to save this project?

E:\samplecode\zlib\zlib1.und

Browse

Optional

- ☐ Configure advanced settings after project creation
- ☐ Enforce portability rules to share this project with others
- ☐ Store analysis data in project folder

Back

Create Project

- 2 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 *Project Configuration Dialog* on [page 46](#) for details.
- 3 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 rules to share this project with others** box. See *Portability Options* on [page 66](#) for details.
- 4 If you want project analysis data files to be stored in the same folder as the project, check the **Store analysis data in project folder** box. See *Project Storage* on [page 35](#) for details.
- 5 Click **Create Project**. The new project is created and opened.

Creating Projects for Git Repositories

If you use Git for version control, additional features related to Git—such as viewing line-specific history, comparisons between commits, and filtering uncommitted changes—are provided when you create an *Understand* project using the New Project Wizard.

Note: If you want to see git error messages generated by actions within *Understand*, define the `STI_GIT_ERRORS` environment variable.

Follow these additional steps in that wizard:

- 1 If you have **not yet cloned your Git repository**, provide the URL for the Git repository and a local directory, and *Understand* will clone the repository. (You must have access to the repository and have Git software installed and configured.) The “git clone” command is performed to the specified local directory when you click **Clone**. A progress bar will show how much of the clone action is complete. If cloning the repository fails, the error message is shown.
- 2 If you have **already cloned your Git repository** to a local directory, use the “Point Us At Your Local Source Code” section of the first page of the wizard to browse to the root directory of your cloned repository. If you point to a local directory that contains a `.git` folder at the top level, you can optionally set a commit to freeze the *Understand* project at and an older commit to use for comparisons:

Where Is Your Source Code?

Point Us At Your Local Source Code

We'll scan it and help you create an *Understand* project. Specify the root directory where the source code for a single executable resides. If your source has multiple targets you may need a separate *Understand* project for each.

D:\mycode\gitproject



Git Options

Freeze this project at a specific point in time.

Freeze at commit hash

Browse

Use an older commit to create a second project for historical comparisons.

Comparison commit hash

Browse


- **Freeze this project at a specific point in time:** If you want *Understand* to analyze a particular commit version in the Git repository, click the **Browse** button next to this field and select the comment for the commit you want to analyze (or paste the hash string for a Git commit). This setting can be made only when the project is created; to analyze a different commit, create a new *Understand* project. This setting affects only the *Understand* project that is created; it does not affect the Git repository in any way.

- **Use an older commit to create a second project for historical comparisons:**
If you want to compare the current state of the repository to another commit version, click the **Browse** button next to this field and select the comment for the commit you want to compare to (or paste the hash string for a Git commit). This setting may be changed at a later time for an *Understand* project by choosing **Compare > Comparison Projects** (see *Comparing Projects* on [page 320](#)).
- 3 After you have specified the Git settings you want to use, click **Next**.
 - 4 Continue using the New Project Wizard as described in *Creating a New Project* on [page 37](#).
 - 5 Most Git-related features, such as displaying “blame” information in the Source Editor windows ([page 326](#)), work automatically if you create a project as described here. Confirm that the cloned Git directory is also specified in the *History Options* on [page 61](#) in order to use Git filtering with CodeCheck ([page 295](#)).

Using Build Watcher to Create Projects

When you import a project that contains C/C++ code, the **Build Project** page of the New Project Wizard asks if you want to create a more accurate project by allowing *Understand* to watch your build process. *Understand* does this by running its Build Watcher tool to analyze information about include paths and macro definitions that are passed to the compiler.

To use this feature, run your build in a terminal window or other application while the **Build Project** page of the New Project Wizard is open. It is best to first clean your build or delete the build directory 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 Build Watcher is collecting compiler actions.

Build Watcher detects compilers with a command of `c++`, `cc`, `cl`, `clang`, `clang++`, `g++`, `gcc`, and `nvcc`. If you run your compiler with a different command, click the  wrench icon and add your compiler command before starting the build.

To run the similar tool, Buildspy, from the command line, see *Using Buildspy to Build Understand Projects* on [page 356](#).

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 37](#) 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 (`command_compile.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`)

- 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 39](#) for the remainder of the project creation process.

To use 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 37](#) 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**. See [page 39](#) for the remainder of the project creation process.

Understand supports the following versions of Visual Studio:

- **C/C++:** Visual Studio 97 through Visual Studio 2022 for C/C++
- **C#:** Visual Studio .NET 2002 through Visual Studio 2022
- **Visual Basic:** Visual Studio .NET 2002 through Visual Studio 2022

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 37](#) 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**. See [page 39](#) for the remainder of the project creation process.

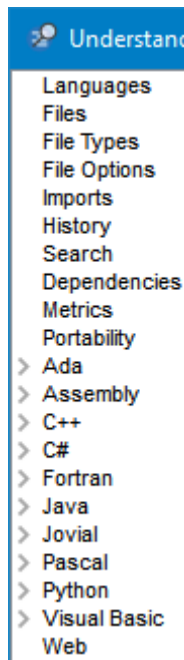
For more information about using the *Understand* with Xcode, see the [Support website](#).

Project Configuration Dialog

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

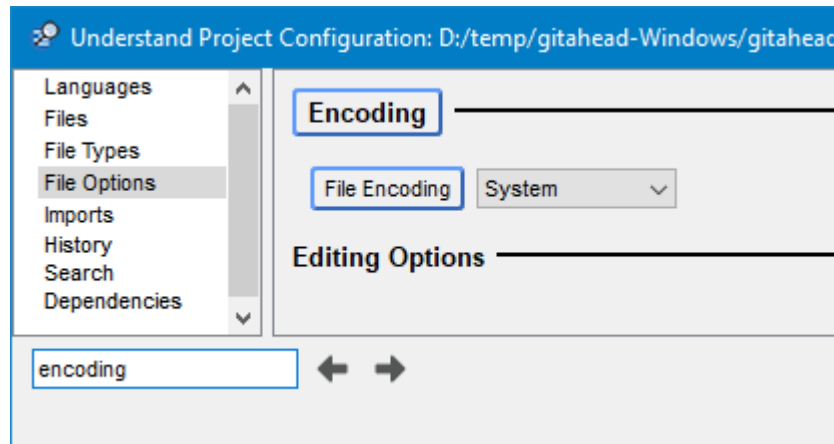
The categories on the left in the Project Configuration dialog allow you to specify various project settings to be used during analysis. The Project Configuration dialog contains the following categories:

- **Languages:** Set the types of languages to be analyzed. See [page 48](#).
- **Files:** Set the locations of source files to be analyzed. See [page 49](#).
- **File Types:** Set how to handle source file types and what file extensions are used. See [page 54](#).
- **File Options:** Set the file encoding and editing mode for source files. See [page 55](#).
- **Imports:** Specify import file and exclude filters. See [page 59](#).
- **History:** Select a Git repository and/or previous project for history information and entity comparison. See [page 61](#).
- **Search:** Update the index used for Instant Search. See [page 62](#).
- **Dependencies:** Set options related to the Dependency Browser and dependency graphs. See [page 62](#).
- **Metrics:** Set options related to Metrics for this project. See [page 63](#).
- **Portability:** Select project portability options. See [page 66](#).
- **Language-Specific Options:** Set options for the languages you selected in the Languages category. For details, see:
 - Ada Options, [page 68](#)
 - Assembly Options, [page 72](#)
 - Visual Basic Options, [page 74](#)
 - C++ Fuzzy Options, [page 75](#)
 - C++ Strict Options, [page 84](#)
 - C# Options, [page 87](#)
 - Fortran Options, [page 89](#)
 - Java Options, [page 92](#)
 - JOVIAL Options, [page 94](#)
 - Pascal Options, [page 96](#)
 - Python Options, [page 98](#)
 - Web Options, [page 100](#)

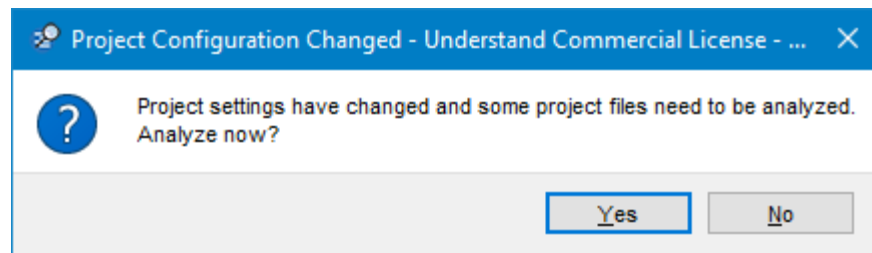


For advice about ways to adjust the project configuration to improve the accuracy of project analysis, see the [SciTools website](#).

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.



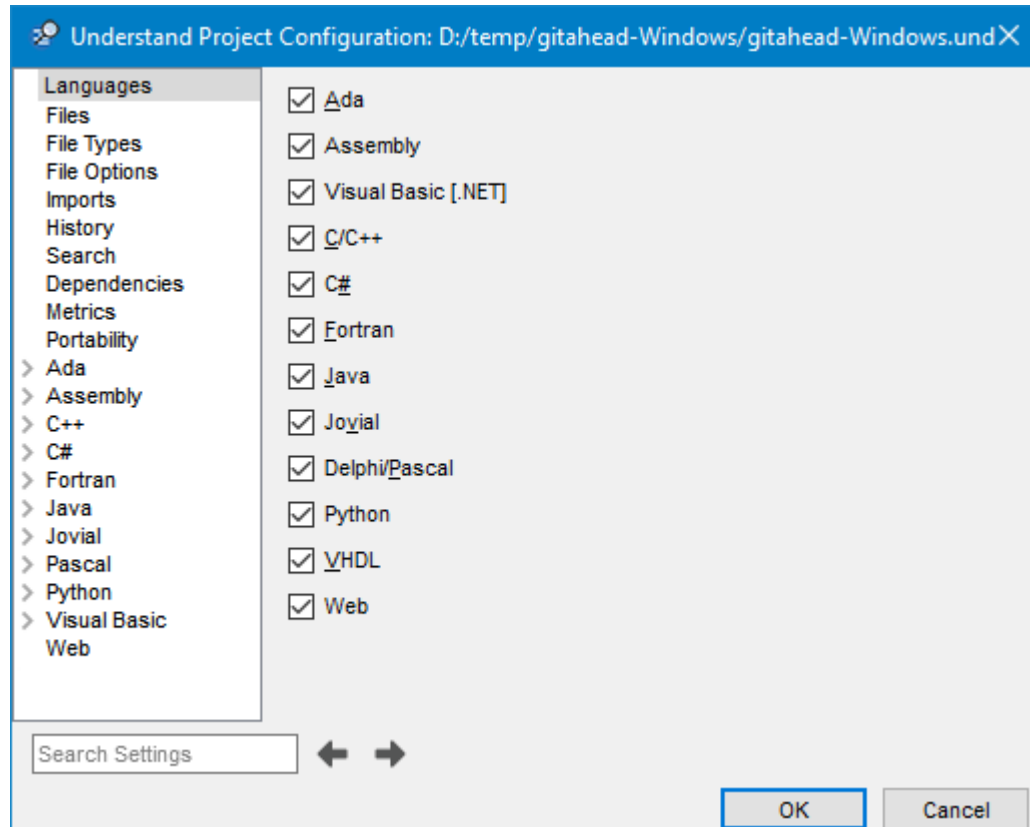
Click **Yes** and *Understand* begins analyzing (parsing) the code ([page 102](#)).

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.

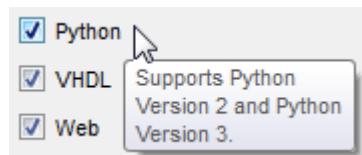
If you want to make a copy of the current configuration, for example to create two variants of one configuration, you can make a copy of the directory tree where your project settings are stored ([page 35](#)).

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.

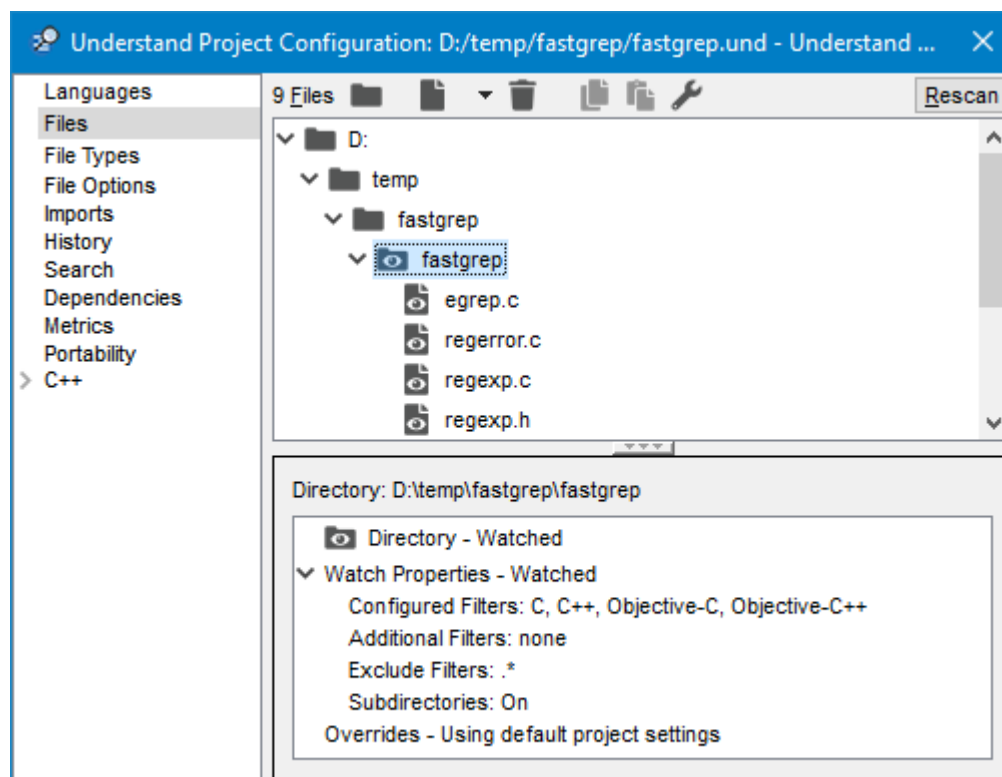


When you select a language, categories for that language are added to the list on the left in the Project Configuration dialog. The languages you choose here not only affect how the source files are analyzed. They also affect the filter types available and the metrics available.

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 44](#), *Creating Projects from Visual Studio Projects* on [page 45](#), and *Creating Projects from Apple Xcode Projects* on [page 45](#).

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.

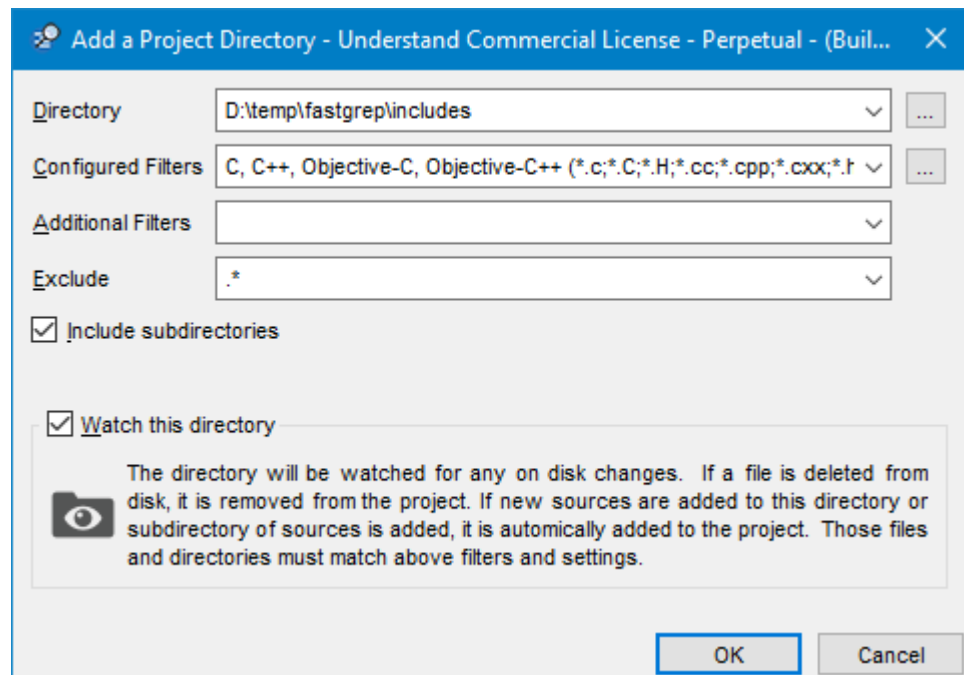
Click **Rescan** if you have added files to a directory that are not shown in this dialog.

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 Project 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 project files. Additional languages beyond those supported by *Understand* are listed in the Languages category, including COBOL, MSDos Batch, Perl, Plm, SQL, Tcl, Text, and Verilog. For languages that are not supported by *Understand*, syntax highlighting is provided, but entities in the code are not analyzed.

If this directory contains source files with extensions that are not listed, click **Configure**. Also, see *File Type Options* on [page 54](#). 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 52](#) 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  icon, you can choose to import a text file that contains a list of source files to import. For example, you might generate such a file from a compiler application or code management system. The file should contain one absolute file path per line. See *Adding Files to a Project* on [page 349](#) 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.

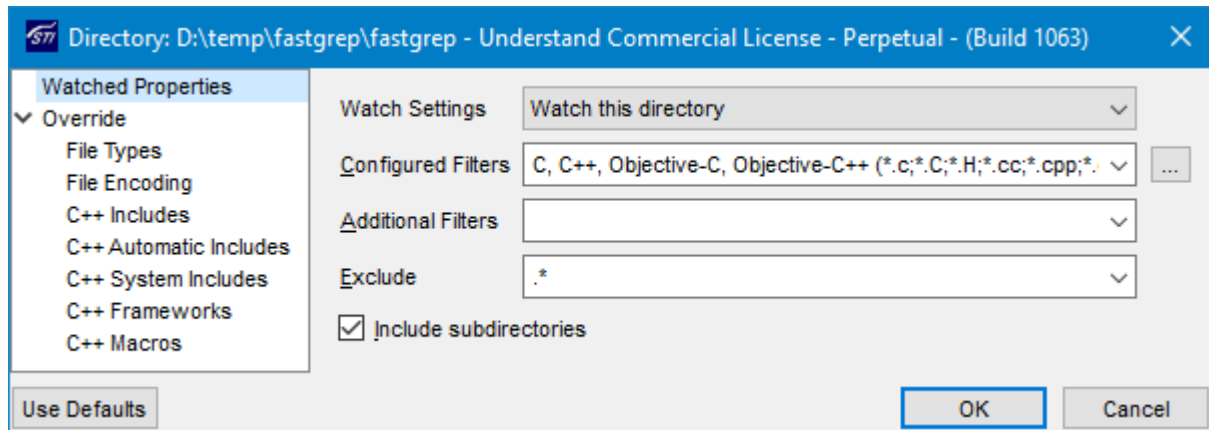
You can right-click on a removed file or directory and choose **Restore Selected Files** to re-add it to the project.

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:

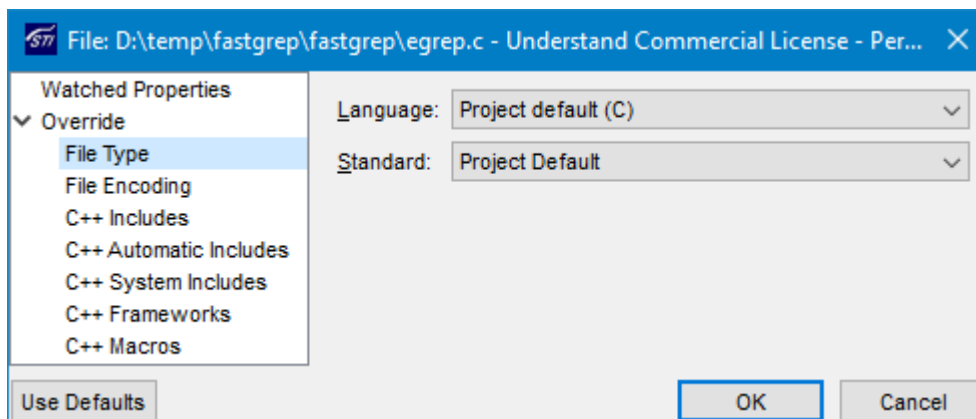
- 1 Select a directory.
- 2 Click  or right-click and select **Configure override settings**.






- 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 controls what types of new and deleted files will be found as described in *Adding Directories* on [page 50](#).
- 4 The **File Type** category lets you override the languages used for files in this directory as described in *File Type Options* on [page 54](#). The **File Encoding** category lets you override the encoding setting described in *File Options* on [page 55](#).
- 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 68](#) through [page 100](#) 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**.



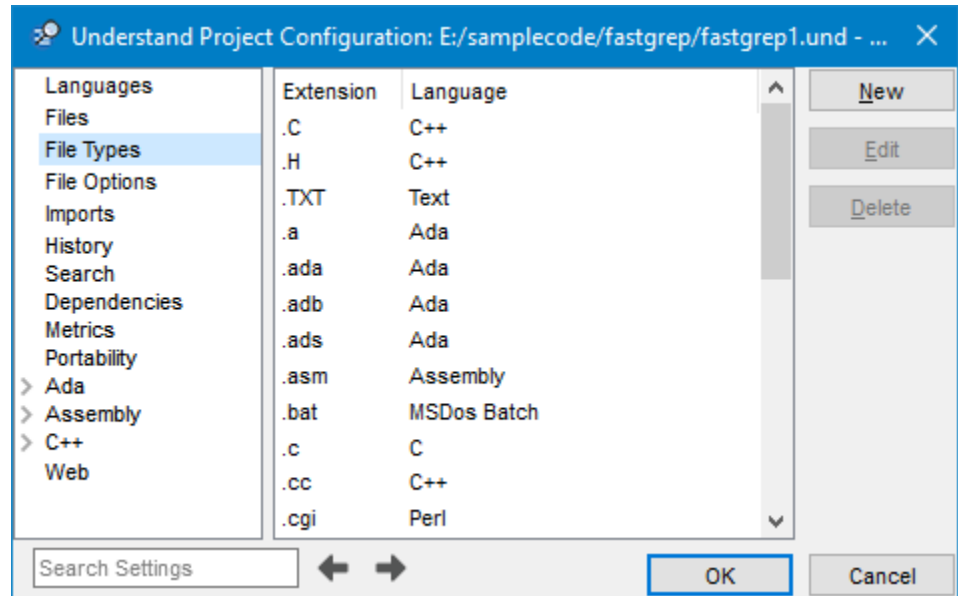
- 3 In the **Watched Properties** category, you can choose **Watched by parent directory** to have the file watched if the parent directory is watched or Exclude from watch.
- 4 The **File Type** category lets you override the languages used for the file as described in *File Type Options on page 54*. The **File Encoding** category lets you override the encoding setting described in *File Options on page 55*.
- 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 68](#) through [page 100](#) 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

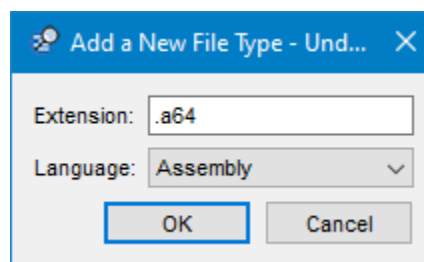
In the **File Types** category of the Project Configuration dialog, you can control how file extensions are interpreted by *Understand*.



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**.



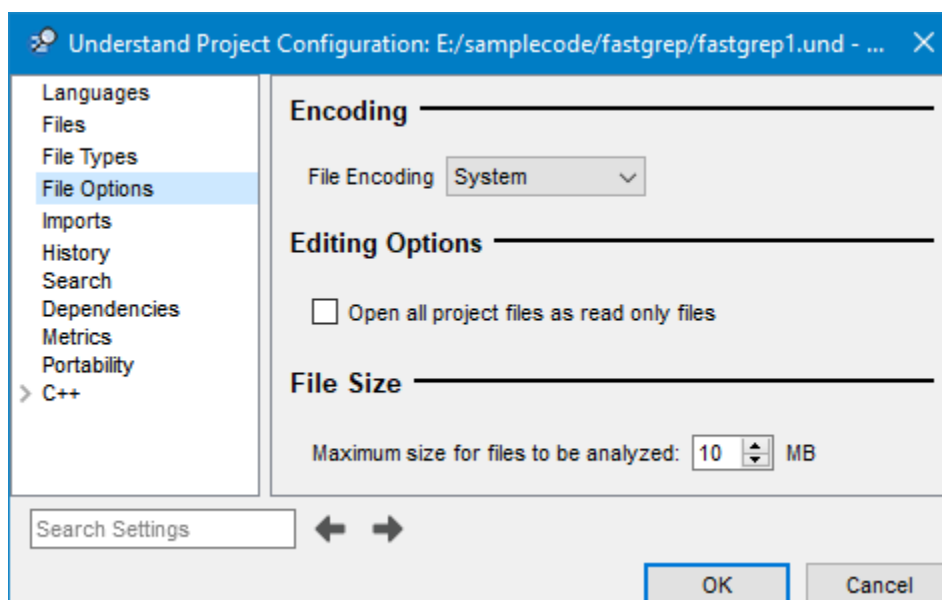
The file extension should begin with the period. It can contain simple * and ? wildcards.

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 52](#)).

File Options

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

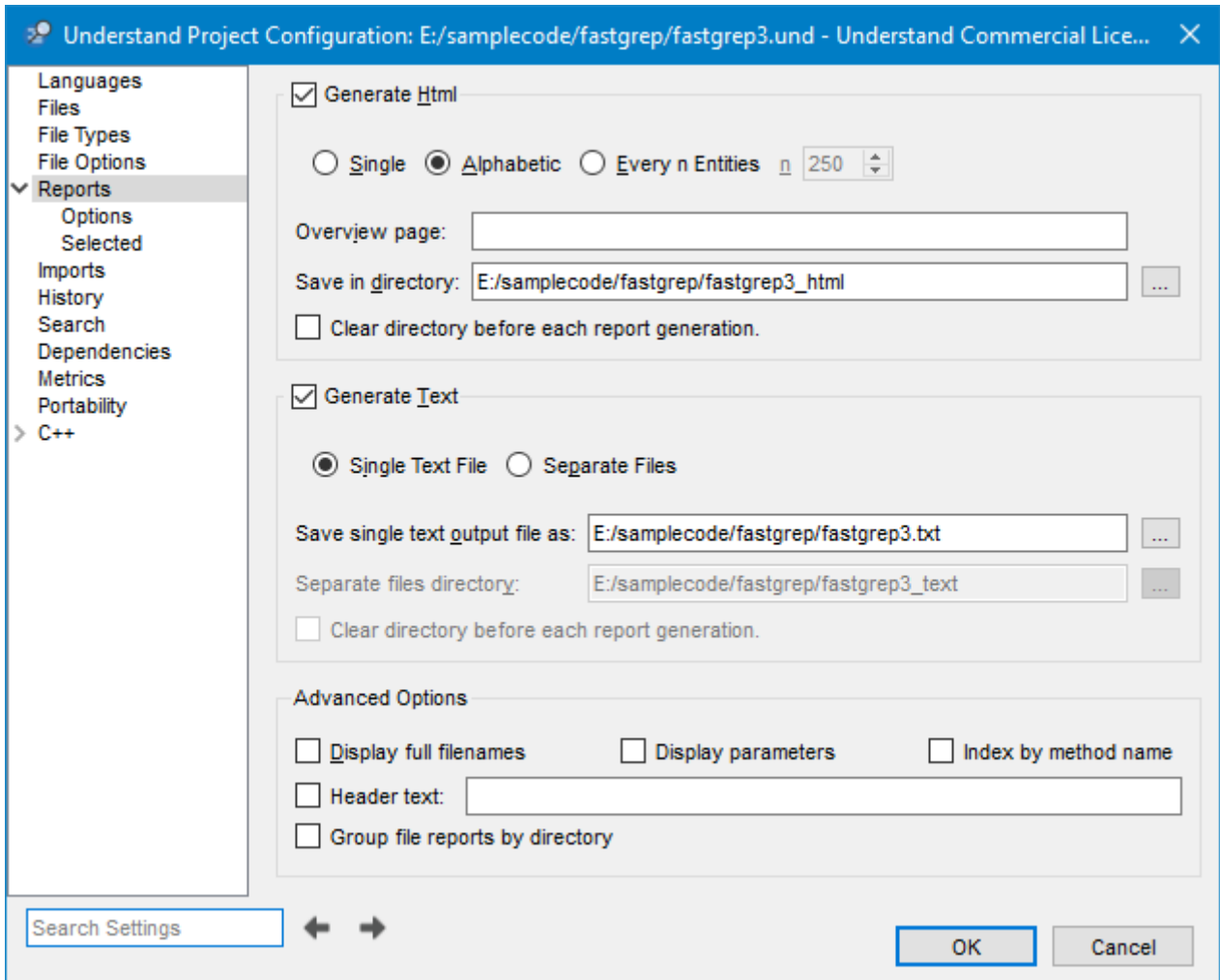


- **File Encoding:** Select the type of encoding to use when saving source files for this project. Many encoding formats are supported. You should change this setting at the project level if you want it to be different than the setting for other projects or if your other applications have problems opening or displaying files saved by *Understand*. See *Editor Category* on [page 121](#) 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 52](#)).
- **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.

Report Options

Note: Access to HTML Reports is controlled by your *Understand* license. Please contact our support department if you would like to enable this feature.

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



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

You can control the colors and font styles in HTML reports as described in *Customizing Report Colors* on page 226.

You can choose to generate reports in either or both HTML and text formats. For information about generating reports, see *Generating Reports* on page 226.

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

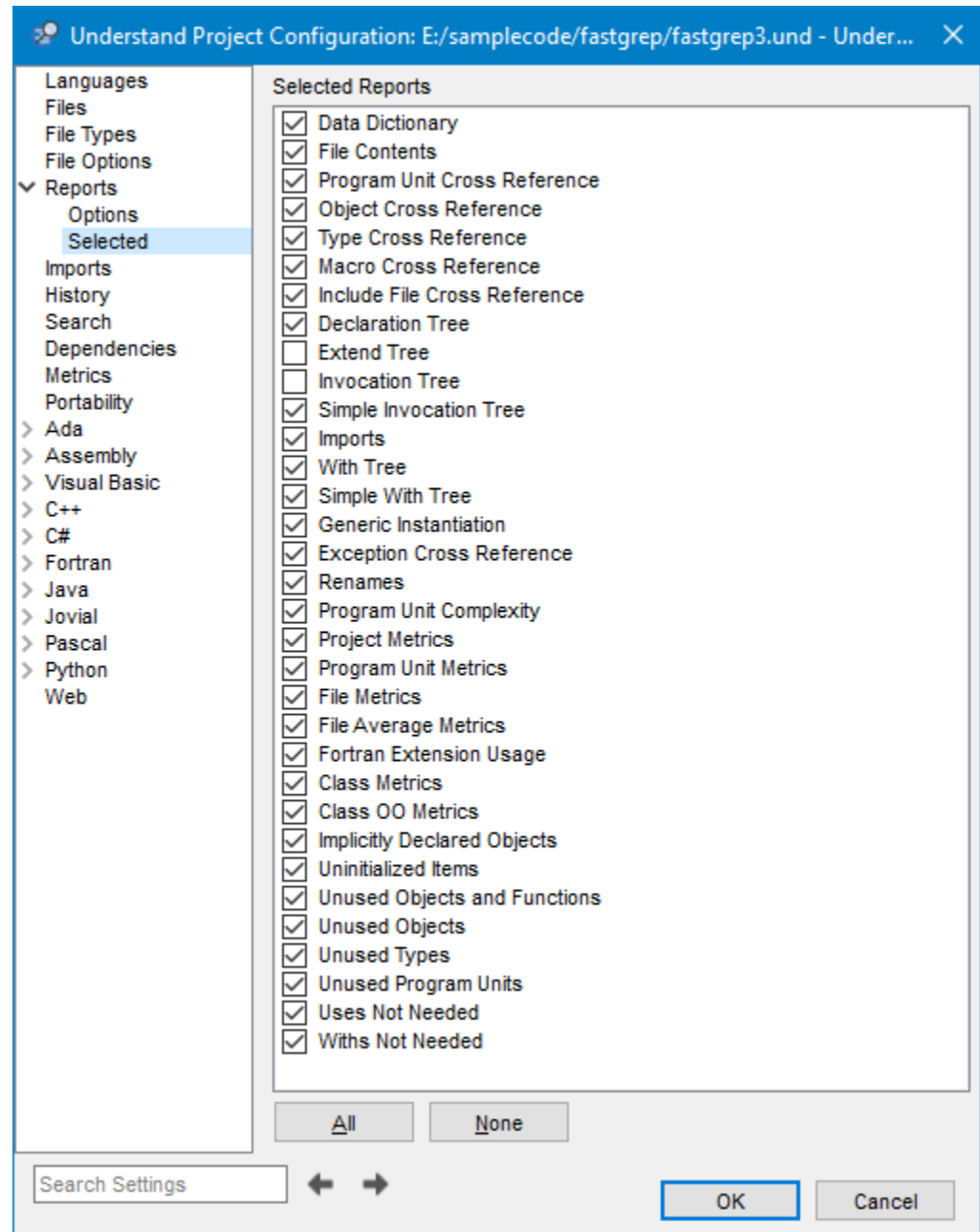
- **Generate HTML:** Checking this option causes the report generation to create a large group of HTML files that are interlinked.
 - You may generate **Single** or multiple HTML files for each report type. It is recommended that you split up the files for large projects. Choose **Alphabetic** to generate multiple HTML files per report that are split up alphabetically by the first letter of the entity name. Choose **Every n Entities** to generate multiple HTML files per report that are split up every “n” number of entities. By default, a single HTML file is generated for each letter of the alphabet.
 - The “home” page for reports is `index.html`. You can select an alternate **Overview Page**.
 - The default **Save in directory** is the `<proj_file>_html` folder within the top level source code directory for the project, but you can select an alternate location.
 - You can choose to clear the previously generated reports and anything else in the selected directory at the beginning of the report generation by checking the **Clear directory before each report generation** box.

If you check Allow Scripts, the reports contain JavaScript code.

- **Generate Text:** Checking this option causes the report generation to create simple text files containing the report data.
 - You may generate one text file of the specified location and name (by choosing **Single Text File**). Alternately, you may generate multiple text files (by choosing **Separate Files**) and specify a directory to contain all the files. The file extensions of each text file will denote the separate reports.
 - Depending on which option you select, you can also select either a file or directory location for the output.
 - If you choose to create separate files in a directory, you can choose to clear the previously generated reports and anything else in the selected directory at the beginning of the report generation by checking the **Clear directory before each report generation** box.
- **Advanced Options:** Checking this option causes the report generation to create simple text files containing the report data.
 - **Display full filenames:** If you check this box, the invocation tree and metrics reports show full entity names. The default is to use short names.
 - **Display parameters:** If you check this box, reports that list the names of functions and similar entities also include any list of parameters declared for that function.
 - **Index by method name:** If you check this box, entities are sorted in the data dictionary, index, and reports by their short names, rather than full names (for example including the class path).
 - **Header text:** To add an extra header to both HTML and text reports, check this box and type the heading to add.
 - **Group file reports by directory:** By default, the text-based Files report is sorted according to filename. If you want to group files by the directory in which they are located, check this box.

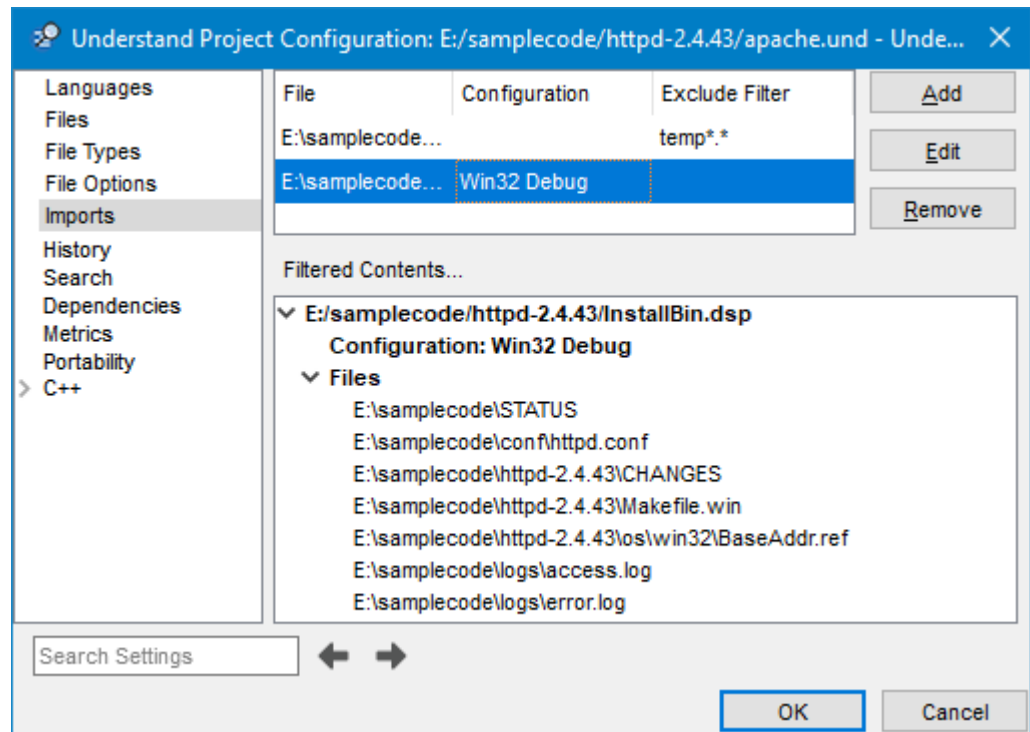
Reports > Selected Category

The **Selected** subcategory lets you check the boxes for the reports you want to generate. The list of reports differs depending on which languages are used in your project. See *An Overview of Report Categories* on [page 228](#) for descriptions of these reports.



Imports Options

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

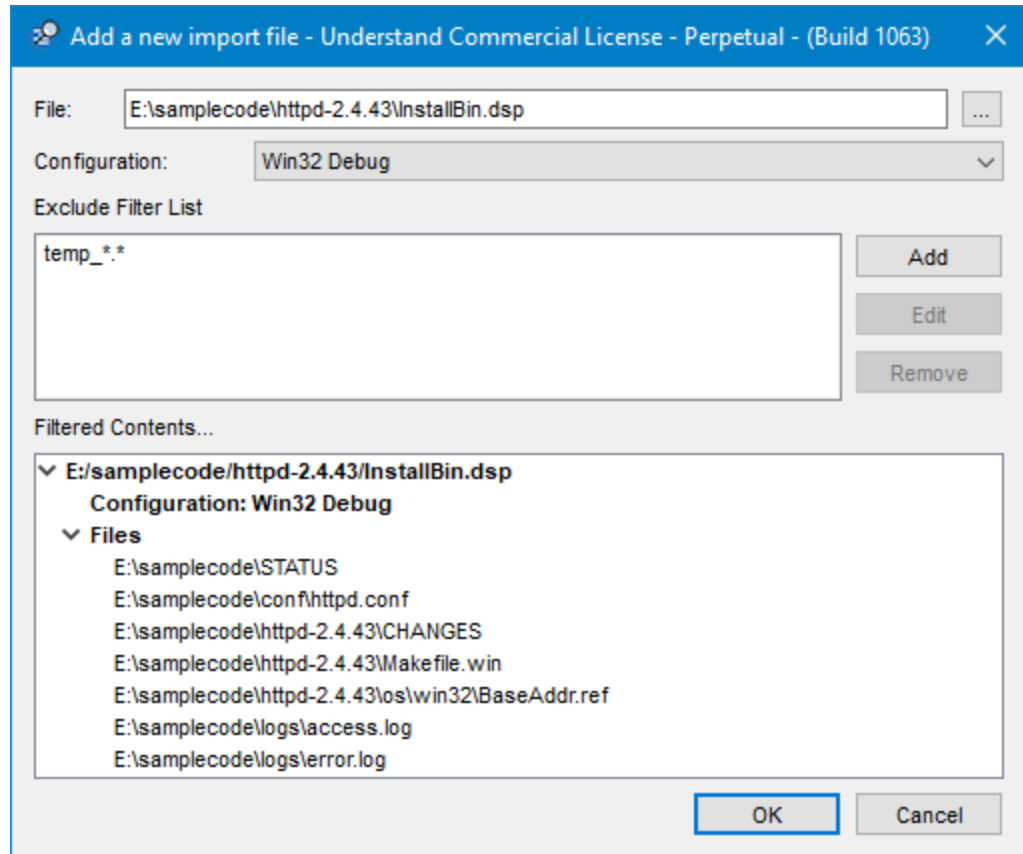


The following file types can be imported:

- **JSON compile command file:** The `compile_commands.json` file is a compilation database, which consists of an array of “command objects.” Each command object specifies one way a translation unit is compiled in the project, including the unit’s main file, the working directory of the compile run, and the actual compile command. You can generate this file using Clang or CMake. If you use GCC/G++, the open source Bear software can be used to generate this file. See *Creating Projects from CMake Projects* on [page 44](#).
- **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 45](#).
- **Apple Xcode project files:** *.xcodeproj. See *Creating Projects from Apple Xcode Projects* on [page 45](#).
- **Keil Uvision project files:** *.uvprojx. See *Creating a New Project* on [page 37](#).
- **Green Hills project files:** *.gpj See *Creating a New Project* on [page 37](#).
- **Renesas Common project files:** *.rcpe. See *Creating a New Project* on [page 37](#).

To import a file, follow these steps:

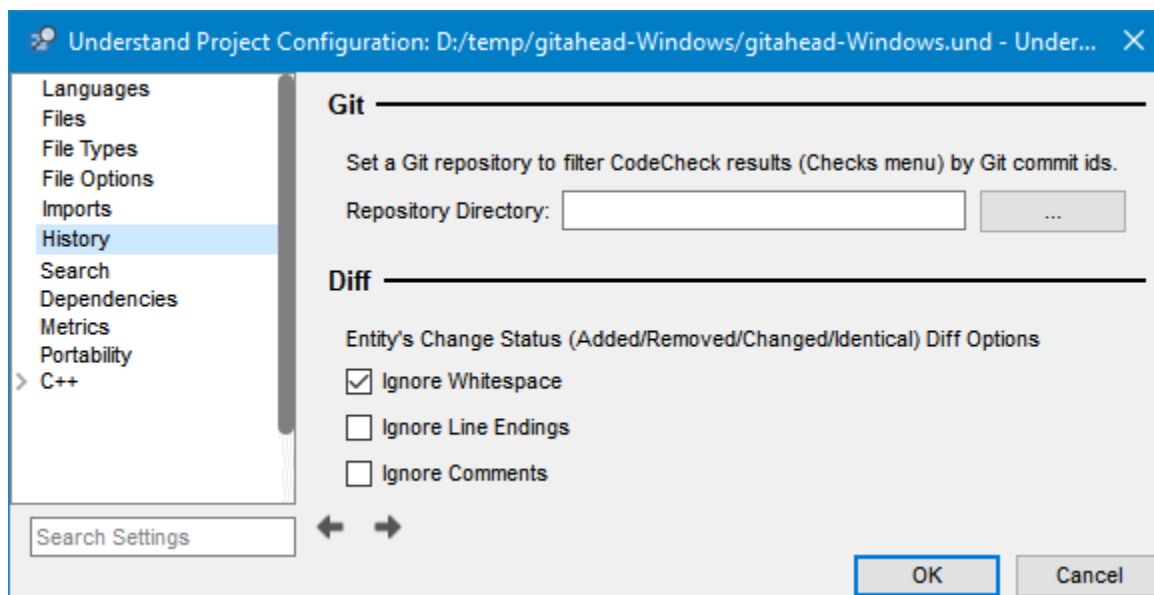
- 1 In the **Imports** category of the Project Configuration dialog, click **Add**.
- 2 Click ... to browse for a file of one of the types listed above.
- 3 Select the file and click **Open**. You should see a list of the project settings in the file you selected.



- 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 historical information from a Git repository (**Git Settings** below) and/or by comparing a previous version of the *Understand* project (**Diff Settings** on [page 61](#)).



Git Settings: If you use Git for version control, create your *Understand* project using the New Project Wizard (see *Creating Projects for Git Repositories* on [page 42](#)). Most Git-related features, such as displaying “blame” information in the Source Editor windows ([page 326](#)), will then work automatically. However, to use Git filtering with CodeCheck ([page 295](#)), you should point to the folder that was cloned from the Git repository in the **Repository Directory** field. Click the ... button and select the top-level cloned folder in your Git repository.

Diff Settings: The **Diff** area of the **History** category in the Project Configuration dialog allows you to control what types of changes are counted as changes to entities in the Changed Entities Locator (see [page 320](#)) and in comparison treemaps (see [page 324](#)). These settings do not affect comparison graphs.

- **Ignore Whitespace:** Check this box to ignore any changes to spacing or tabs in the code of an entity.
- **Ignore Line Endings:** Check this box to ignore any changes to line endings in the code of an entity.
- **Ignore Comments:** Check this box to ignore any changes to comments in the code of an entity.

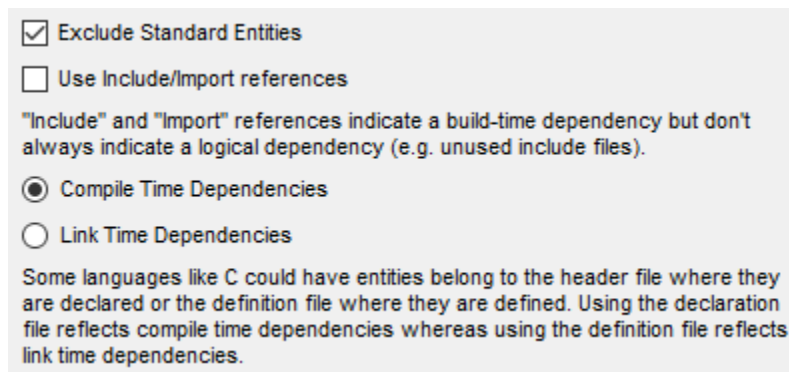
To point to a previous version of your project for comparison graphs, see [page 323](#).

Search Options

If the results of Instant Search (see [page 161](#)) are incomplete or unstable, update the search index by clicking the **Reset Index** button in the Search category of the Project Configuration dialog.

Dependencies Options

The Dependencies category of the Project Configuration dialog lets you set options related to the Dependency Browser ([page 148](#)) and dependency graphs ([page 220](#)).



The screenshot shows a configuration window with the following options:

- ☒ **Exclude Standard Entities**
- ☐ **Use Include/Import references**
"Include" and "Import" references indicate a build-time dependency but don't always indicate a logical dependency (e.g. unused include files).
- ☒ **Compile Time Dependencies**
- ☐ **Link Time Dependencies**
Some languages like C could have entities belong to the header file where they are declared or the definition file where they are defined. Using the declaration file reflects compile time dependencies whereas using the definition file reflects link time dependencies.

- **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 150](#) for more information.

For project-independent options related to dependencies, see *Dependency Category* on [page 120](#).

Metrics Options

The Metrics category of the Project Configuration dialog lets you focus on metrics that matter to you. For large projects, showing many metrics may affect performance when selecting nodes in the Metrics Browser.

For metrics for lines and statements, the following options can be enabled or disabled:

Lines and Statements

- ☒ Show line count metrics
 - For blank, comment, and code line metrics:
 - ☒ Exclude inactive lines
 - ☐ Include inactive lines
 - ☐ Show counts both ways
- ☒ Show statement count metrics
- ☐ Breakdown counts by web language

- **Show line count metrics:** Enables metrics such as Average Blank lines and Comment Lines. If enabled and the project is configured to analyze C/C++ code, you can choose whether to exclude or include inactive lines or show counts for both.
- **Show statement count metrics:** Enables metrics such as Declarative Statements and Empty Statements.
- **Breakdown counts by web language:** Enable to see separate count metrics for languages such as HTML, PHP, JavaScript. This option is available only if the project is configured to analyze Web languages.

For complexity metrics, the following options can be enabled or disabled:

Complexity

- ☐ Show all available cyclomatic complexity metric variants or
 - ☒ Count each decision in a multi-way-decision structure
 - ☐ Count logical conjunctions (OR) and logical and (AND)
- Currently selected cyclomatic metric: Cyclomatic
- Show Additional Complexity Measures:
 - ☒ Maximum Nesting
 - ☐ Paths
 - ☐ Knots
 - ☐ Essential

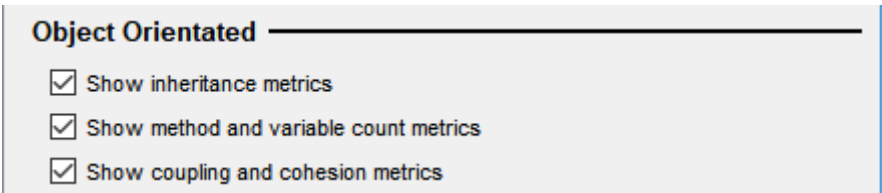
- **Show all available cyclomatic complexity metric variants:** Enable to see the full list of cyclomatic complexity metrics.

- **Count each decision in a multi-way-decision structure** and **Count logical conjunctions (OR) and logical and (AND)**: Enable one or both options to see the following types of cyclomatic complexity metrics:

		Count logical conjunctions (OR) and logical and (AND)	
		On	Off
Count each decision in a multi-way-decision structure	On	CyclomaticStrict	Cyclomatic
	Off	CyclomaticStrictModified	CyclomaticModified

- **Maximum Nesting**: Enable to see the Max Nesting level metric.
- **Paths**: Enable to see the Paths and Paths Log(x) metrics.
- **Knots**: Enable to see Knots, Min Essential Knots, and Max Essential Knots metrics.
- **Essential**: Enable to see the Essential Complexity metrics.

For object-oriented metrics (for projects that use languages that support such features), the following options can be enabled or disabled:



- **Show inheritance metrics**: Enable to see counts of Base Classes and Derived Classes.
- **Show method and variable count metrics**: Enable to see metrics related to methods and variables, such as counts of private, protected, public, class, and instance methods.
- **Show coupling and cohesion metrics**: Enable to see counts such as Coupled Classes or Coupled Packages.

For project and architecture metrics, the following options can be enabled or disabled. These options differ depending on languages used in the project (these examples are for a C/C++ project).

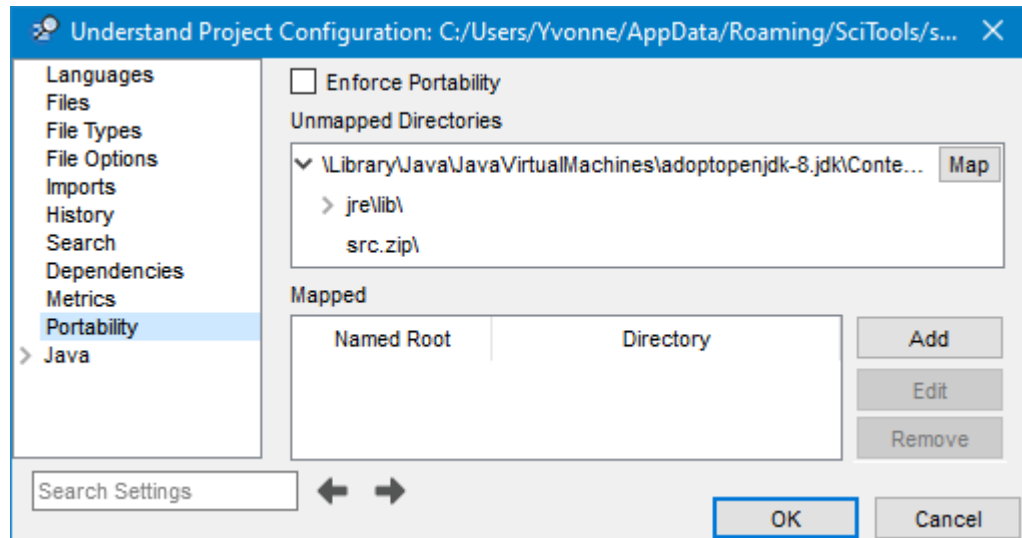


- **Show default summary metrics**: Enable to see basics metrics such as counts of files, functions, and classes. If you disable this option, at least one of the following options in this section must be enabled.

- **Show aggregated file metrics:** Enable to show aggregated metrics available for file entities. Enabling this option shows metrics such as the count of functions and the count of classes, because those are metrics for file entities.
- **Show aggregated function metrics:** Enable to show aggregated metrics available for function entities, such as cyclomatic complexity. Because these metrics require finding all functions in the project/architecture, they can be slower than file metrics.
- **Show aggregated class metrics:** Enable to show aggregated metrics for object-orientated entities. These metrics are the slowest to display because they require finding all classes in the project, and because object-orientated metrics are calculated on demand instead of during project analysis.

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.

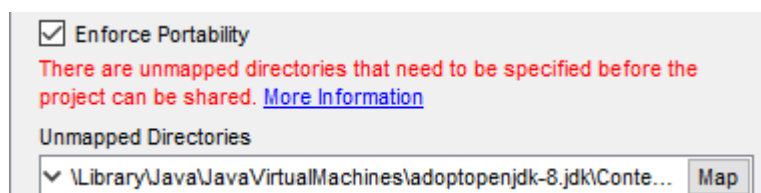
For example, C:\src\proj1 and C:\projects\proj1.und are in different root folders, so the project would use absolute paths. In contrast, C:\src\proj1 and C:\src\projects\proj1.und are in the same root folder (C:\src), so the project would use relative paths.

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.

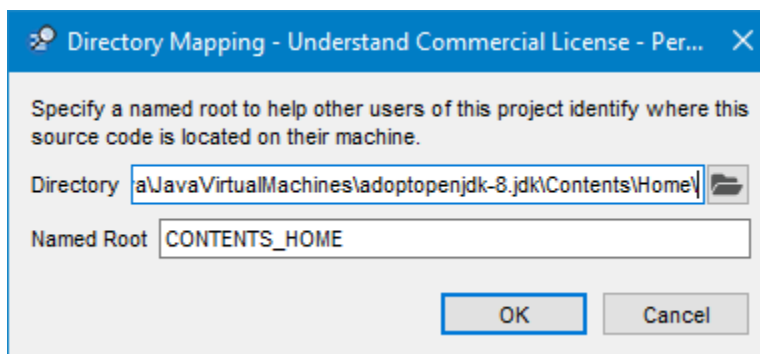
If you check the **Enforce Portability** box, the *Understand* project files will contain relative paths to all directories. These paths are relative to the location of the *Understand* project (*project.und* directory). If you store the project in the source file tree and move it along with the source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



- 3 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.



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

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

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 347](#)). 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 are using the “und” command-line tool, named root definitions on the “und” command line have the highest precedence. For example, the following command line defines the “rootName” environment variable in an *Understand* project:

```
und add -root rootName=c:\mypath\mydir
```

This takes precedence over 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 screenshot shows the 'Understand Project Configuration' dialog box with the 'Ada' category selected in the left sidebar. The 'Options' sub-category is highlighted. The main panel displays the following settings:

- Compiler**
 - Version: Ada05 (dropdown)
 - Preprocessor: None (dropdown)
 - Standard: derstand\ada\ada05 (text field) with a browse button and a Reset button.
- Multiple Language Linkage**
 - The case of externally linkable entities is:
 - ☒ all lowercase
 - ☐ all uppercase
- Metrics**
 - ☐ Count and/or operators in strict complexity
 - ☒ Count exception handlers in complexity
 - ☒ Count for-loops in complexity
- Optimize**
 - ☐ Create and cross reference record object components
 - ☐ Create relations between formal and actual parameters
 - ☐ Less memory usage versus speed
 - ☒ Save comments associated with entities
- Options**
 - Display entity names as: Original (dropdown)
 - ☒ Prompt on parse errors
 - Main subprograms: (text field)
 - Library Directories: (text field) with an Edit button.

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 68) and/or with Ada pragmas.

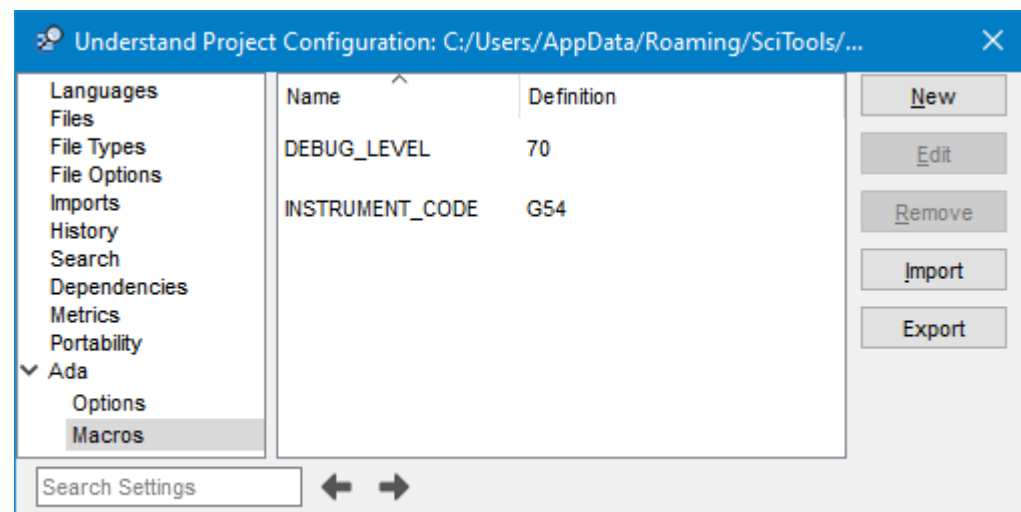
Ada code may contain conditional instructions in pragma statements. For example:

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

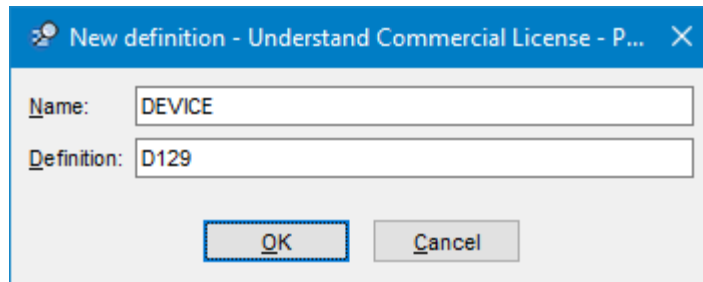
The supported pragmas are IF, IFDEF, ELSIF, ELSE, and ENDIF. These pragmas are similar to preprocessor directives such as `#ifdef` in C code.

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 106 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.

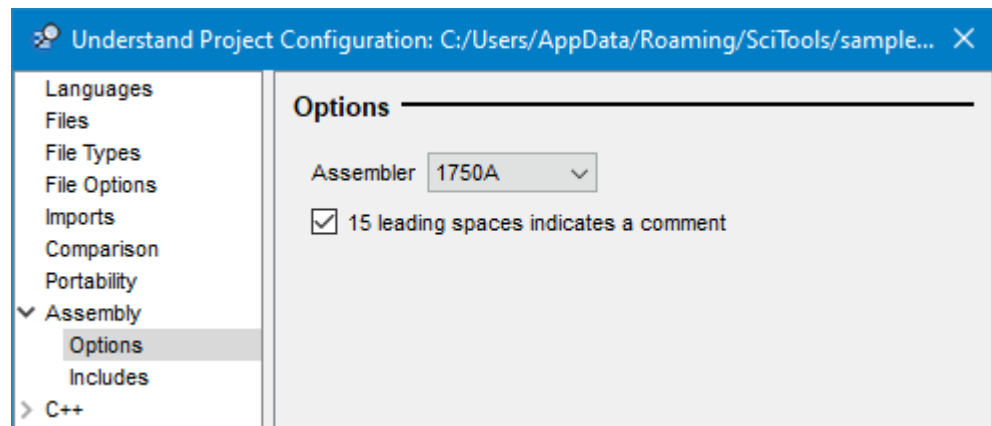
To change the definition of an existing macro without changing the name, select the macro and click **Edit**.

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.

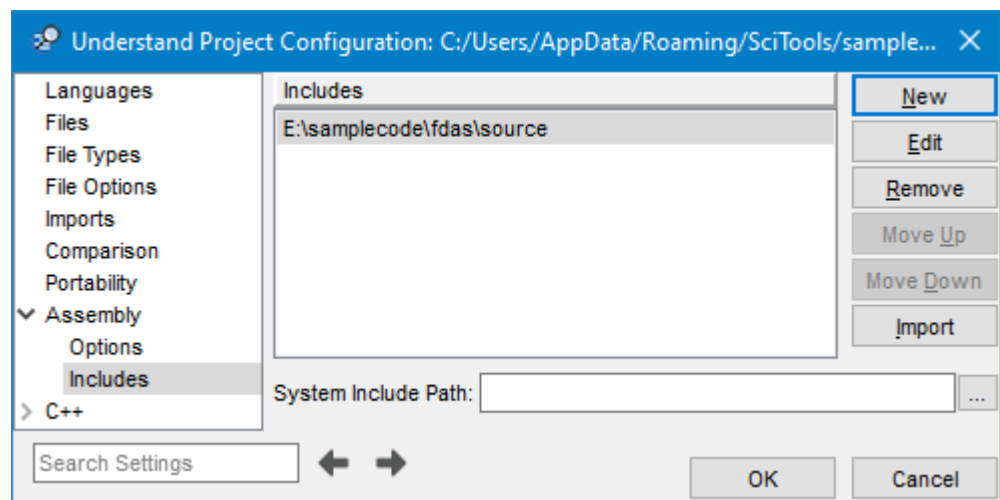


The assemblers supported are:

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

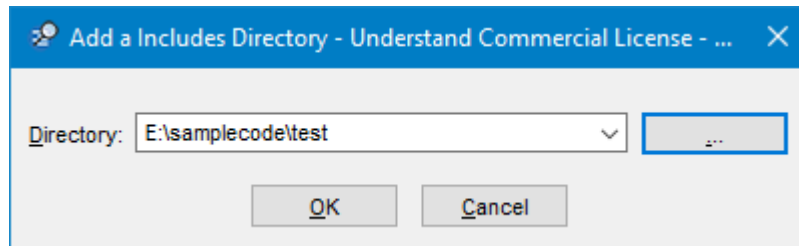
If you select the JIPSE MIL-STD-1750A (1750A) assembler, you can also specify that a line with 15 leading spaces is a comment.

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.

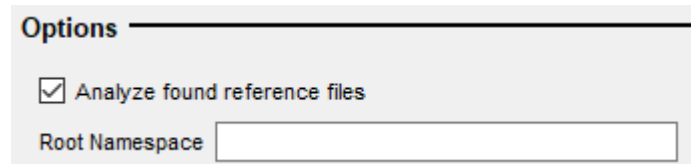
You may use environment variables in include file paths. Use the \$var format on Unix and the %var% format on Windows. You can also use named root in include file paths (see *Portability Options* on [page 66](#)).

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.)

Visual Basic (.NET) Options

Understand supports Visual Studio .NET 2002 through Visual Studio 2022 for Visual Basic.

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.

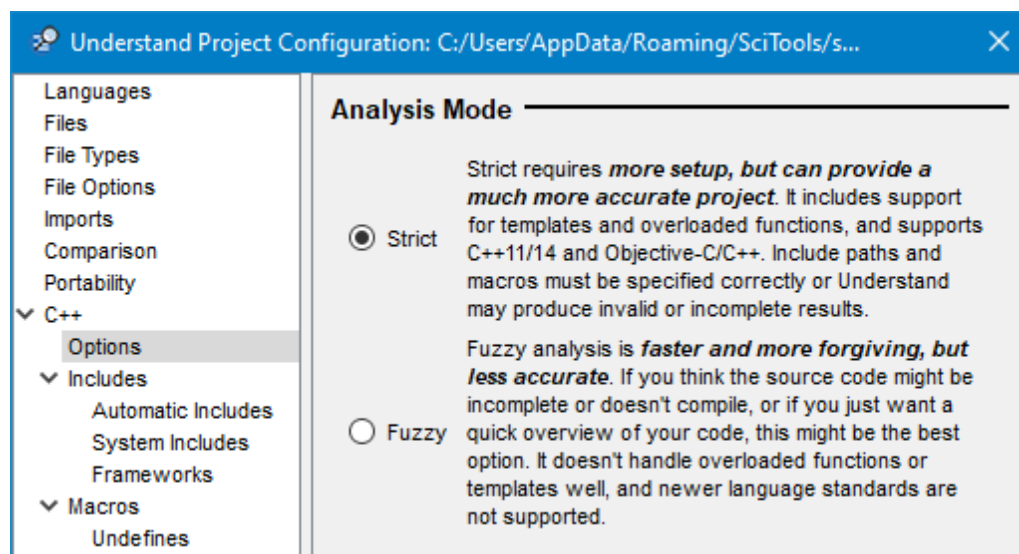
C/C++ Options

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 MacOS and iOS). Include paths and macros must be specified correctly or *Understand* may produce invalid or incomplete results. (CUDA files—.cu and .cuh—may also be analyzed as part of a project that uses Strict mode.) See *Strict C/C++ Mode Options* on [page 76](#).
- **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. See *Fuzzy C/C++ Mode Options* on [page 79](#).

For more details, see “Creating Accurate C/C++ Projects” in the [support knowledge base](#).

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



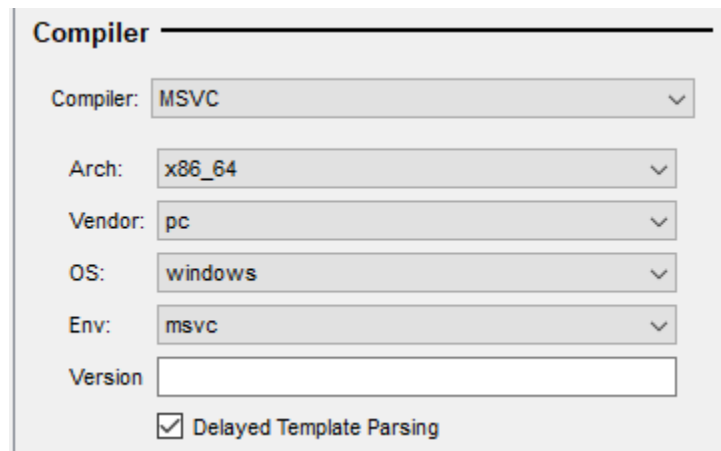
The rest of the options change depending on the mode you select.

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
- CUDA

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:



The screenshot shows a dialog box titled "Compiler". It contains the following fields:

- Compiler: MSVC (dropdown)
- Arch: x86_64 (dropdown)
- Vendor: pc (dropdown)
- OS: windows (dropdown)
- Env: msvc (dropdown)
- Version: (empty text field)
- ☒ Delayed Template Parsing

- **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. Some additional compilers are supported only in Fuzzy mode; see *Fuzzy C/C++ Mode Options on page 79*.
- **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 MacOSX, 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.
- **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:



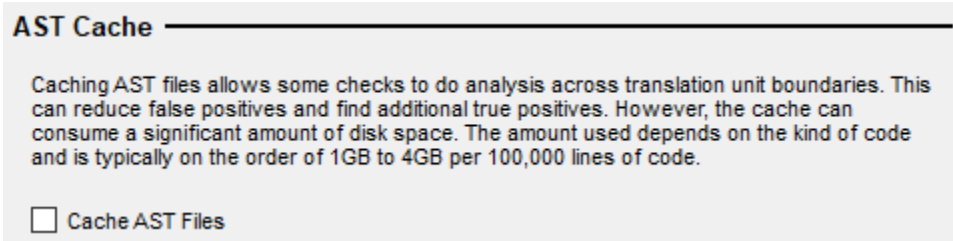
- **Memory Management:** If you use Objective-C, select the memory management mode used. The options are MMR (manual retain-release), ARC (automatic reference counting), and GC (garbage collection).

The **CUDA** option is as follows:



- **GPU Architecture:** Select the compute capability (also called the SM version) of the GPU architecture used for this project, if any. This corresponds to NVIDIA's nvcc sm flags. Supported compute capabilities include the range from sm_20 to sm_80.

The **AST Cache** option is as follows:



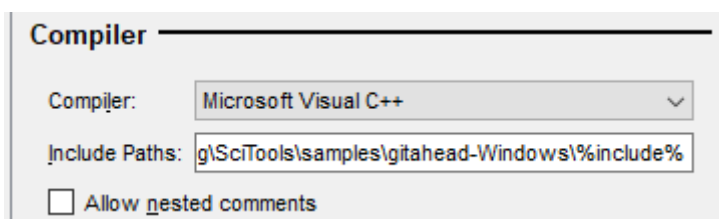
- **Cache AST Files:** Check this box if you want to perform project analysis across translation unit boundaries. Such analysis can improve the quality of the analysis. However, it requires a significant amount of additional disk space.

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:

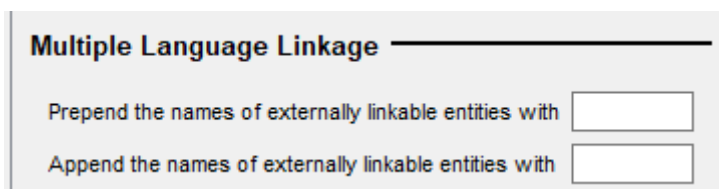
- Compiler
- Multiple Language Linkage
- Optimization

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:



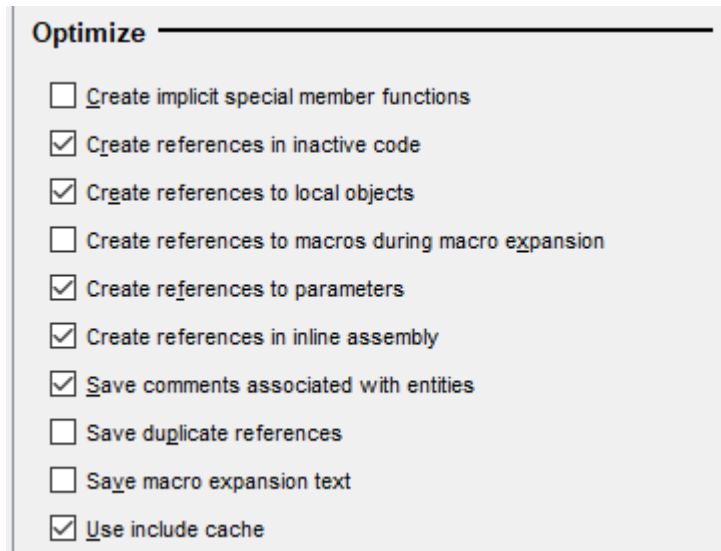
- **Compiler:** Select the compiler/platform that you use. Many different compilers are supported. Your choice affects how *Understand* analyzes the project. Note that not all features of a particular compiler will necessarily be handled.
- **Compiler Include Paths:** Type or paste the path the compiler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



The screenshot shows a window titled "Optimize" with a list of ten checkboxes. The first checkbox, "Create implicit special member functions", is unchecked. The next five checkboxes, "Create references in inactive code", "Create references to local objects", "Create references to macros during macro expansion", "Create references to parameters", and "Create references in inline assembly", are all checked. The next two checkboxes, "Save comments associated with entities" and "Save duplicate references", are unchecked. The next checkbox, "Save macro expansion text", is unchecked. The final checkbox, "Use include cache", is checked.

Option	Checked
Create implicit special member functions	<input type="checkbox"/>
Create references in inactive code	<input checked="" type="checkbox"/>
Create references to local objects	<input checked="" type="checkbox"/>
Create references to macros during macro expansion	<input type="checkbox"/>
Create references to parameters	<input checked="" type="checkbox"/>
Create references in inline assembly	<input checked="" type="checkbox"/>
Save comments associated with entities	<input type="checkbox"/>
Save duplicate references	<input type="checkbox"/>
Save macro expansion text	<input type="checkbox"/>
Use include cache	<input checked="" type="checkbox"/>

- **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. Caching speeds up analysis, but also uses more memory. If you have problems with excessive memory use during analysis, turn this option off. Note that there are also situations where turning the include cache on or off can affect analysis results, particularly where include actions are dependent on where they are included.

C++ > Includes Category

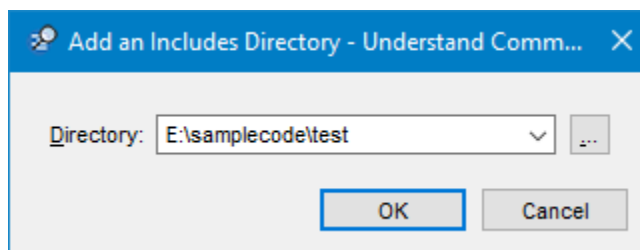
The **C++ > Includes** category in the Project Configuration dialog (which you open with **Project > Configure Project**) allows you to specify include directories. You can specify multiple directories to search for include files used in the project.

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

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.

Click **Search** to open the Missing Header Files dialog. See *Using the Missing Header Files Tool* on [page 105](#) for details.

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



Click **Edit** or **Remove** to modify or delete an include directory you have added.

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.

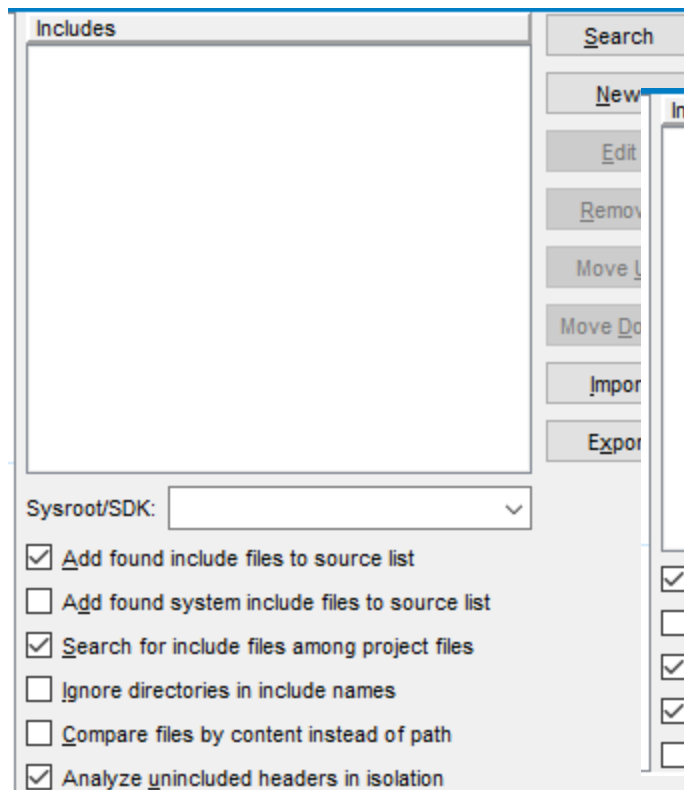
Typically only include files that are not directly related to your project (such as system-level includes) and that you do not want to analyze fully are defined here. For project-level includes that you want to be analyzed, add those include files as source files in the **Files** category.

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 66](#)).

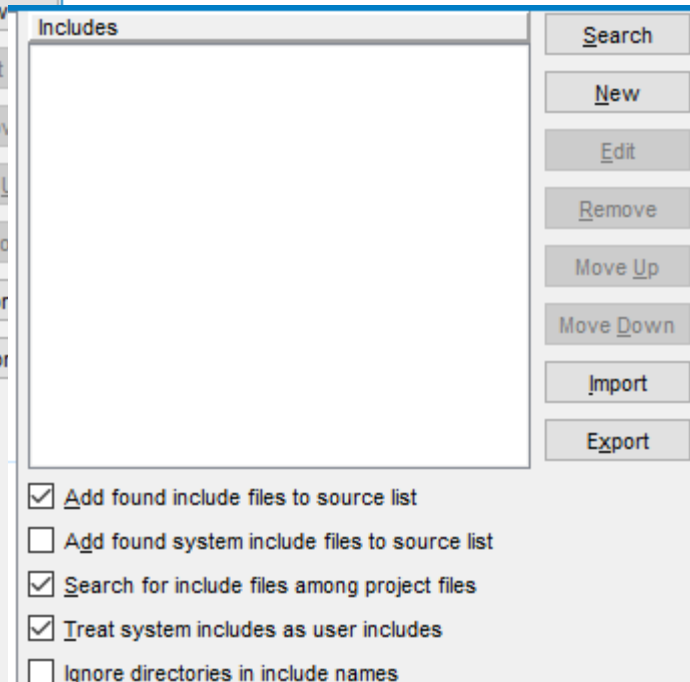
You can import or export a list of include directories from a text file by clicking **Import** or **Export** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative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

Includes Options (Strict mode)



Includes Options (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.

C++ > Includes > Automatic Includes Category

(Both Strict and Fuzzy mode) In the **C++ > Includes > Automatic Includes** category you can specify include files that should be included before each file in a project.

To add a file, click **New** and browse for the file(s). Then click **Open**. You can import or export a list of auto include files from a text file by clicking **Import** or **Export** and selecting the text file that contains one file path per line.

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

C++ > Includes > System Includes Category

(Strict mode only) In the **C++ > Includes > System Includes** category lets you specify system paths that the source code uses.

To search a directory and its subdirectories for a framework, click **Search**. This opens the Missing Header Files tool, which is described in *Using the Missing Header Files Tool* on [page 105](#).

To add a specific location, click **New** and browse for the folder. Then click **Select Folder** and then **OK**. You can import or export a list of framework folders from a text file by clicking **Import** or **Export** and selecting the text file that contains one path per line.

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

C++ > Includes > Frameworks Category

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

To search a directory and its subdirectories for a framework, click **Search**. Click **OK** to add the found directories to the list of frameworks.

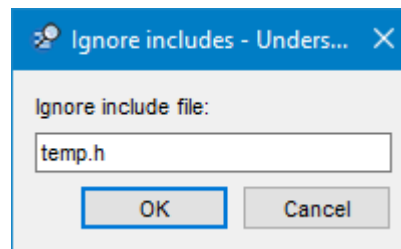
To add a location, click **New** and browse for the folder. Then click **Select Folder** and then **OK**. You can import or export a list of framework folders from a text file by clicking **Import** or **Export** and selecting the text file that contains one path per line.

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

C++ > Includes > Ignore Category

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

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



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

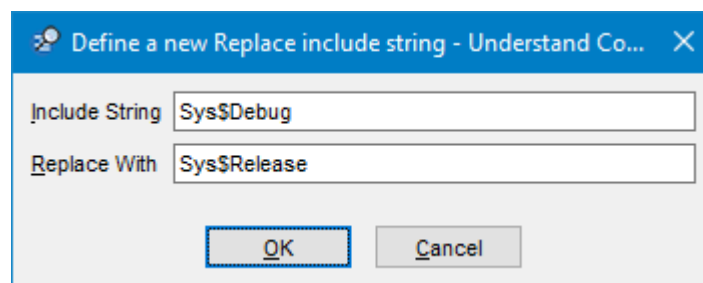
You can import or export a list of files to ignore from a text file by clicking **Import** or **Export** and selecting the text file that contains one filename per line.

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

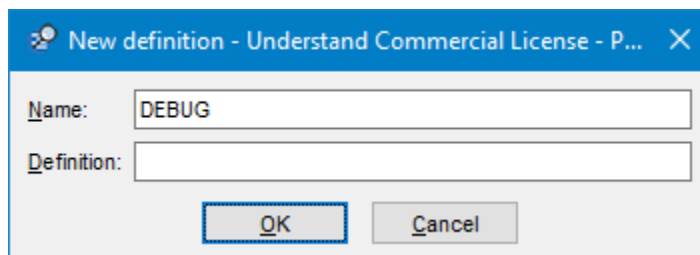
C source code is often sprinkled with pre-processor directives providing instructions and options to the C compiler. Directives such as the following affect what the software does and how it should be analyzed:

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

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

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. If any macros used in the source are undefined, you will see an **Undefined Macros** button. You can click this button and use the tool to define additional macros. See *Using the Undefined Macros Tool* on [page 106](#).

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
```

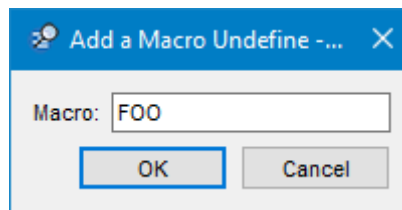
You can import or export a list of macros and their optional definitions from a text file by clicking **Import** or **Export** and selecting the file. The file must contain one macro definition per line. A # sign in the first column of a line in the file indicates a comment. The file should separate 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 und command line using -define
- 7 Define in source file (#define / #undefine in source)

C++ > Macros > Undefines Category

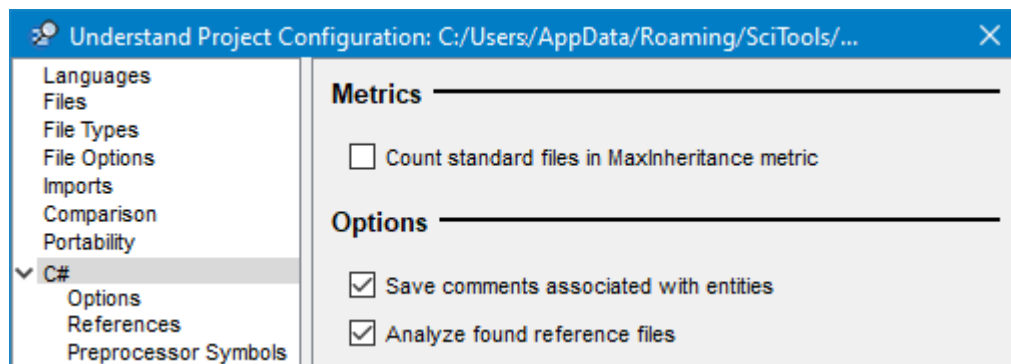
You can list undefined macros in the **C++ > Macros > Undefines** category in the Project Configuration dialog. Click **New** and type the name of a macro that you do not want to be defined. Then click **OK**.



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.

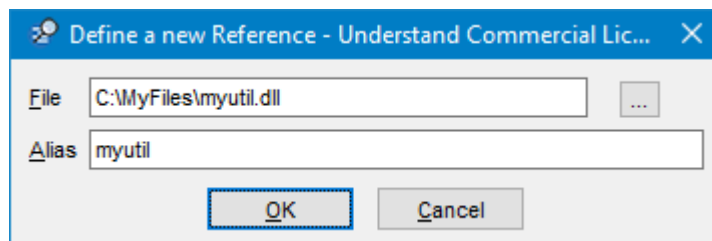


The fields in the **C# > Options** category are as follows:

- **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.
- **Framework:** Choose whether the code uses the .NET framework or no framework. If you select the .NET framework, the Implicit Framework References area lists the additional standard files that will be included in the project for analysis, including their full paths.

If your C# code is managed in a Visual Studio project, *Understand* supports importing project files from Visual Studio .NET 2002 through Visual Studio 2022.

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 screenshot shows the 'Understand Project Configuration' dialog box with the 'Fortran > Options' category selected in the left-hand tree. The main area is divided into three sections: 'Compiler', 'Multiple Language Linkage', and 'Options'.

Compiler Section:

- Version:** A dropdown menu set to 'Fortran95'.
- Format Options:**
 - Format:** A dropdown menu set to 'Auto'.
 - Truncate column:** A dropdown menu set to 'Auto'.
 - Column:** A text box containing the value '72'.
 - Free format file filters:** An empty text box.
- Checkboxes:**
 - ☐ Allow C-style comments
 - ☐ Allow :* comments
 - ☐ Allow colons in names
 - ☐ Allow function declaration without parentheses
 - ☐ Allow parameter declaration without parentheses
 - ☐ Allow quote in octal constants
 - ☐ Case sensitive identifiers
 - ☒ Use preprocessor
- Intrinsics File:** A text box containing 'ran\intrinsics95.txt', followed by a browse button ('...') and a 'Reset' button.

Multiple Language Linkage Section:

- The case of externally linkable entities is:**
 - ☒ all lowercase
 - ☐ all uppercase
 - ☐ preserved
- Prepend the names of externally linkable entities with:** An empty text box.
- Append the names of externally linkable entities with:** An empty text box.

Options Section:

- Display entity names as:** A dropdown menu set to 'Original'.
- ☒ Prompt on parse errors
- ☒ Save comments associated with entities

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. 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 or Auto format, you may choose what column terminates statements. Common columns 72 and 132 are available or you may specify a custom column or no truncation. If Auto format detection is selected, the setting for the Truncate column applies only to files that are identified as fixed format. Selecting Auto for the Truncate column causes *Understand* to determine the correct truncation column for fixed format files on a file-by-file basis.
- **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 (`intrinsic77.txt`, `intrinsic90.txt`, and `intrinsic95.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.
- **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.
- **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.
- **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.

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 105](#) 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 81](#).

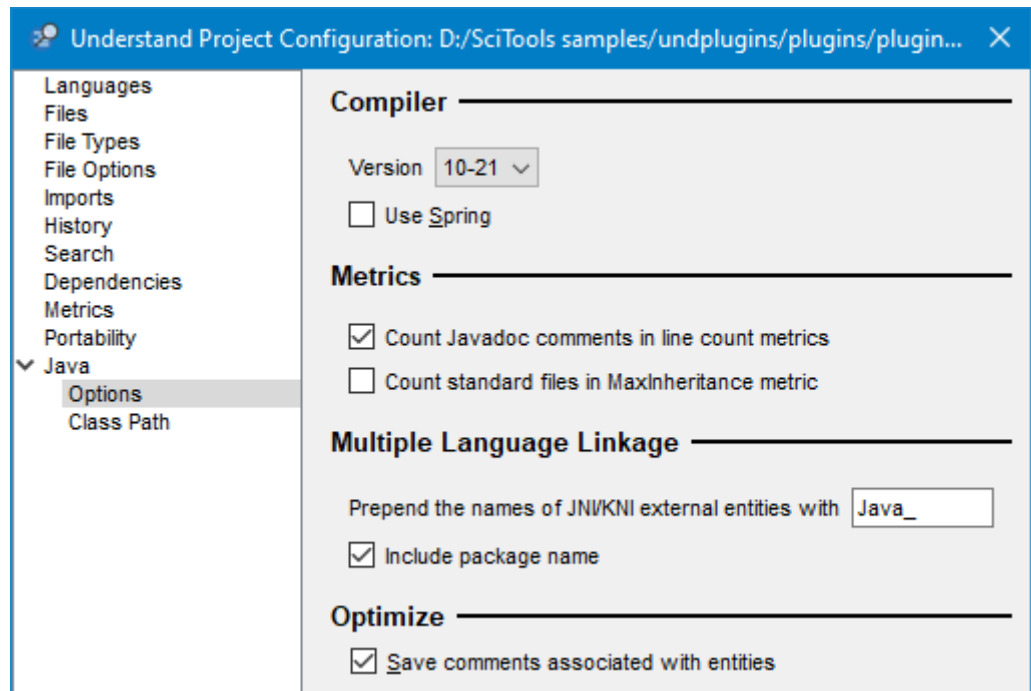
Other Fortran Categories

For information about the **Fortran > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on [page 84](#).

For information about the **Fortran > Macros** category, see *C++ > Macros Category* on [page 85](#). 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 10-21.
- **Use Spring:** Check this box if your application uses the Spring framework. Currently, *Understand* supports Spring annotations for Dependency Injection, Aspect Oriented Programming, and JPA/Hibernate. It does not support XML configuration. After analyzing the project, Framework entities and information about them are available in the Entity Filter and Information Browser. For more information, see the “Java Spring Support” topic on the [Support website](#).
- **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.

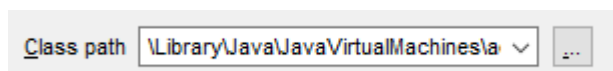
Both jar files and class files are supported.

- 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**.

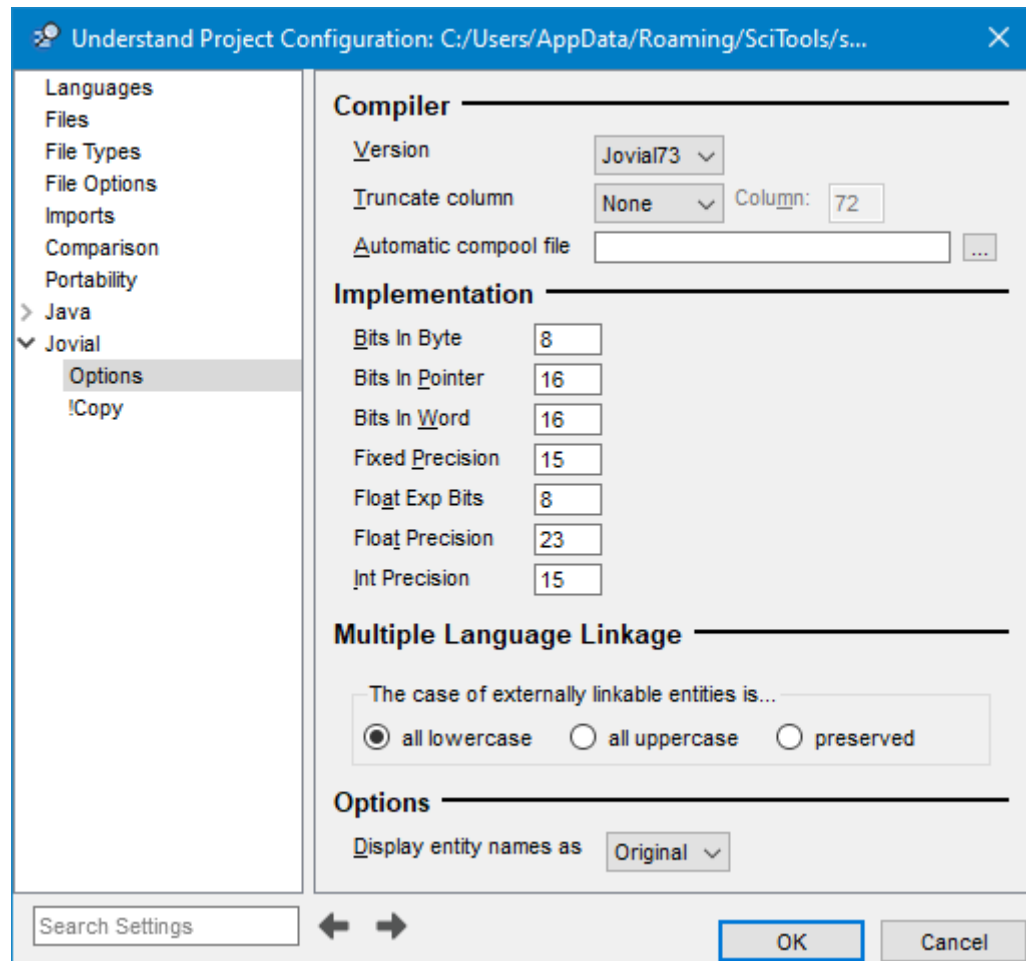
If a class is found in both a .java and .class file in the class path, the class in the .java file is used.

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

Select a jar or class file and click **Edit** to modify the path or **Remove** to delete it from the list. You can change the order of the list with the **Move Up** and **Move Down** buttons.

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.

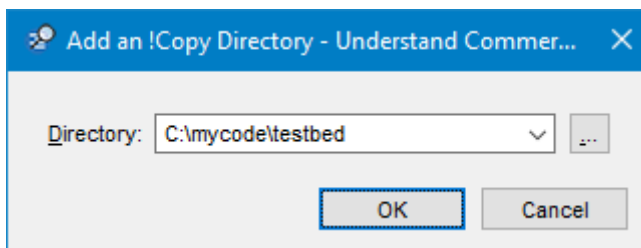
- **Automatic compool file:** Click ... and browse to the compool file you want to use. The file extension can be *.txt, *.cpl, or *.jov. The selected file is automatically imported into all other files in the project.
- **Implementation fields:** The fields in this section allow you to specify the sizes 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. !Copy files in Jovial are similar to include files in other languages.

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.)

To create additional entity references at !Copy commands, check the **Expand copy file references** box. For example, if a variable is set in a !Copy file and this box is checked, the project analysis creates “set” references to the variable both in the !Copy file and at the location of the !Copy command. If this box is unchecked, the analysis creates only the “set” reference in the !Copy file.

Pascal Options

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

Compiler

Version: Delphi

☒ Allow embedded SQL

Predeclared entities file: ca\predeclareddelphi.txt

dfm converter exe:

Multiple Language Linkage

The case of externally linkable entities is: Preserve

Prepend the names of externally linkable entities with:

Options

Display entity names as: Original

Namespaces:

☐ Parse library implementation code

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 97](#) 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 85](#).

Pascal > Standard Library Paths Category

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

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

```
Uses System;
```

The standard libraries are not used when computing project metrics.

To add a directory, follow these steps:

- 1 Click the **New** button.
- 2 Click the ... button and browse to a directory. Then click **OK**.
- 3 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.

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.

Version

☒ Python executable ...

Version: Python3

☐ Python2

☐ Python3

Standard Library

☒ Use built-in standard library files

Dynamic Resolve

☐ Assume nearest matches

Errors

☐ Ignore import errors in try blocks

Optimize

☒ Save comments associated with entities

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. If you specify an invalid location for the Python executable, *Understand* attempts to use the default Python executable to determine the Python version to use for analysis.
- **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.

The Python API always uses the version bundled with *Understand*. By default, this is the interpreter installed in `C:\Program Files\SciTools\bin\pc-win64\Python` on Windows.

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.

Note: Files with an extension of *.upy are treated as Python files. This is the file extension used for *Understand* plugins written in Python.

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 81](#).

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.

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 following web-related file types are recognized by default:

- HTML: .htm, .html
- Javascript: .js, .cjs, .mjs,
- TypeScript: .ts, .tsx, .cts, .mts (including d.*.ts files, such as file1.d.css.ts)
- PHP: .php,
- Perl: .pl, .pm
- XML: .xml

You can cause *Understand* to add other types of files to a project, including Perl files. For unsupported languages, syntax highlighting is provided, but entities in the code are not analyzed. See *Adding Directories on page 50* for details.

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

The screenshot shows a configuration window with two sections: 'Javascript Options' and 'PHP Options'. The 'Javascript Options' section includes checkboxes for 'Analyze jQuery' and 'Analyze Node.js', a 'Search Path' text box, a 'Predefined' dropdown menu showing 'redefined.txt', a button with three dots, and a 'Reset' button. Below these are checkboxes for 'Search Strings for Entity Names' and 'Add found imported files to source list' (which is checked). The 'PHP Options' section includes a 'Version' dropdown menu set to '5.3', and checkboxes for 'Allow Short Tags' (checked), 'Allow ASP Style Tags', and 'Save comments associated with entities' (checked).

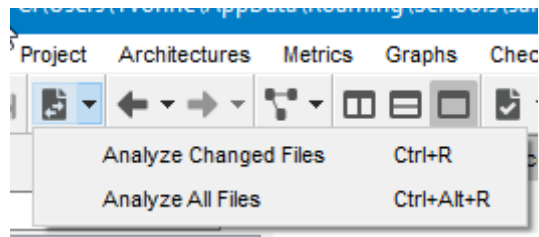
- **Analyze jQuery:** Check this box if you want the analysis to interpret a `$(...)` call as a jQuery call. A **jQuery Selector** entity is created for the string literal parameter passed to `"$"`. For example, `$(".foo")` would create a jQuery Selector entity named `".foo"`. Also, a **jQuery Selector 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 range from version 5.3 to 8.3.
- **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

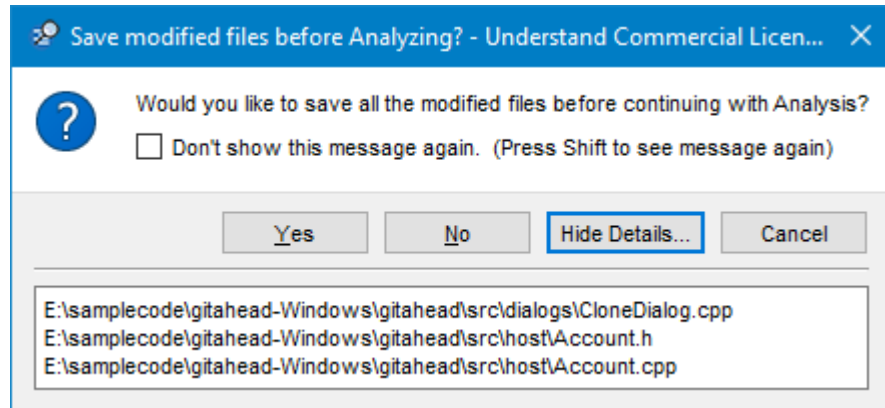
Once you configure a project, *Understand* can analyze the project. During analysis, the source files are examined, and data is stored in a database to allow quick browsing of large projects. When you modify a project's configuration, a prompt to analyze the project appears automatically. You can also analyze a 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)

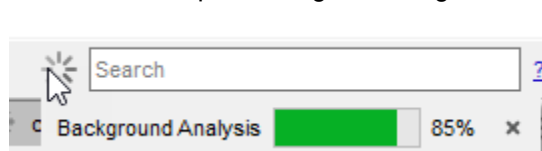
If some files have been modified but not saved, you are asked whether you want to save all modified files. Click **Show Details** to see a list of the files that will be saved.



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



When processing is running in the background, a busy icon is shown in the *Understand* toolbar. Click this icon to see what processing is running and its progress.



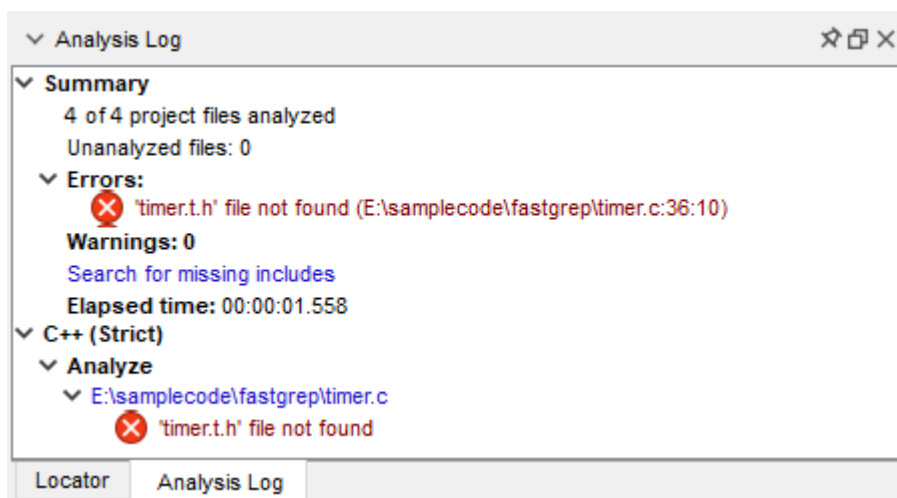
To pause or cancel the project analysis, click the **X** next to the progress bar. You will be asked if you want to Abort the analysis or Continue the analysis.

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 Home tab is open, statistics about the accuracy and errors found when parsing are added to the top of the Home tab. Click a statistic in the gray bar to open a view that can be used to help correct the problem.

25	67%	113	3	39,852	103	31	495	184
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 105](#)).

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 119](#) for options that affect the project analysis.

When you open an existing project, you may be asked to allow certain CodeCheck checks to run in the background each time you analyze the project. See *Running Configurations in the Background* on [page 294](#).

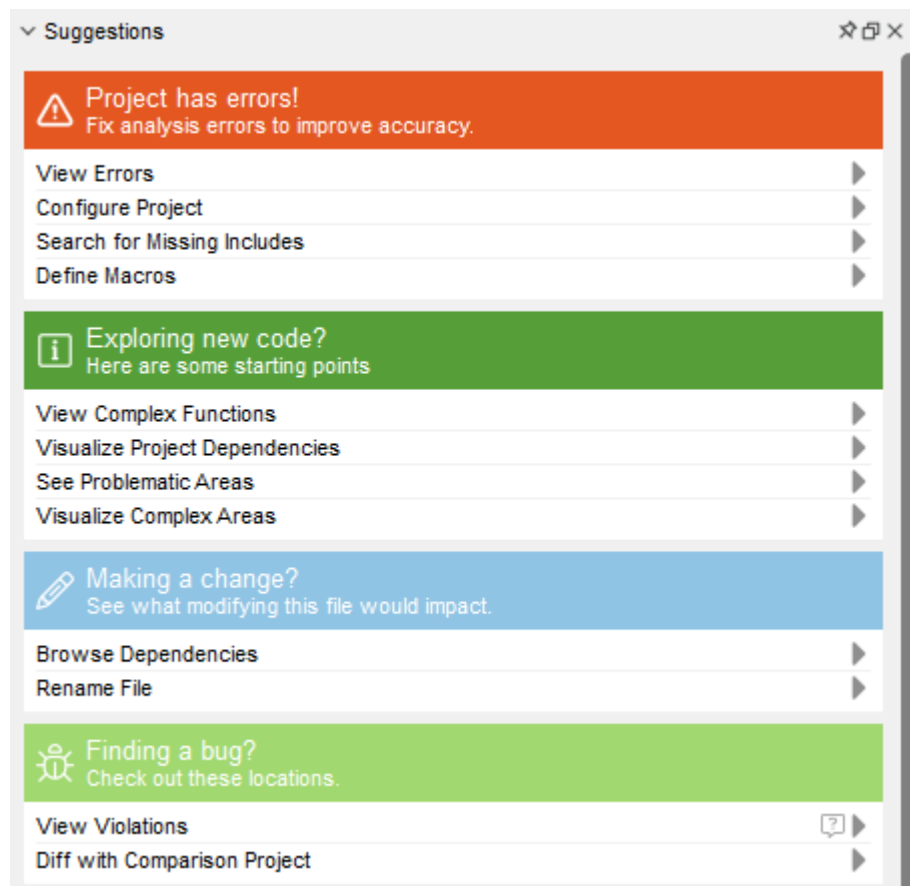
Tip: A configured project may be analyzed in batch mode using the command line program “und”. Refer to *Using the und Command Line* on [page 347](#) for details on using “und”.

Improving the Analysis

If your project analysis results in warnings about missing files, choose **Project > Improve Project Accuracy > Missing Includes** and see [page 105](#) 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 106](#) for how to use the Undefined Macros tool.

For links to tools that can help you find and correct errors and warnings in your project, choose **Help > Show Suggestions**. These tools include the Violation Browser, Home tab graphs, Dependency graph, and Configuration dialog.

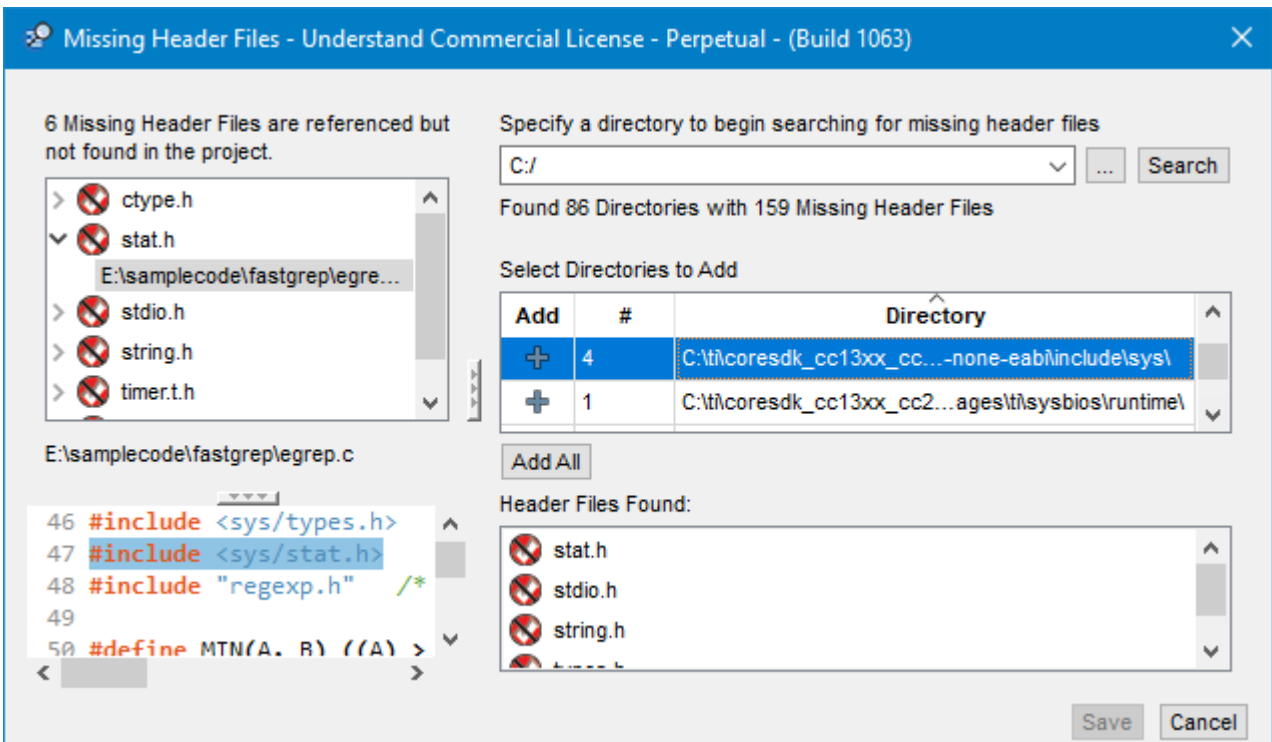


For other types of warnings, revisit the categories of the *Project Configuration Dialog* on [page 46](#) to make sure your project is configured correctly. Multiple similar errors can often be fixed quickly. For advice about tuning configuration settings to improve your project analysis, choose **Project > 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.

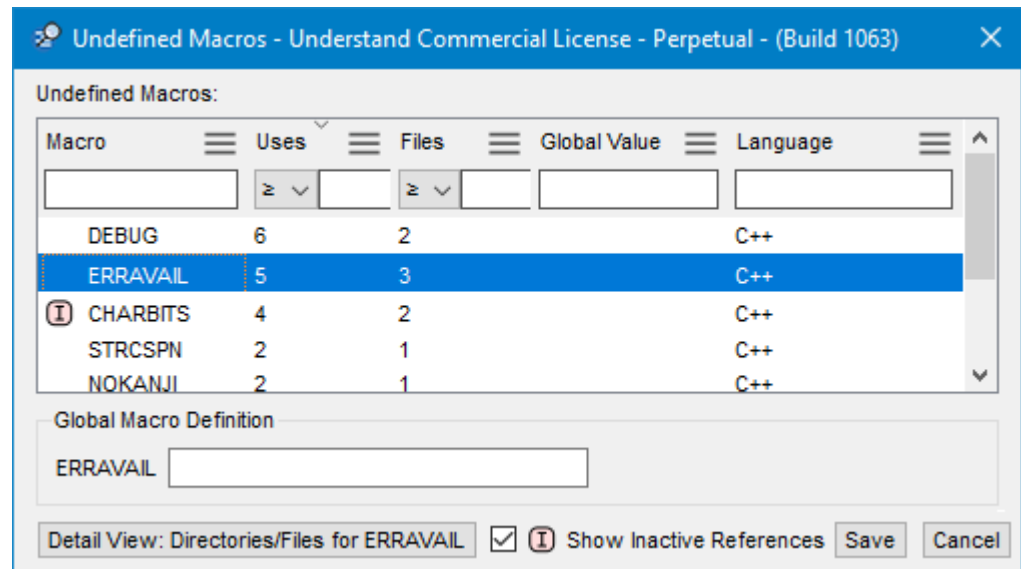


- 3 In the **Specify a directory to begin searching** field, click the ... button and browse to find a directory that may contain one or more missing header files. All subdirectories of the directory you select will be searched. The search is case-insensitive on Windows.
- 4 Click **Select Folder** and 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 and which files have been found with a green . 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 170](#) 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.

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:

Section	Page
Understand Options Dialog	108
General Category	109
User Interface Category	111
Key Bindings Category	116
CodeCheck Category	118
Analyze Category	119
Dependency Category	120
Editor Category	121
Graphs Category	130
Annotations Category	132
User Tools Category	133

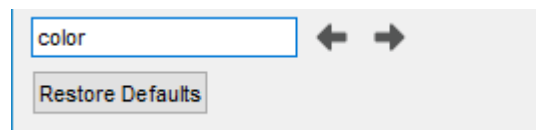
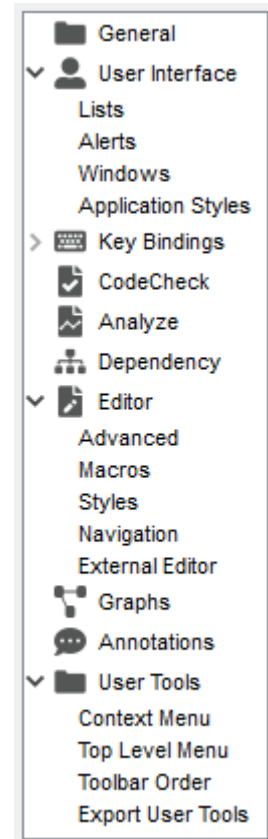
Understand Options Dialog

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 109*
- *User Interface Category on page 111*
- *Key Bindings Category on page 116*
- *CodeCheck Category on page 118*
- *Analyze Category on page 119*
- *Dependency Category on page 120*
- *Editor Category on page 121*
- *Graphs Category on page 130*
- *Annotations Category on page 132*
- *User Tools Category on page 133*

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:

The screenshot shows the 'General' category of the 'Tools > Options' dialog. It contains the following sections and options:

- Application Font:** Font: Arial, Size: 8, with a 'Change Font ...' button.
- Show on Startup:** A checked checkbox for 'Show Welcome Page on startup'.
- Auto Loading/Saving Options:**
 - Unchecked checkbox: 'Save all modified editor windows when application loses focus.'
 - Unchecked checkbox: 'Open last project on startup (currently: E:/samplecode/zlib/zlib.und).'
 - Unchecked checkbox: 'Use Default Working Directory'.
- Performance:**
 - Unchecked checkbox: 'Enable permissions checking for NTFS filesystems'.
 - Checked checkbox: 'Allow interactivity during intensive processing'.
 - A text field 'Allow events processing every' with the value '100' and a spinner, followed by 'milliseconds'.
- Settings Locations:** A text field 'Settings Folder:' with the value 'C:/Users/Yvonne/AppData/Roaming' and a 'Browse...' button.

- **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 22](#)) 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. Additional parse data (the `parse.udb` file) and other local data are also stored in the location specified here.

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

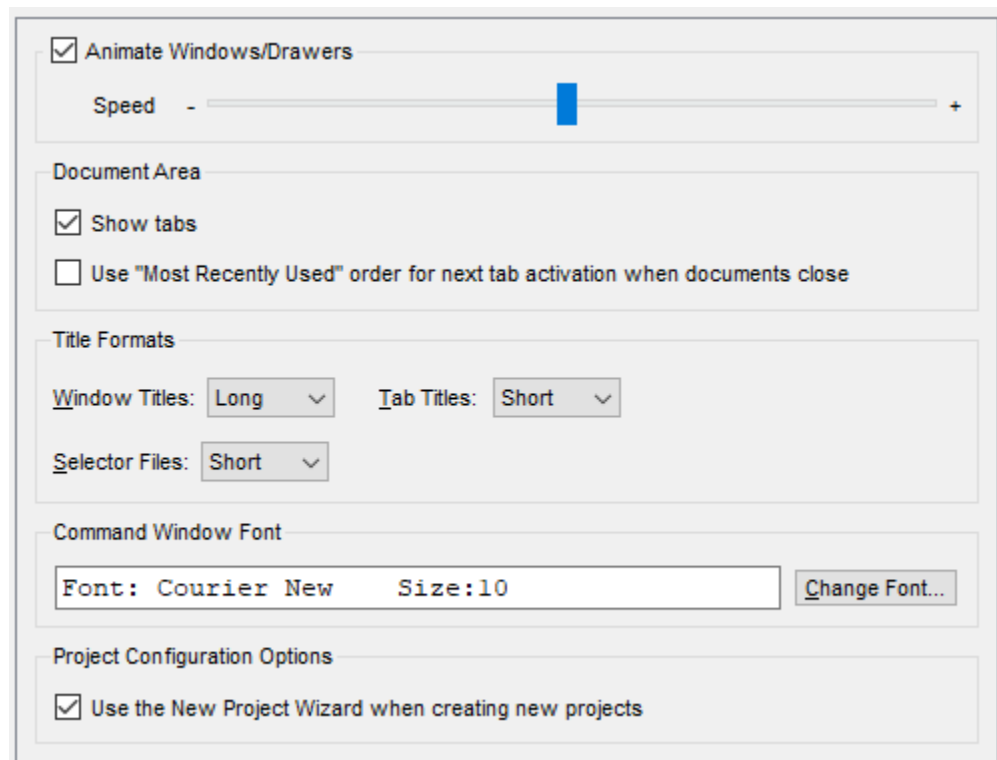
Some settings in the Understand Options dialog take effect only after you restart *Understand*. A red message in the dialog informs you of this issue if you make one of these changes.

User Interface Category

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

- *User Interface > Lists Category on [page 112](#)*
- *User Interface > Alerts Category on [page 113](#)*
- *User Interface > Windows Category on [page 114](#)*
- *User Interface > Application Styles Category on [page 115](#)*

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

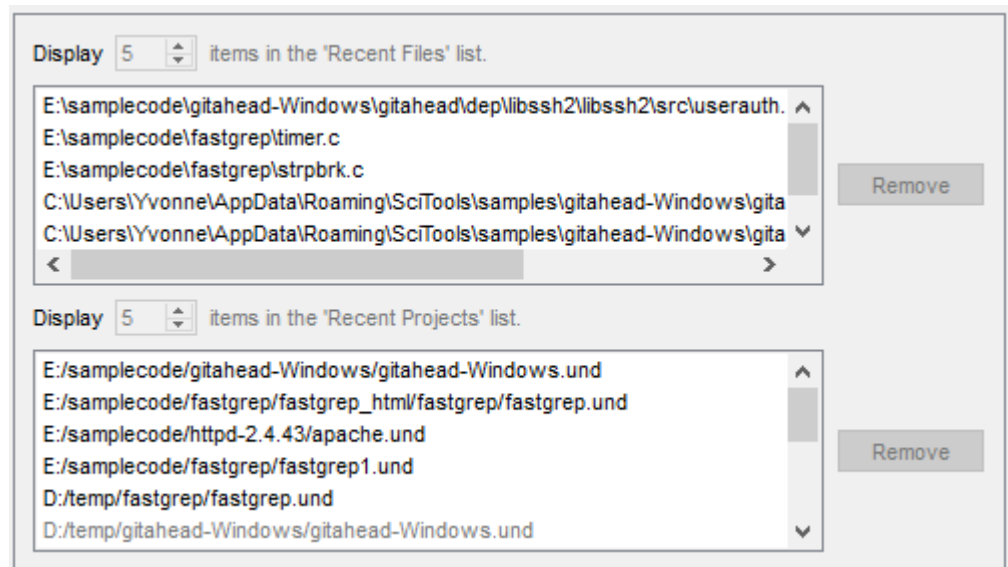


- **Animate Windows/Drawers:** If checked (on by default), opening and closing windows and tabbed areas (drawers) is animated. You can choose a faster or slower **Speed** than the default.
- **Show tabs:** If checked (on by default), tabs are shown at the top of the document area for each of the windows 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 37) 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:

The screenshot shows the 'User Interface > Alerts' category in the 'Tools > Options' dialog. It contains five sections with the following options:

- Save On Parse:**
 - ☒ Always Prompt
 - ☐ Save modified files before parsing
 - ☐ Don't save modified files before parsing
- Save On Command:**
 - ☒ Always Prompt
 - ☐ Save modified files before running a command
 - ☐ Don't save modified files before running a command
- Project Close:**
 - ☒ Prompt before closing the current project
- CodeCheck:**
 - ☒ Prompt when Violation count exceeds 300,000
- Audible Alerts:**
 - ☐ Sound beep on analysis completion
 - ☐ Sound beep when entity locator search does not match
 - ☐ Sound beep on all other actions

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 on analysis completion:** Turn this on to have a beep notify you when a project analysis is complete. By default, all types of audible alerts are off.
- **Sound beep when entity filter entry does not match:** Turn this on to have the computer beep if you type a filter in the Entity Filter that does not match any entity of the selected type.
- **Sound beep on all other actions:** Turn this on to have a beep notify you if another action occurs, such as completion of a Find command.

User Interface > Windows Category

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

The screenshot shows the 'User Interface > Windows' category in the 'Tools > Options' dialog. It contains several sections with settings:

- Editor Windows:** 'Open as:' with radio buttons for 'Captured' (selected) and 'Released'.
- Find in Files Search Window:** 'Display' with a spin box set to '5' and the text 'items in the Find in Files 'Find' list.'; 'Directory & Find List:' with radio buttons for 'Global' (selected) and 'Project'.
- Graph Windows:** 'Open as:' with radio buttons for 'Captured' (selected) and 'Released'.
- Information Browser Windows:** A checked checkbox for 'Show Code Snippet ToolTip in References'.
- Released Windows:** Two buttons, 'Save Released Window Positions' and 'Clear Saved Positions', and an unchecked checkbox for 'Keep Released Windows Forward'.

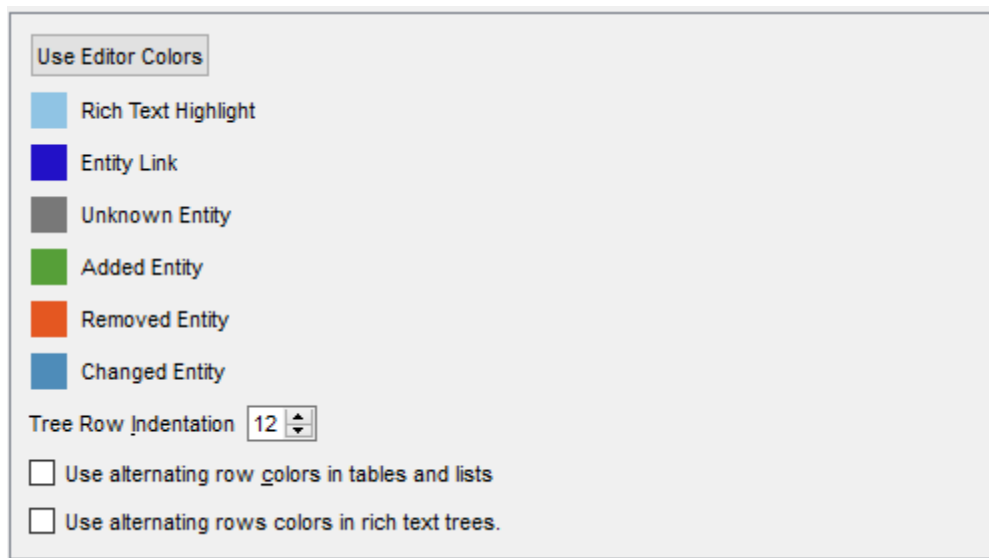
You can choose various options for different types of windows.

- **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:

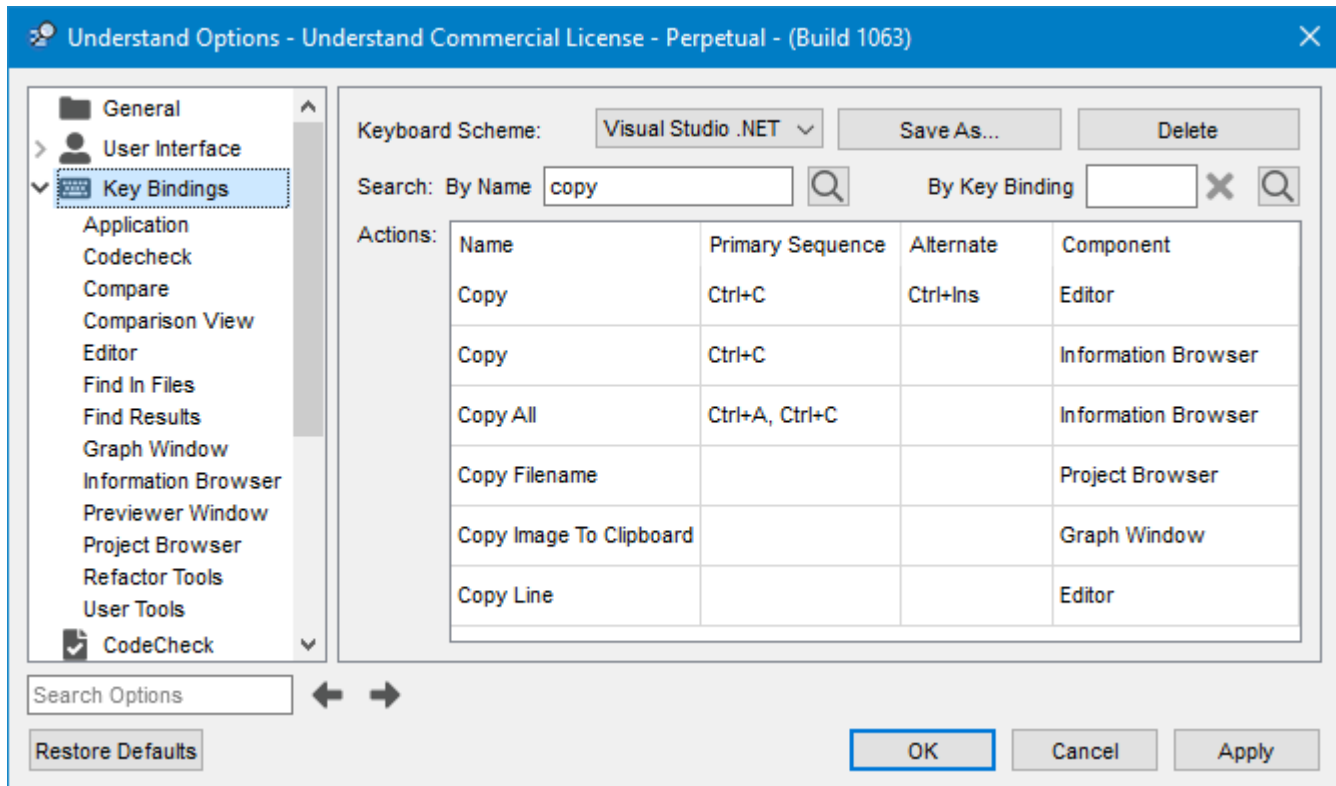


You can choose various options for different types of windows.

- **Use Editor Colors:** Click this button to apply the source editor colors from the Editor > Styles category ([page 127](#)) 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*:

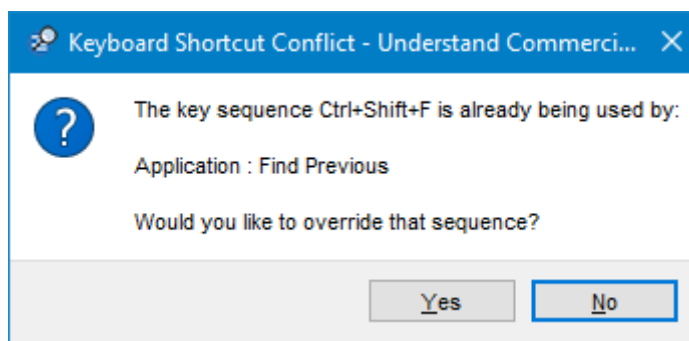


- **Keyboard Scheme:** This field allows you to choose groups of keyboard settings that are similar to other applications. The default settings are those native to *Understand*. Other choices are Visual Studio .NET key bindings and the Emacs editor key bindings. If you choose a scheme and click **OK**, 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 289](#) for more information.

The screenshot shows the 'CodeCheck' category in the 'Tools > Options' dialog. It is divided into two main sections: 'Item Views' and 'Priority Label Overrides'.

Item Views

- Foreground color of selection (white square)
- Background color of selection (dark blue square)
- Foreground color of Manual Ignores (white square)
- Background color of Manual Ignores (blue square)
- Foreground color of Inline Comment Ignores (white square)
- Background color of Inline Comment Ignores (orange square)
- Foreground color for Baseline Ignores (white square)
- Background color for Baseline Ignores (green square)

Priority Label Overrides

Original Label	Display Name
Urgent	<input type="text" value="Urgent"/>
High	<input type="text" value="High"/>
Medium	<input type="text" value="Medium"/>
Low	<input type="text" value="Low"/>
Informational	<input type="text" value="Informational"/>

In the Item Views area, click a color square next to an item in the list. Use the Select Color dialog to choose a new color for that type of item.

In the Priority Label Overrides area, you can change the text labels used for the various CodeCheck severity levels. See *Configuring Checks* on [page 291](#) for information about CodeCheck severity levels.

Analyze Category

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

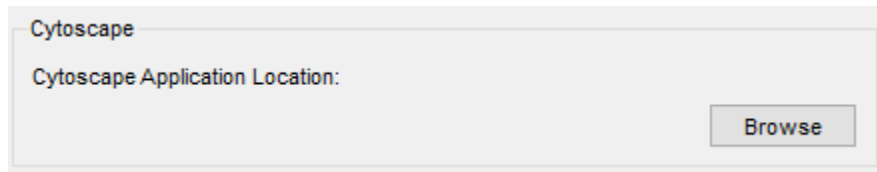
Project Analysis options

- ☐ Show log during analysis
- ☐ Show standard library files in log
- ☐ Automatically analyze changed files on save
- ☐ Automatically analyze changed files periodically

- **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.
- **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.
- **Automatically analyze changed files on save:** Check this option to cause any files that have been changed since the project was last analyzed to be analyzed when you save a source code file.
- **Automatically analyze changed files periodically:** Check this option to look for changed files every 10 seconds and begin a project analysis if needed.

Dependency Category

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



- **Cytoscape Application Location:** You can browse for the location where you installed Cytoscape (www.cytoscape.org), a free open-source program for analysis and visualization. Specifying this location allows *Understand* to open Cytoscape for viewing the dependency XML files exported as described in *Exporting Dependencies to Cytoscape* on [page 154](#).

For project-dependent options related to dependencies, see *Dependencies Options* on [page 62](#).

Editor Category

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

- *Editor > Advanced Category on [page 123](#)*
- *Editor > Macros Category on [page 127](#)*
- *Editor > Styles Category on [page 127](#)*
- *Editor > Navigation Category on [page 128](#)*
- *Editor > External Editor Category on [page 129](#)*

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

The screenshot shows the 'Editor' category options dialog. It contains the following settings:

- Default Style:**
 - Font: Consolas
 - Size: 10
 - Antialias: ☒
- File Mode:**
 - Encoding: System
 - Line Endings: CRLF (Windows)
 - On Save:
 - Convert existing line endings: ☐
 - Convert tabs to spaces: ☐
 - Add newline at end of file if absent: ☒
 - Remove trailing whitespace: ☐
- Page Guide:**
 - Show Page Guide: ☐
 - Column: 80
- Caret Line:**
 - Highlight Caret Line: ☒
- Indent:**
 - Show Indent Guide: ☐
 - Insert Spaces Instead of Tabs: ☒
 - Indent Width: 2
 - Tab Width: 8
- Whitespace:**
 - Invisible: ☒
 - Always Visible: ☐
 - Visible After Indent: ☐
 - Show End-of-Line: ☐
- Margins:**
 - Line Number: ☒
 - Blame: ☒
 - Fold: ☒
 - Status: ☒
 - Bookmark: ☒

- **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.

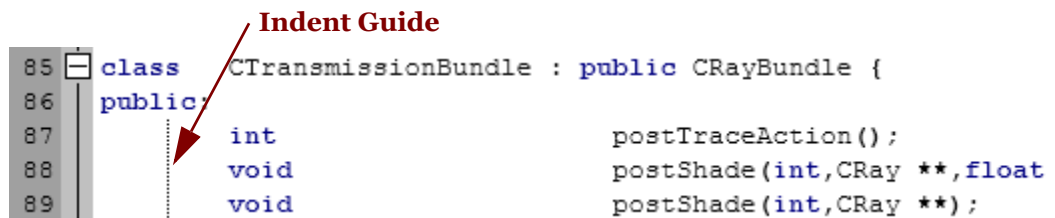
- **File Mode:** Select the type of **Encoding** to use when saving source files and the **Line Endings** character you want used. Many encoding formats are supported. The “System” encoding option uses the same encoding format defined for your operating system. You should change these settings only if 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 are deleted automatically when a file is saved.

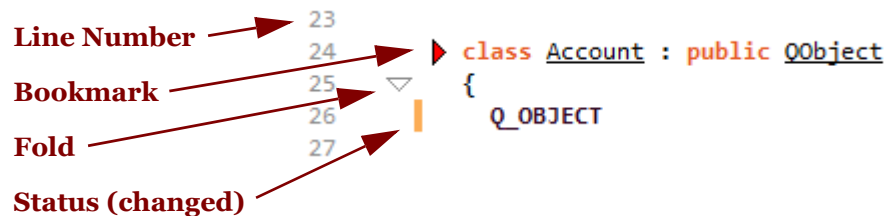
- **Page Guide:** Check the **Show Page Guide** box to display a line similar to the **Indent Guide** at a defined line width (that is, at the right edge of the code). Set the **Column** to the character width you want to see indicated.
- **Caret Line:** Check the **Highlight Caret Line** box if you 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 197](#)). Also, see *Editor > Advanced Category* on [page 123](#) 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 326* for details.
 - **Fold:** (on by default) turns on the ability to “fold” source code entity blocks out of the way.



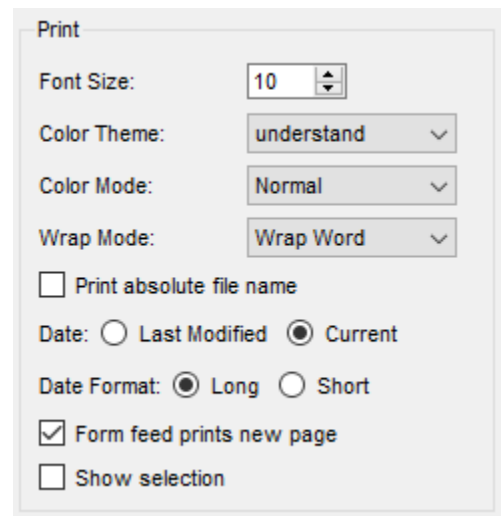
- **Status:** (on by default) shows change bars if the line has been modified but not saved.
- **Bookmark:** (on by default) shows bookmarks (red arrows) next to line numbers.

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.

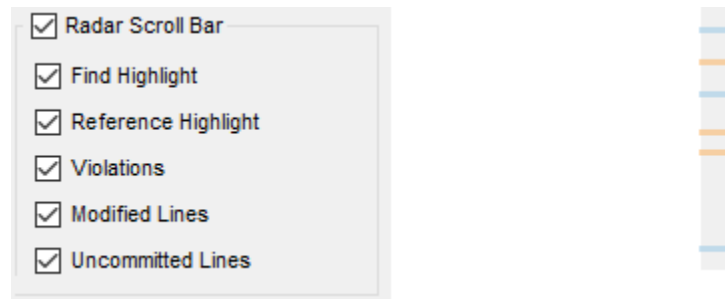
Print: The following options control how source code looks when you print it from an editor window:

- **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 196](#).
- **Color Theme:** Choose the color theme to be used when printing source code. The default is the “understand” theme. This can be different from the display theme, which is chosen using the *Editor > Styles Category on page 127*.
- **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. This is the default for printing.
- **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 198](#) 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.
- **Show selection:** Check this box if you want the text you have highlighted in the Source Editor to be highlighted in printouts.

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. The colors of these highlights can be changed using the *Editor > Styles Category* on [page 127](#). You can disable all radar highlighting by unchecking the **Radar Scroll Bar** box.



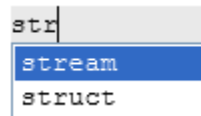
- The **Find Highlight** box controls highlighting for strings found with Ctrl+F (blue by default). Up to 1024 results in a file can be highlighted in the scroll bar.
- The **Reference Highlight** box controls highlighting for entities found by reference, for example, using the Information Browser (blue by default).
- The **Violations** box controls highlighting for CodeCheck violations. See [page 302](#).
- 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 326](#) for details.).

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 retain whitespace. If you disable this option, whitespace is retained using ` ` (non-breaking space) and `
` tags. If you use an application that does not respect the `<pre>` tag, disable this option to force formatting to match.

The image shows two settings panels. The top panel, titled 'Copy-and-paste', contains two checkboxes: 'Include line numbers in rich text' (unchecked) and 'Use preformatted white space' (checked). The bottom panel, titled 'Auto-complete', contains three checkboxes: 'Enable Auto-complete' (checked), 'Automatically suggest matches' (unchecked), and 'Ignore case' (checked).

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 of indenting in leading whitespace and inserts tabs or spaces everywhere else.

The image shows the 'Auto-indent' settings panel. It includes a section for 'Auto-indent' with three checkboxes: 'Enable auto-indent' (checked), 'Indent after newline' (checked), and 'Indent Braces' (unchecked). Below these are two fields: 'Tab auto-indents' set to 'Never' and 'Trigger characters' set to ':{ }'. A 'Vertical Caret Policy' section at the bottom has four checkboxes: 'Even' (checked), 'Jumps' (checked), 'Strict' (unchecked), and 'Slop' (unchecked). Below this is a 'Slop Value' field set to '0'.

- **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_SETYCAPOLICY.

- **Even:** Checking this box causes source code to scroll the same 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 326* for details.

By default, comments and text strings in code are checked for spelling errors. To disable spell checking of either comments or text strings, uncheck the **Enable Spellchecking in Comments** and/or **Enable Spellchecking in Strings** box. See *Spell Checking on page 202* for more about checking your spelling.

The screenshot shows a settings dialog with the following sections:

- Unused Entities:** Contains a checkbox labeled "Highlight Unused Entities" which is currently unchecked.
- Annotation Wrap:** Contains a checked checkbox labeled "Wrap Annotations" and a "Column:" field with a numeric input set to 79.
- Version Control:** Contains a checkbox labeled "Show Inline Blame" which is currently unchecked.
- Spellcheck:** Contains two checkboxes: "Enable Spellchecking in Comments" (checked) and "Enable Spellchecking in Strings" (unchecked).

Editor > Macros Category

You can record, save, and replay Source Editor macros as described [page 198](#). 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.

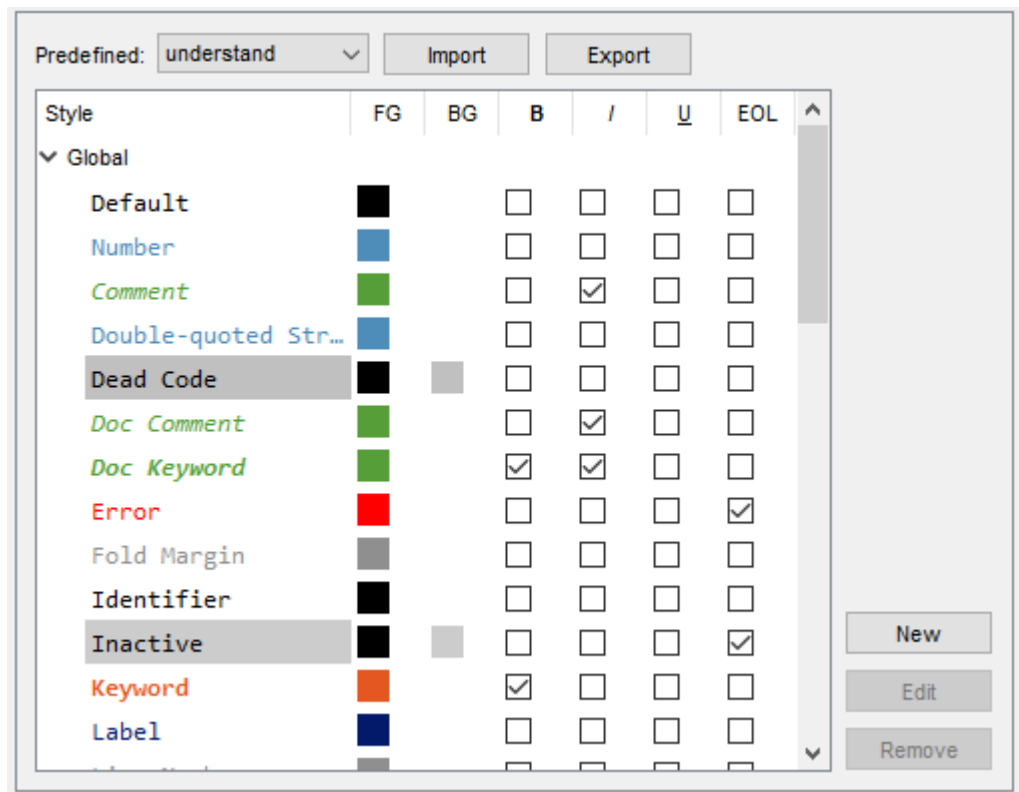
Editor > Styles Category

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

By default, *Understand* uses the “understand” theme and switches between dark mode and light mode based on your system setting.

To choose a different color theme, choose from the **Predefined** list. Other options include understand-dark, aqua, onyx, and terminal. To change the theme used when printing source code, see *Editor > Advanced Category* on [page 123](#).

To change an individual color within a theme, 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:

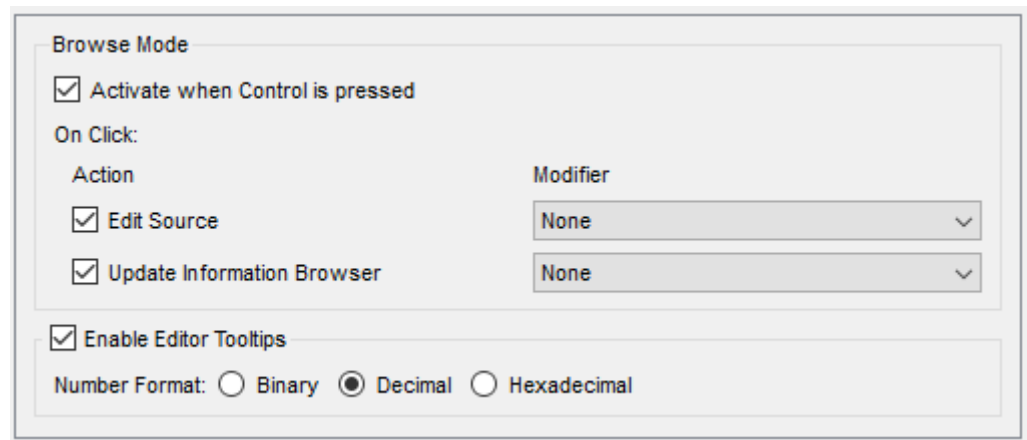
- **Dark orange text:** Used for language and preprocessor keywords
- **Blue text:** Used for characters and character strings
- **Italic green text:** Used for comments
- **Black text:** Used for all other source text
- **White background:** Used for most source text
- **Gray background:** Used for 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 182](#)). 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.

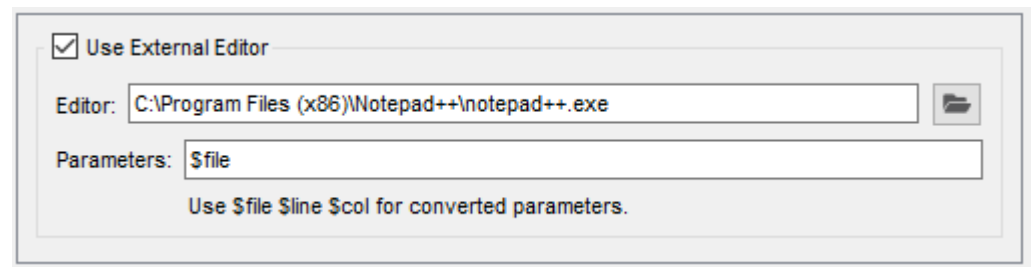
Editor > External Editor Category

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
```

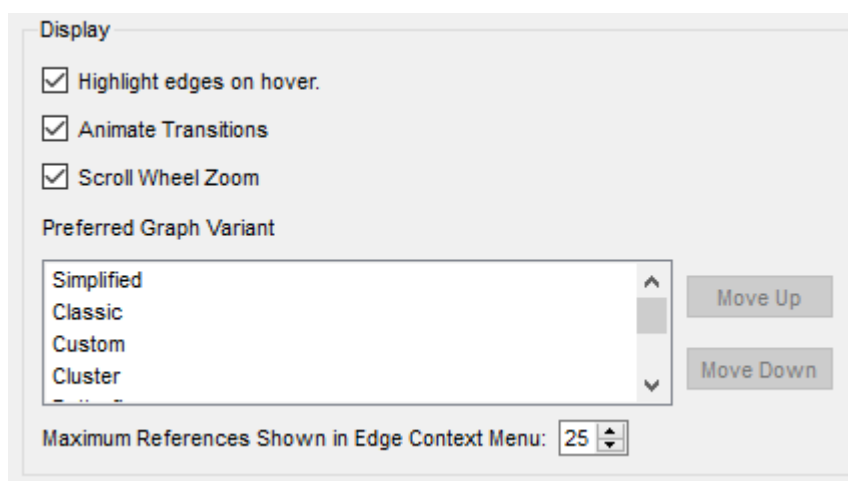
For the TextPad editor on Windows, the **Editor** is most likely `c:\Program Files\textpad4\textpad.exe`, and the **Parameters** should be as follows:

```
$file($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

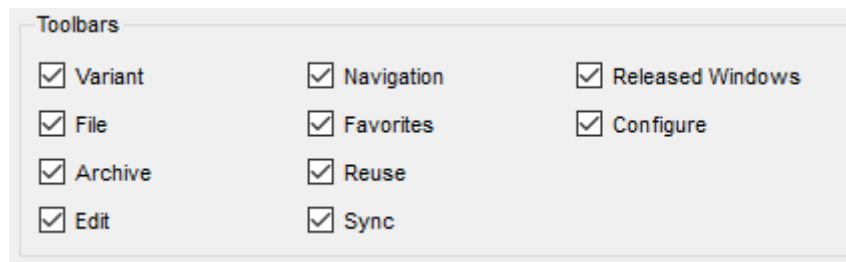
The Graphs category of the **Tools > Options** dialog lets you control options related to how graphs are displayed. Some options apply only to certain types of graphs, such as the cluster graphs.



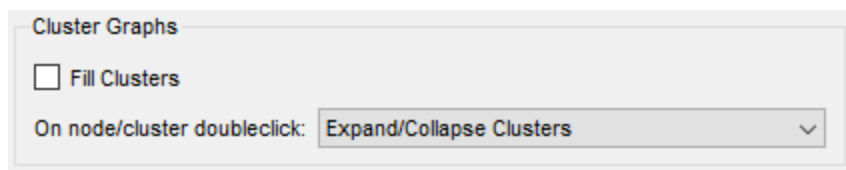
- **Highlight edges on hover:** By default, relationships (connecting lines) within a graph are highlighted when your mouse cursor hovers over the relationship. This makes it easier to distinguish between overlapping relationships. Text describing the relationship is always shown when your mouse cursor hovers over a relationship; this option does not affect display of the relationship description. See [page 220](#).
- **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 261](#) 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. By default, the Archive, Favorites, Reuse, and Released Windows icons are hidden. See [page 263](#) for the toolbar icons controlled by these check boxes.



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

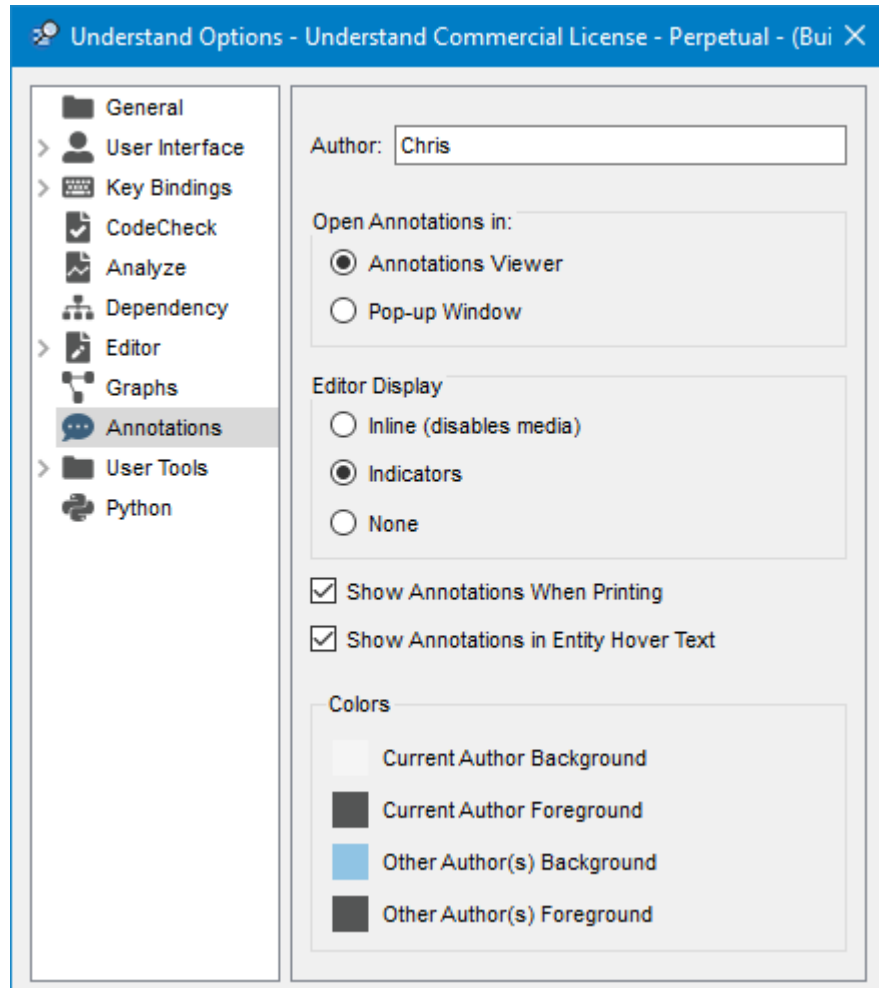



- **Fill Clusters:** By default, the background of a cluster is transparent. Check this box to add a colored background for clusters.
- **On node/cluster double-click:** Controls what happens when you double-click on a node in a graph. By default, clusters are expanded or contracted. You can change this 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.

Note: The settings previously provided in the Cluster Graph Styles section of this category can now be accessed by right-clicking on the graph legend in a graph. See *Controlling Graph Styles* on [page 271](#).

Annotations Category

The **Annotations** category of the **Tools > Options** dialog lets you control how annotations are displayed and edited. See [page 203](#) 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  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 331](#) for information about how to configure tools and provide commands to run these tools in the right-click menus, Tools menu, and toolbar.

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 159](#).

Details on the use and operation of the Source Editor is contained in the chapter *Editing Your Source* on [page 178](#).

This chapter contains the following sections:

Section	Page
PLEASE RIGHT-CLICK	135
Various Windows Explained...	136
Entity Filter	137
Information Browser	139
Project Browser	145
Exploring a Hierarchy	147
Dependency Browser	148
Favorites	155

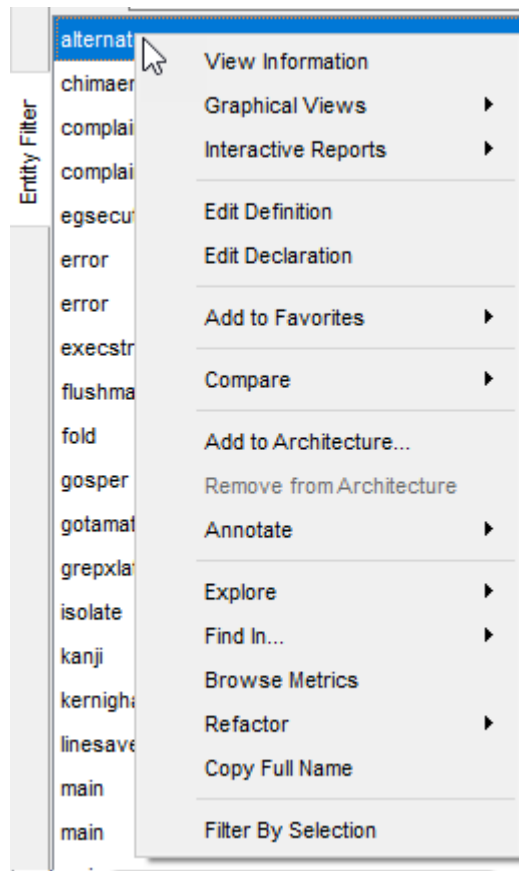
PLEASE RIGHT-CLICK

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.

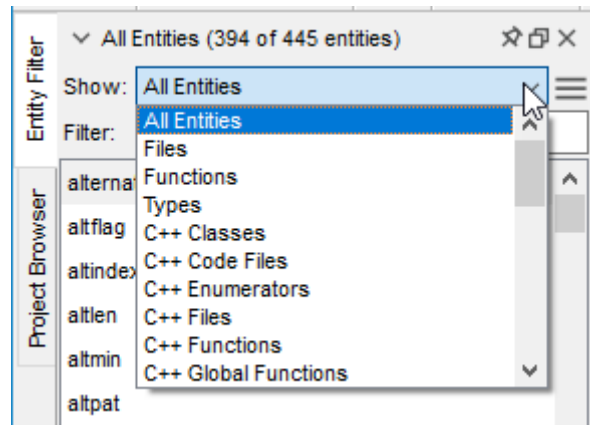
The tools available for finding and exploring entities are:

- **Analysis Log:** Shows results of project analysis, including any errors and warnings. See [page 102](#).
- **Architecture Browser:** Defines named regions and views of the project. See Chapter 8.
- **Bookmarks:** Show and jump to bookmarks. See [page 199](#).
- **Contextual Information Sidebar:** Show context information about the current source editor file. See [page 177](#).
- **Dependency Browser:** Lets you browse dependency relationships. See [page 148](#).
- **Entity Filter:** Provides an alphabetic list of entities of the selected type. See [page 137](#).
- **Entity Locator:** Lets you filter all entities in a project in complex ways. See [page 168](#).
- **Exploring View:** Lets you browse a relationship hierarchy. See [page 147](#).
- **Favorites:** Lets you provide quick links to frequently-used entities. See [page 155](#).
- **Find in Files:** Searches multiple files. See [page 163](#).
- **Information Browser:** Provides an explorer for entity characteristics and connections. See [page 139](#).
- **Previewer:** Provides a quick way to view code in successive locations. See [page 195](#).
- **Project Browser:** Lets you browse a hierarchical file list. See [page 145](#).
- **Source Editor:** Shows source code. See [page 178](#).
- **Scope list:** Lists the functions or similar constructs in a file. See [page 180](#).
- **Graphical Views:** Shows connections and structures of entities. See Chapter 11.
- **Metrics Browser:** Generate statistics about entities and architectures. See [page 248](#).
- **Window Selector:** Jump to a window that is open. See [page 174](#).
- **Annotations View:** Viewing annotations in the project. See [page 203](#).

Entity Filter

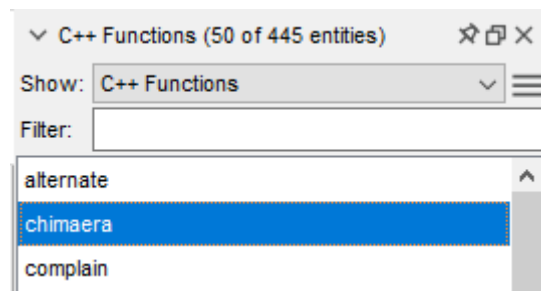
The *Entity Filter* provides a quick list of the selected entity type. You can filter this list to match a text string.


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 copy the names of one or more selected entities in the Entity Filter by pressing Ctrl+c or right-clicking and selecting **Copy Fullname**.


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 170](#) for examples.
- **Regular Expression:** With this option selected, you can use Unix-style regular expressions. See [page 170](#) 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.

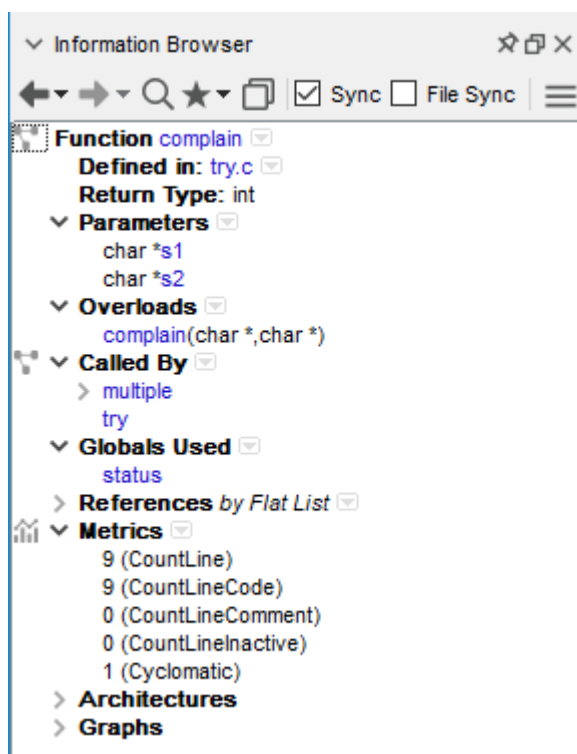
- **Root Calls:** Lists only entities that call others, but are not called themselves. These are either high-level code (mains), code called by hardware (interrupt handlers), or dead (unused) code.
- **Root CallBys:** Lists only entities that are called by other entities, but does not list entities that 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 Decls:** 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.

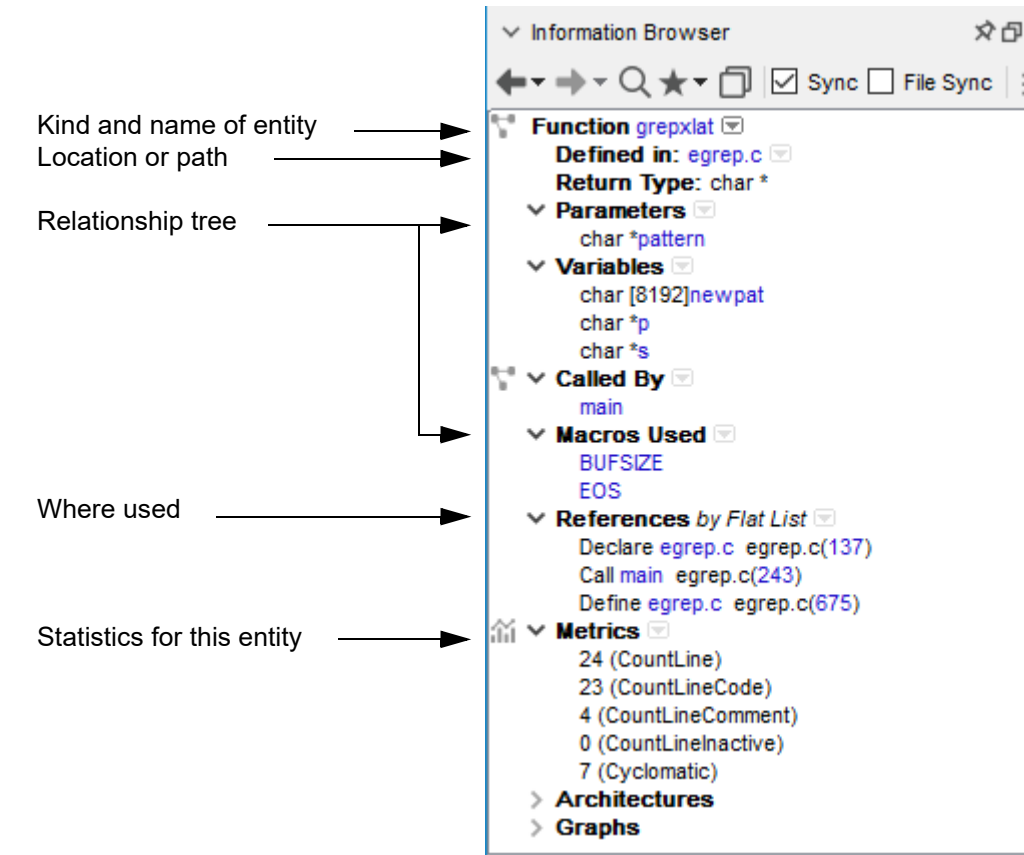
If the Information Browser isn’t open, you can right-click on an item anywhere in *Understand* and choose **View Information**. Or choose **View > Information Browser** from the menus.

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 backtracking.



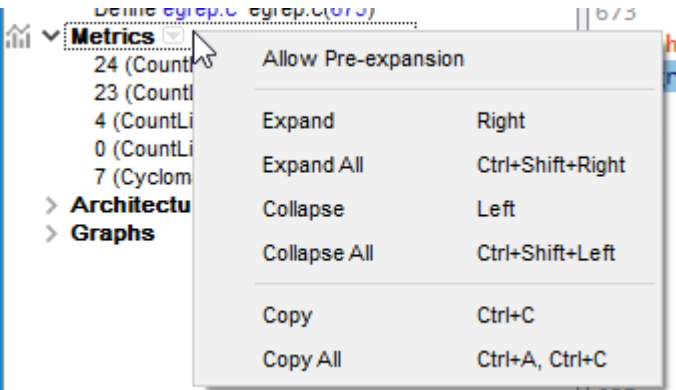
Drilling Down a Relationship

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 144](#) for details on other options in this context menu.



Displaying More or Less Information

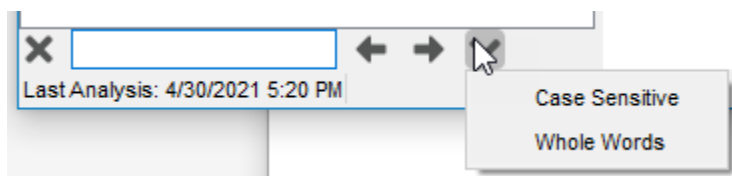
If you click the  icon next to a bold heading such as **Calls**, **Called By** or **References** in the Information Browser (or right-click on the heading), you'll see options that let you modify how that entity is listed. These options include:

- **Allow Pre-expansion:** If enabled, the Metrics node can stay open when you change entities. See *Viewing Metrics* on [page 143](#).
- **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.
- **Fullname:** 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.



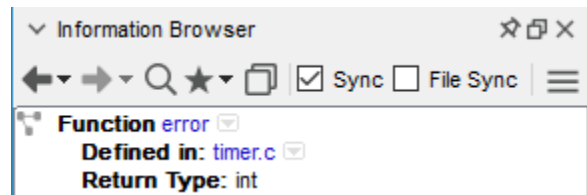
To make the search **Case Sensitive** or to match only **Whole Words**, use the drop-down arrow to select those commands.

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.

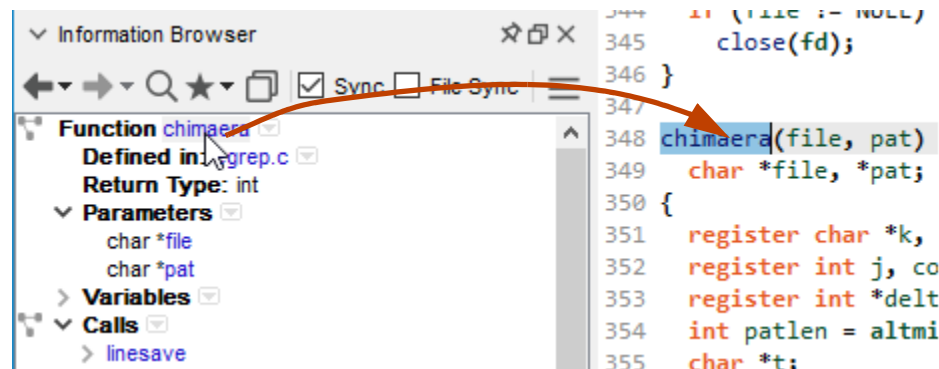
If **Sync** is checked but you want to open a separate Information Browser, hold down the Ctrl key while choosing **View Information** from the right-click menu. You can also open another Information Browser by clicking the  **Copy** icon in the toolbar of the Information Browser

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



Visiting Source Code

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

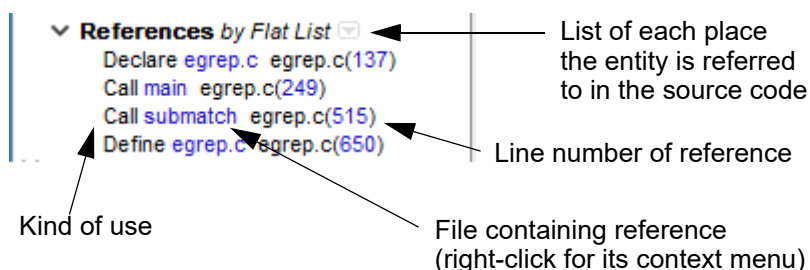


Another way to visit source from any entity you see in *Understand*, is the context menu. Where appropriate, an entity's context menu contains an **Edit Source** (Ctrl+Shift+S) menu item. In some cases, there are separate menu items for **Edit Definition** and **Edit Declaration** (Ctrl+Shift+D) or separate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

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.

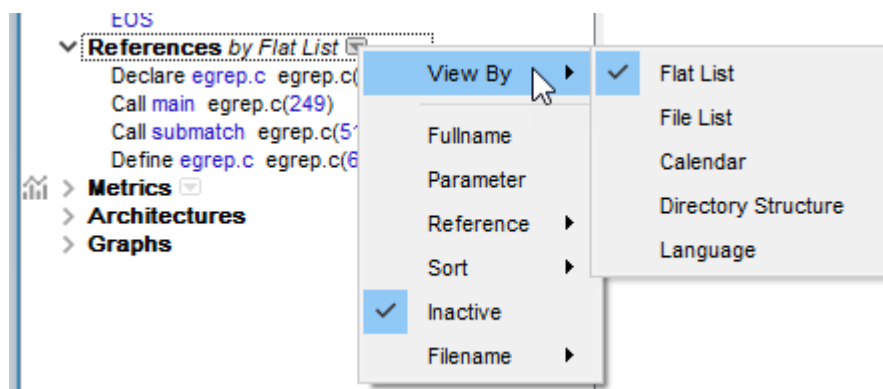
Visiting References

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.

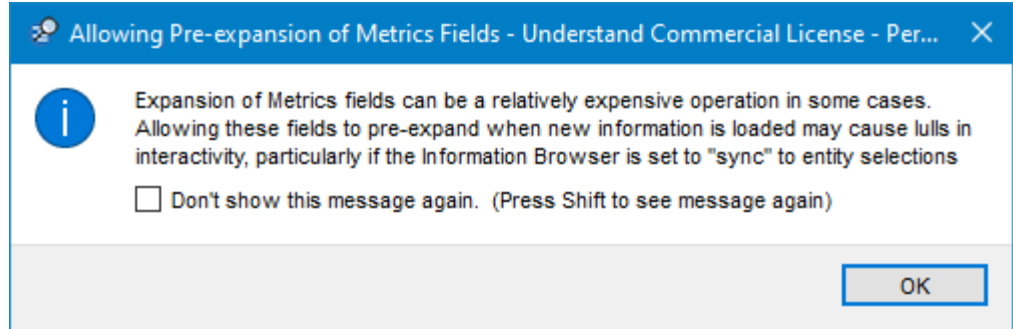


Viewing Metrics

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 246](#) for details on metrics.

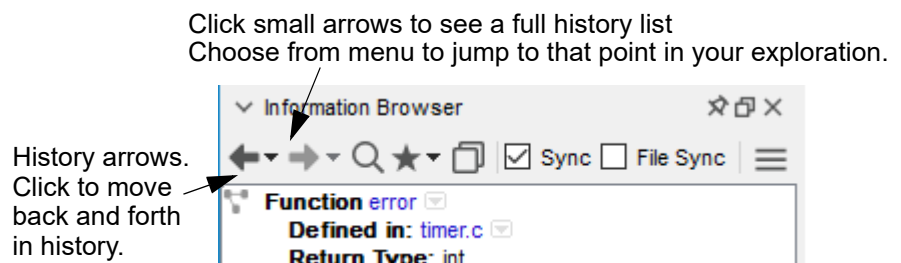
Saving and Printing Information Browser Text

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:

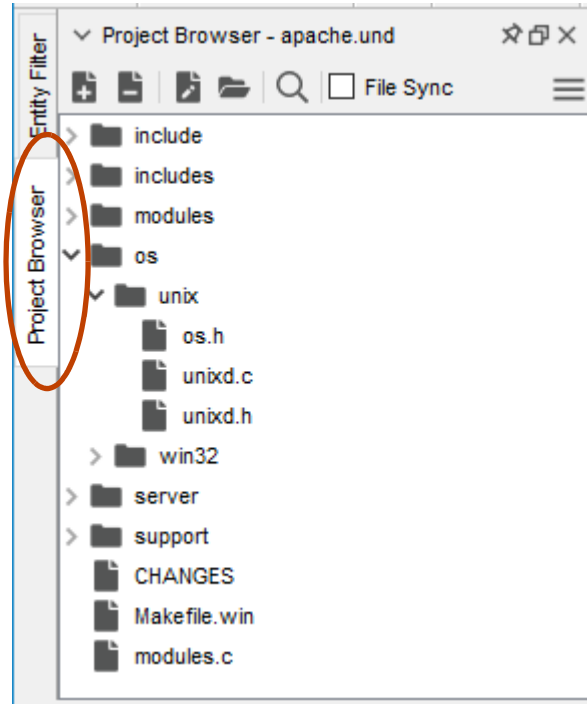


Use the right and left arrows to move back and forward in the history list. The down-arrows show the whole list.

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 From Project	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	
Open OS file browser to show file	Show in File Explorer	
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 **Expand All** and **Collapse All** commands expand or contract the file list.

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

To use the **Remove From Project** 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 **Edit** command opens the selected file in a Source Editor window.

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

The **Copy Filename** command copies the full file path for the selected file to your clipboard.

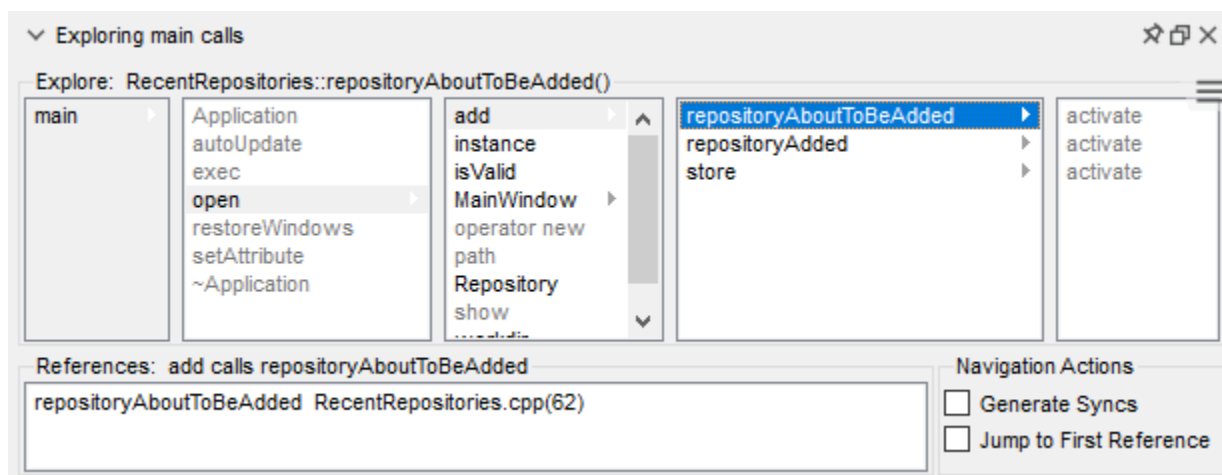
The **Show in File Explorer/Finder** command opens the File Explorer (Windows) or Finder (MacOS) to the folder that contains the selected file.

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 108](#)).

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 if the relationship is set to References From or References To. See [page 220](#) 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. A progress bar shows the completion percentage. 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 152](#) for other ways to export information about dependencies.

Click the  **Favorites** icon to add the current dependency to your Favorites list. See [page 155](#).

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 120](#)).
- **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 are 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?

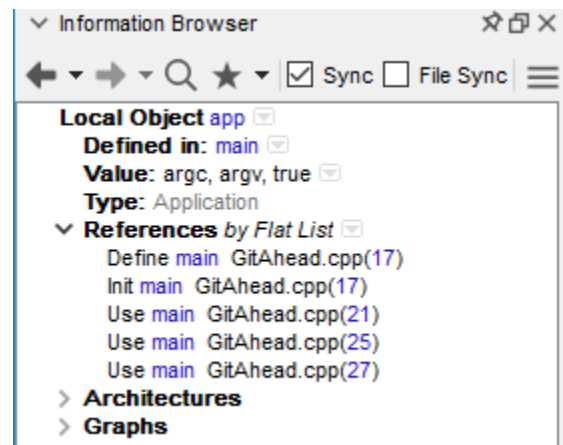
This section provides more detail about how *Understand* determines dependencies for display in the Dependency Browser and in dependency graphs.

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.

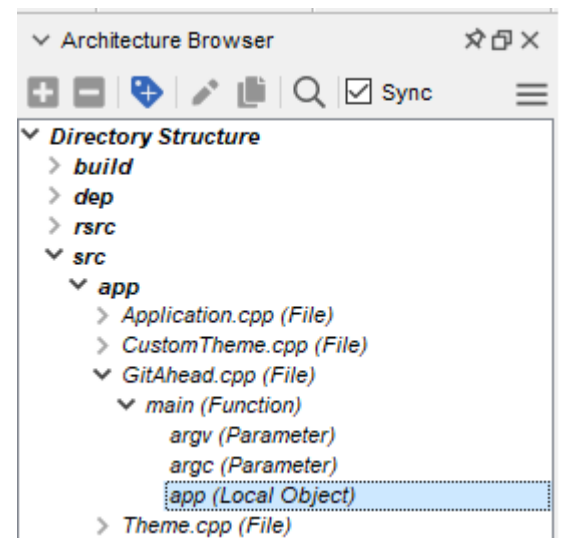
A *dependency* is a connection due to the reference(s) between two files, classes, architecture nodes, and certain other language-specific constructs. For example, file A depends on file B if file A needs file B to compile or link. Whether dependencies are based on compilation or linking is determined by your setting of the Compile Time vs. Link Time option (see *Dependency Category* on page 120).

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**. This report may take some time to process. 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 120](#)). 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.

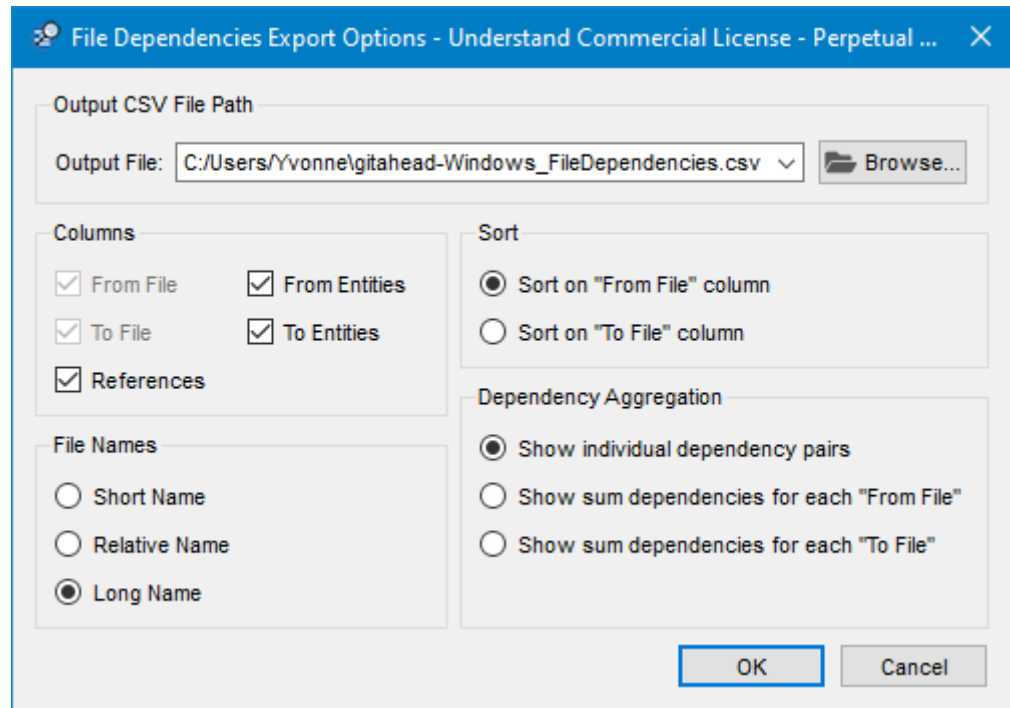
The output for most of these commands is in CSV (comma-separated values) format, which can be opened with most spreadsheet programs. When you create a CSV file with *Understand*, it is automatically opened in a text file window.

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 153](#).)
 - **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 154](#).)
 - **Export Cytoscape XML:** This output format can be opened in Cytoscape (www.cytoscape.org), a free open-source program for analysis and visualization. It draws large diagrams very quickly, and can be useful if you want an overview picture of dependencies in a very large project. (See [page 154](#).)
 - **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. (See [page 285](#).)
- **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 153](#).)
 - **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 154](#).)
 - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See [page 154](#).)
- **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 153](#).)
 - **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 154](#).)
 - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See [page 154](#).)

Exporting Dependencies to a CSV File

When you choose to export a CSV file, you can also set the following options. (This figure shows the dialog for exporting File Dependencies; the other two CSV Export dialogs are very similar.)

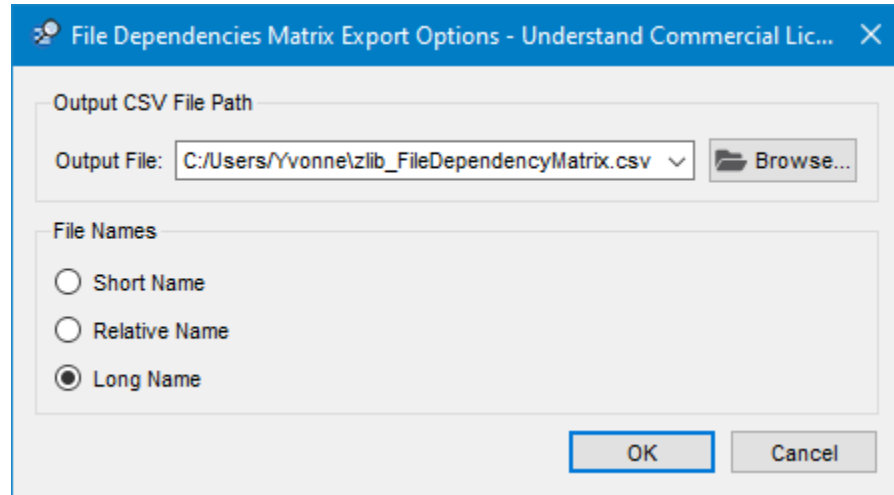


In this dialog, you can set the following options:

- **Select an architecture to analyze:** This option is available only when you are exporting architecture dependencies.
- **Output File:** Browse for the location to save the CSV file.
- **Columns:** Check the boxes for columns you want to include in the output. The “From” and “To” columns for the type of entity you are exporting are required, and cannot be deselected.
- **Names:** Choose a length for entity names. For example, all types can have a short or long name. Files can also have a relative name.
- **Sort:** Choose how you want dependencies sorted. You can sort based on the “From” column or the “To” column.
- **Dependency Aggregation:** Choose how you want to summarize dependency pairs that occur multiple times. You can show each pair individually or sum pairs for the “From” or “To” column for the type of entity you are exporting.

Exporting Dependencies to a CSV Matrix File

When you choose to export a CSV matrix file, you can also set the following options. (This figure shows the dialog for exporting File Dependencies; the other two CSV Export dialogs are very similar.)



In this dialog, you can set the following options:

- **Select an architecture to analyze:** This option is available only when you are exporting architecture dependencies.
- **Output File:** Browse for the location to save the CSV matrix file.
- **Names:** Choose a length for entity names. For example, all types can have a short or long name. Files can also have a relative name.

Exporting Dependencies to Cytoscape

Cytoscape (www.cytoscape.org) is an open source software tool for visualizing complex networks.

When you choose to export a Cytoscape file, you can Browse to select the location and filename for the output file. If you are exporting architecture dependencies, you can also select an architecture to analyze.

Once you have exported the *.xml file, you are asked if you want to open the file in Cytoscape. Note that you can only open Cytoscape if it is installed on your computer. See *Dependency Category* on [page 120](#) for how to configure the location of the Cytoscape installation.

Favorites

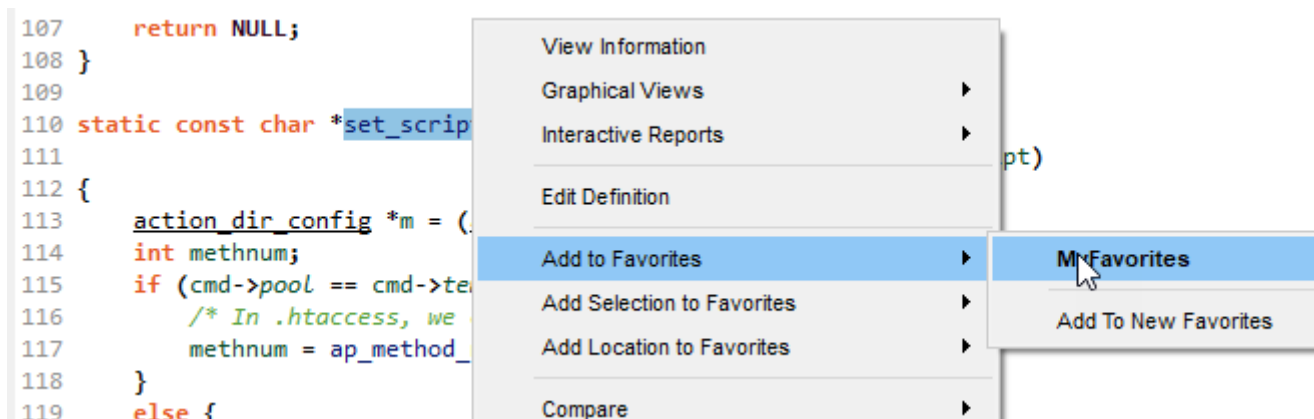
You can mark all kinds of things in *Understand* as “Favorites” so that you can quickly access them as you would web pages in a browser’s Favorites group. Your favorites can include entities, code locations, graphs, Information Browser displays, and dependencies. You can also store multiple plain text 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 203*.

Creating a Favorite Entity

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

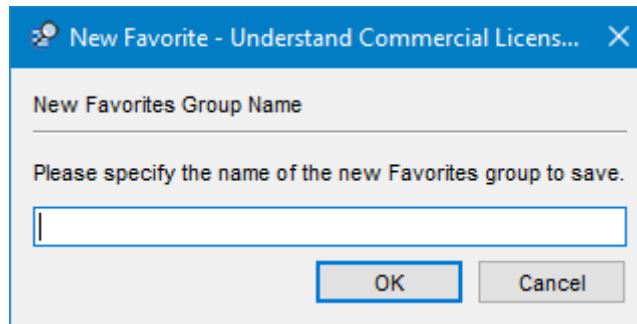
- 1 Select an entity name and right-click in source code, the Entity Filters area, the Information Browser, a graphical view, or anywhere else entities occur.



- 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 158* 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.

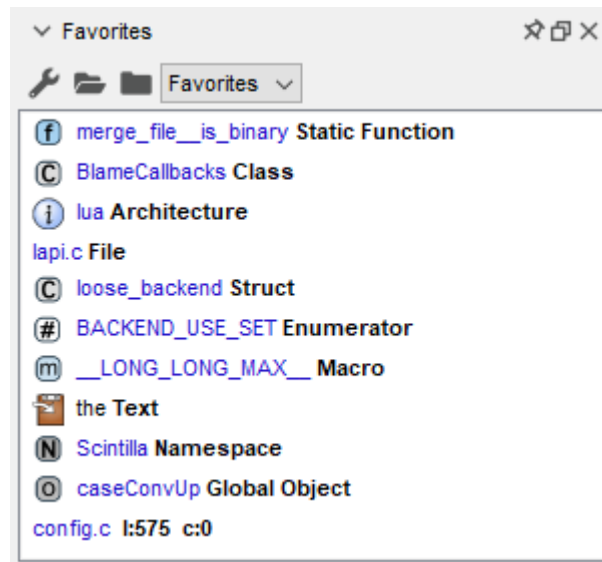
Creating a Favorite View

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

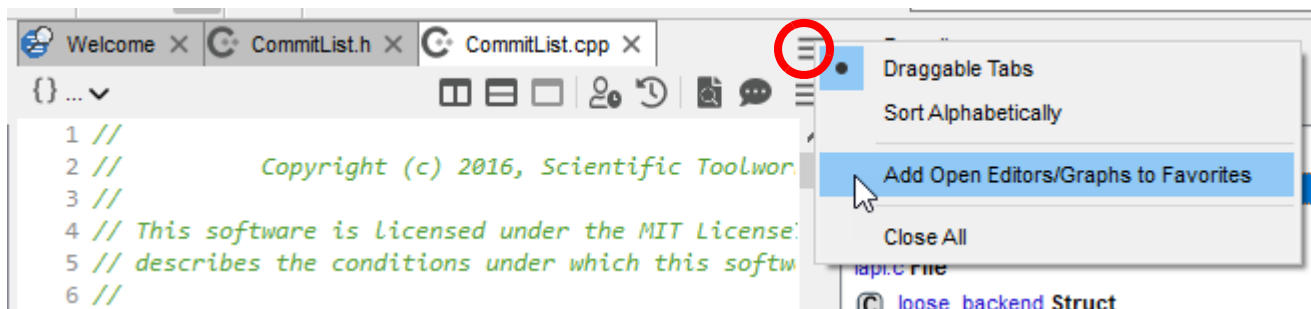
To open a Favorites group, choose **View > Favorites**.



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 158](#) 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 158](#) 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.

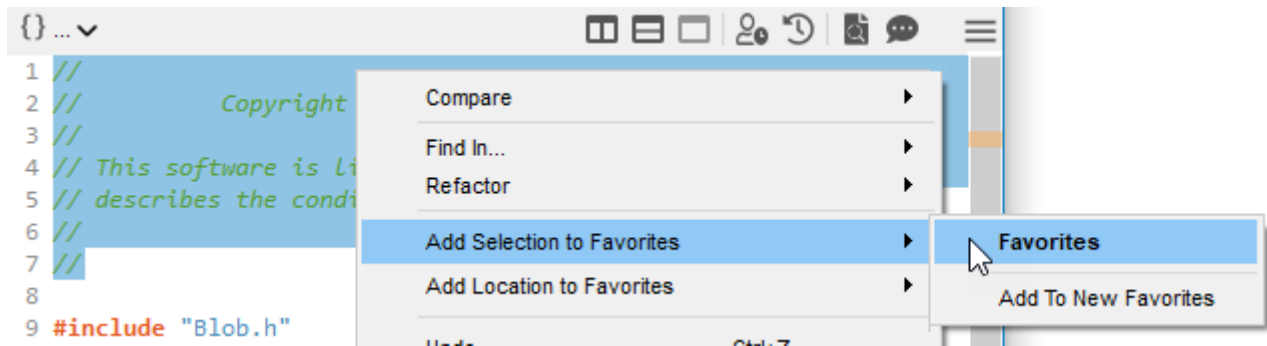
 Delete the current group of favorites.

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.

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

This chapter contains the following sections:

Section	Page
Searching: An Overview	160
Instant Search	161
Find in Files	163
Entity Locator	168
Finding Windows	172
Searching in a File	176

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 168	project-wide, entity-based
Find in Files	page 163	project-wide, text-based
Replace in Files	page 166	project-wide text-based
Instant Search	page 161	project-wide text-based
Find Next and Previous	page 176	single file
Find & Replace	page 176	single file
History	page 173	project-wide
Bracket Matching	page 195	single file
Favorites	page 155	project-wide
Contextual Information Sidebar	page 177	single file
Bookmarks	page 199	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 136](#) for a more complete list of the code exploration tools in *Understand*.

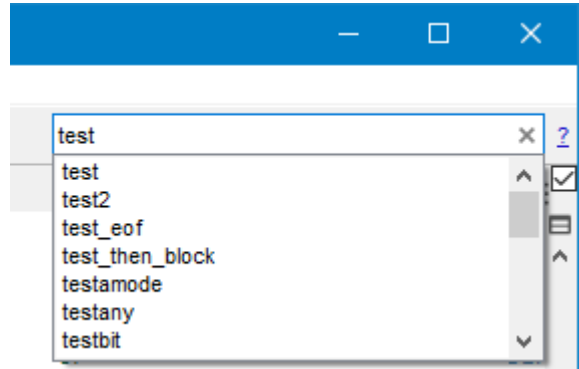
Instant Search

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.

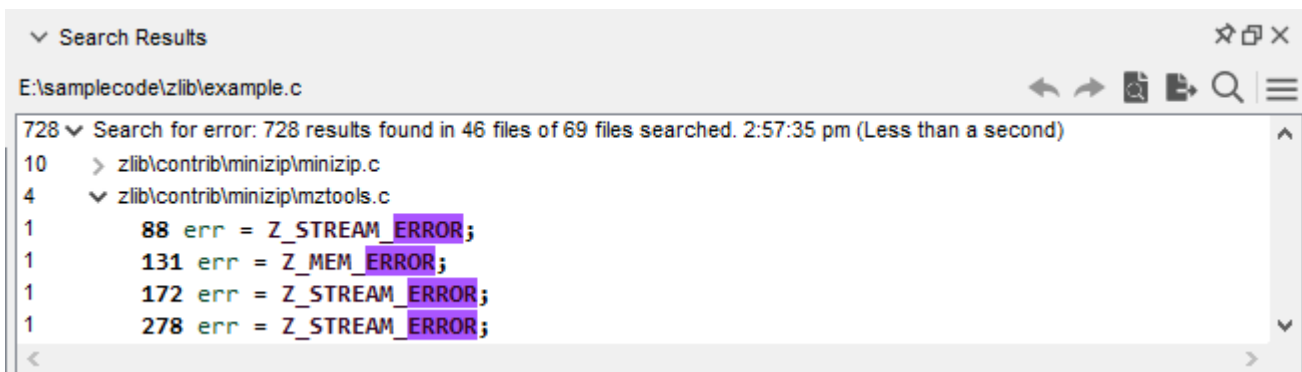
If you don't see this field, choose **View > Toolbars > Search** from the menus.

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 163](#)).

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.



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 for details.

If the results are incomplete or unstable, update the search index by clicking the **Reset Index** button in the Search category of the Project Configuration dialog (see [page 62](#)).

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:

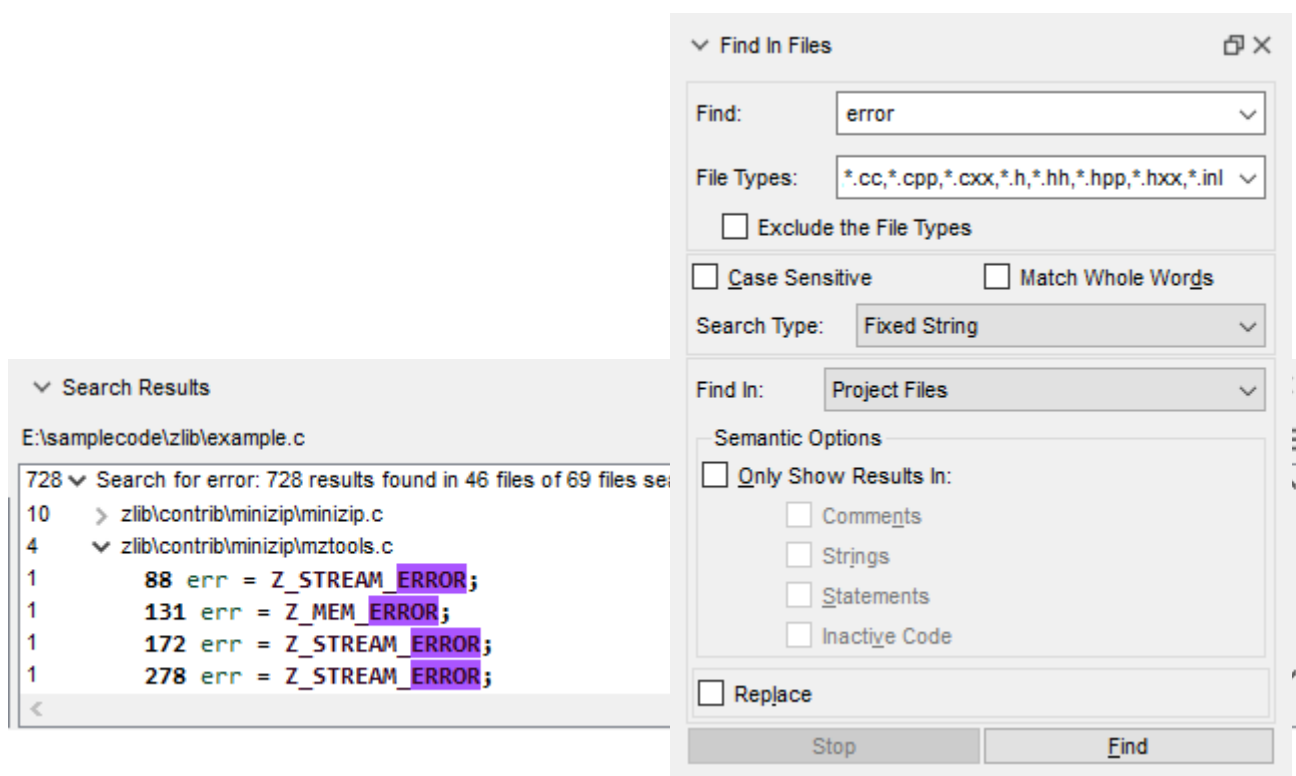
```
(delta0 OR delta1) AND change
```

- 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 166](#)) dialogs by checking the **Replace** box.

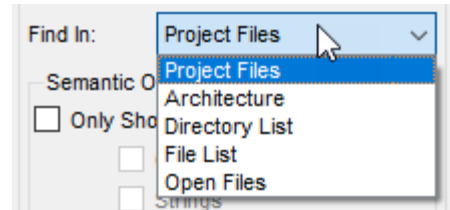
To open the Find in Files tool, choose **Search > Find in Files** from the menu bar, choose **Find in...** from any context menu, or press F5.



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 cases.

- **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 171](#) 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.

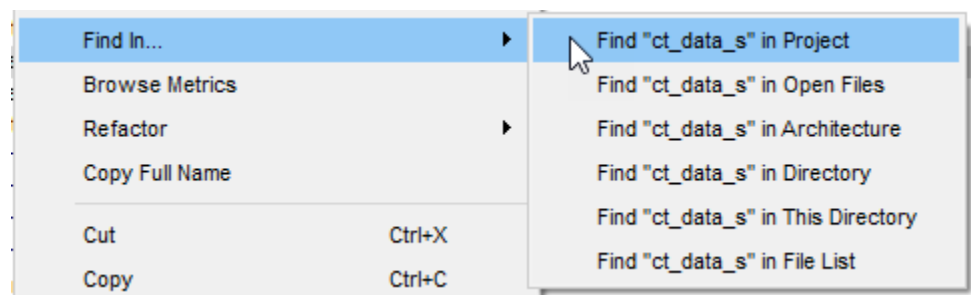
If you select **Architecture**, click **+** to browse for an architecture node.

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.

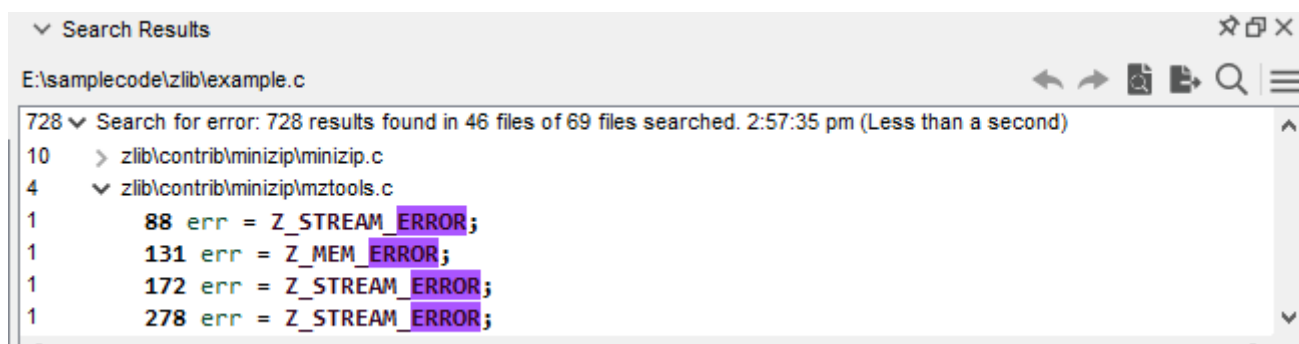


- **Semantic Options:** If you choose to Find In “Project Files” or “Architecture”, you can check the **Only Show Results In** box to be able to control which matches are reported. Then you can check any combination of the **Comments**, **Strings**, **Statements**, and **Inactive Code** boxes to include 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 166](#)) 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.



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 114](#) 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 In Results** Search within the current set of results (using the same search bar described in *Searching the Information Browser* on [page 141](#)).

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.
- **Plain Text** displays the results without highlighting the search string and code as defined in the *Editor > Styles Category* on [page 127](#).
- **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

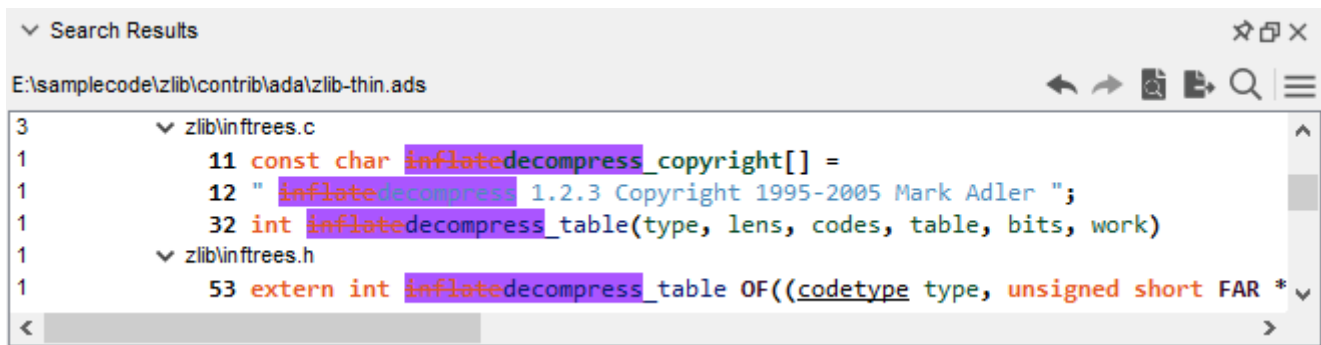
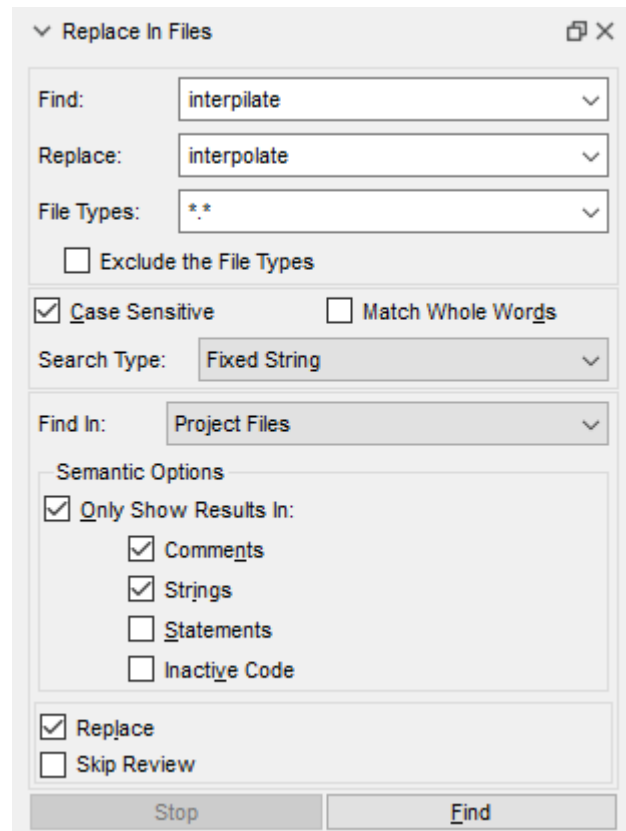
You can use the Replace in Files tool by choosing **Search > Replace in Files** from the menu bar or by checking the **Replace** box in the Find in Files tool.

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

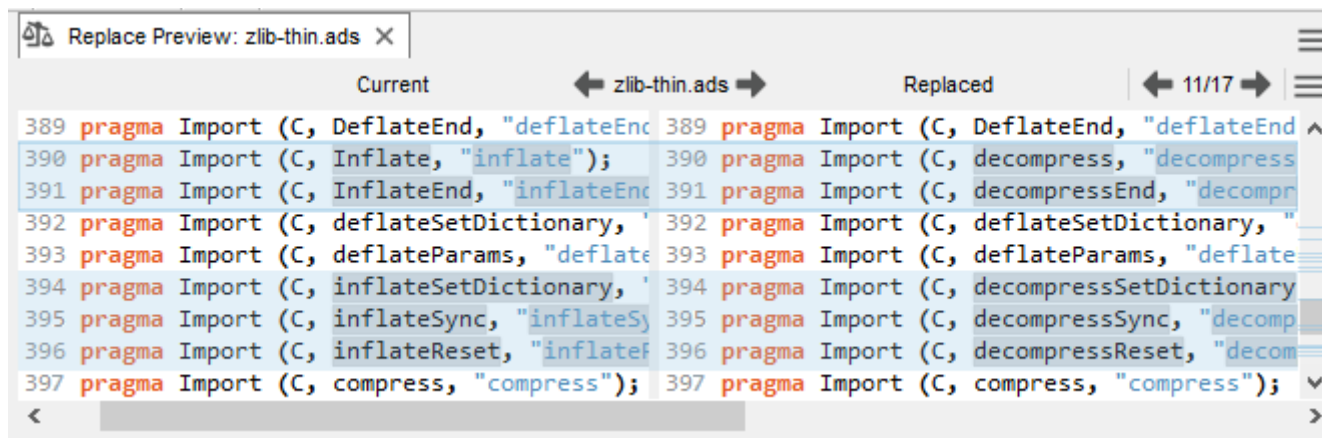
- There is a **Replace** field where you type the text you want to replace the matched string.
- In the **Search Type** field, you can select “Regular Expression - Non Greedy” in addition to the options available for Find in Files.
- To switch between the Replace in Files and Find in Files tools, check or uncheck the **Replace** box just above the Stop button.
- If you are completely sure that you want all the changes to be made, check the **Skip Review** box. The changes will be made immediately when you click **Find**, rather than displayed as shown below to be accepted or dismissed.

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.



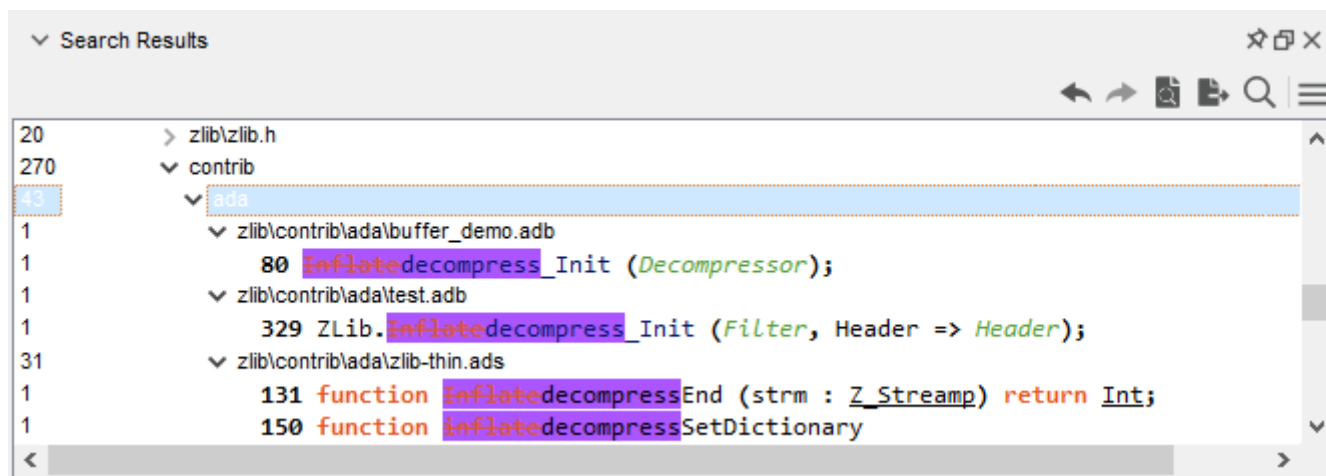
Click on a result to open the Replace Preview view, which shows a comparison of the file that contains the result before and after the text is replaced. This is just a preview; the text has not yet been replaced.



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. **Accept** and **Reject** ✓ ✕ icons are shown at the right side of the Search Results for the selected result, and can be clicked to accept or dismiss a result.

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.

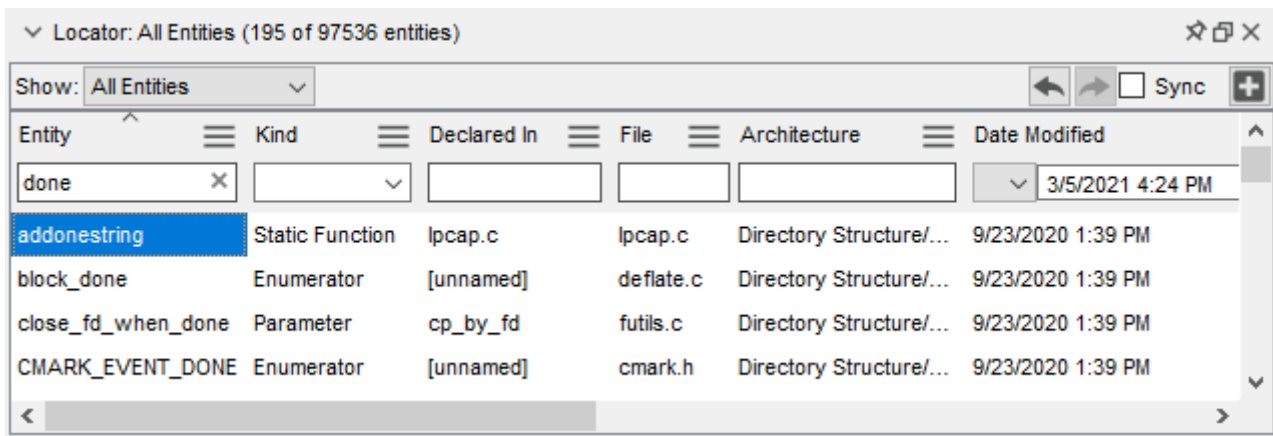


Other right-click commands in the Search Results area allow you to expand and collapse the results tree, copy results, and navigate to the previous or next result, file, or set of results from another search.

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.

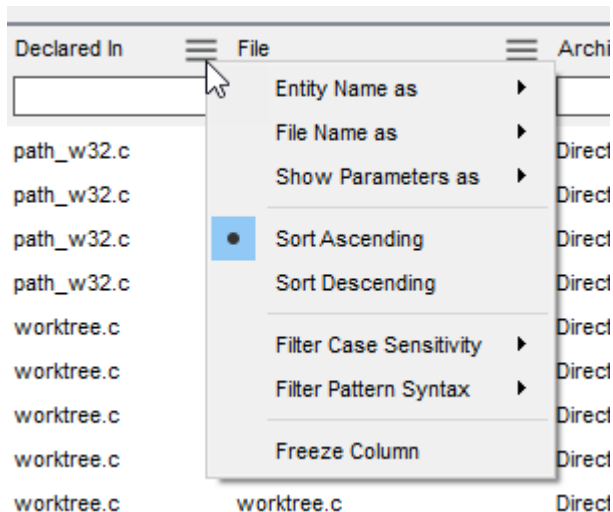
You can right-click on a column header and choose **Freeze Column** to move that column to the left and prevent it from being repositioned.

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

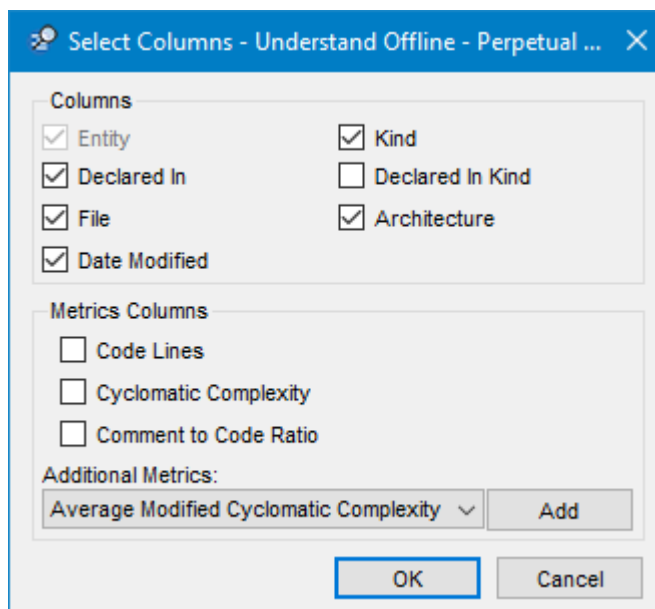
Column headers are tools in the *Entity Locator*. Left-click them to sort according to that column. Right-click a column or click the hamburger menu to see commands that let you control how entities are listed, sorted, and filtered.



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 open the Column Chooser.



The **Entity** column must always be displayed. You can enable or disable other columns, including any available metric or metric plugin. See [page 63](#) to customize the list of available metrics and [page 256](#) for information about adding metric plugins.

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.

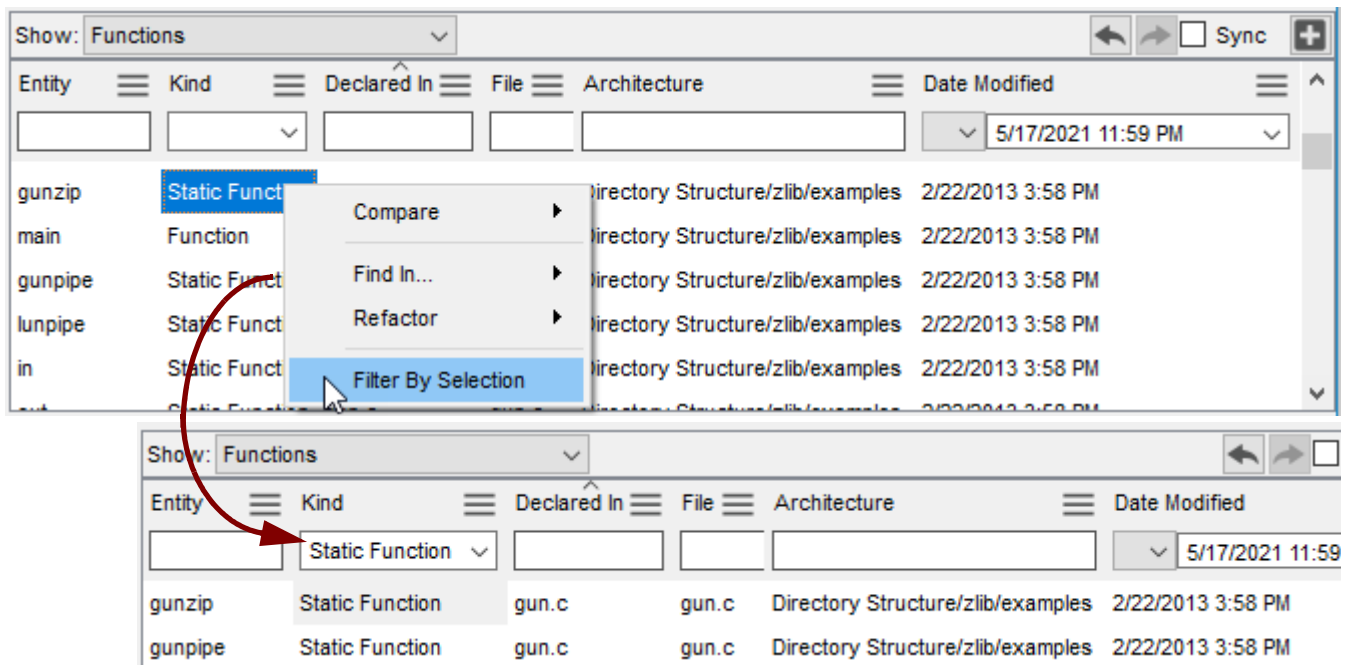
For example, you may filter by the *Kind* column selecting a kind from the drop-down list. You can also right-click on any item listed in the *Kind* column and select **Filter By Selection** from the menu. This filters the list of entities to contain only entities of the kind you selected. The title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

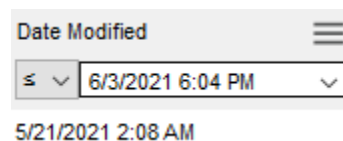
To clear a filter, just delete the text from the field in the column heading or right-click a column header and choose **Clear Filter** or **Clear All Filters**.

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



Similarly, the metrics columns, which you can enable by *Choosing Columns on page 169*, 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.	<i>^word</i> Finds lines with word starting in the first column.
\$	Match at end of a line only.	<i>word\$</i> Finds lines that end with “word” (no white space follows word).
\<	Match at beginning of word only.	<i>\<word</i> Finds wordless and wordly but not fullword or awordinthemiddle.
\>	Match at end of word only.	<i>\>word</i> Finds keyword and sword but not wordless or awordinthemiddle.
.	A period matches any single character.	<i>w.rd</i> 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.	<i>word*</i> Finds word, wor, work, and so on.
+	Match one or more occurrences of the previous character or expression.	<i>wor+d</i> Finds word, worrd, worrrd, and so on.
?	Match zero or one occurrences of the previous character or expression.	<i>wor?d</i> Finds word and wod.


Symbol	Description	Example
[]	Match any one of the characters in brackets but no others.	<code>[AZ]</code> Finds any line that contains A or Z. <code>[Kk][eE][Nn]</code> Finds any variation of case when spelling "Ken" or "KEen" or "keN".
[^]	Match any character except those inside the brackets.	<code>[^AZ]</code> Finds any line that does not contain the letters A or Z.
[-]	Match a range of characters.	<code>[A..Z]</code> 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.	<code>word let+er</code> Finds word, leter, letter, lettter, and so on.
\	Make a regular-expression symbol a literal character.	<code>*\$</code> Allows searching for *. This example finds all lines ending in */

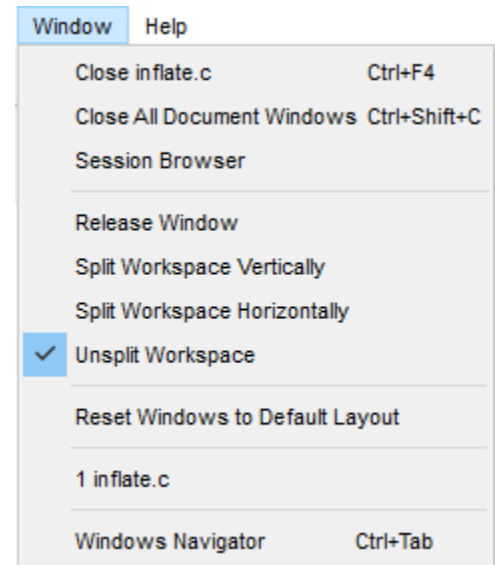
A full explanation of regular expressions is beyond the scope of this manual. Unix users may refer to the manual page for *regex* using the command “*man -k regex*”. For a comprehensive explanation, we refer you to the book “*Mastering Regular Expressions*”, published by O’Reilly and Associates (www.ora.com/catalog/regex).

Finding Windows

If you have 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**.

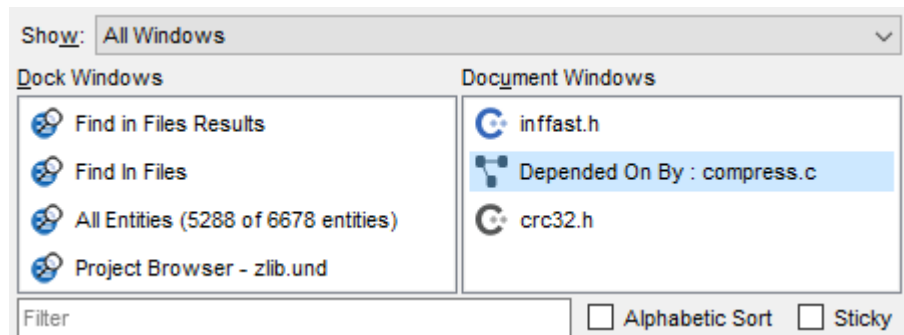
Choose **Window > Release Window** to change the tabbed area to a separate window that can be resized and moved around your screen. Select **Capture Window** from the hamburger menu  in the upper-right corner of a released window to replace the window within the main window.



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.



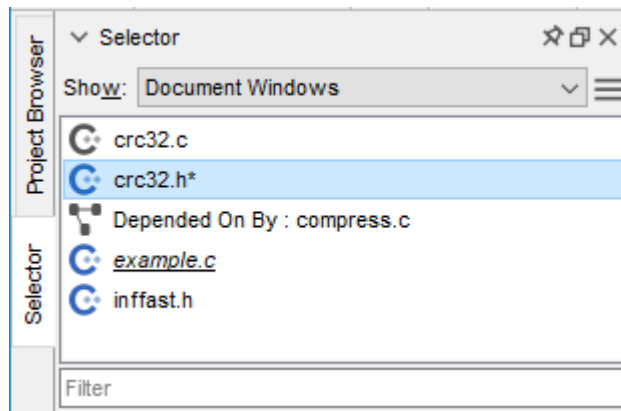
You can click the down-arrows to see the list of source locations in the history. You can choose **Clear History** from the drop-down Go Back list to clear the browsing history.

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** drop-down 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.

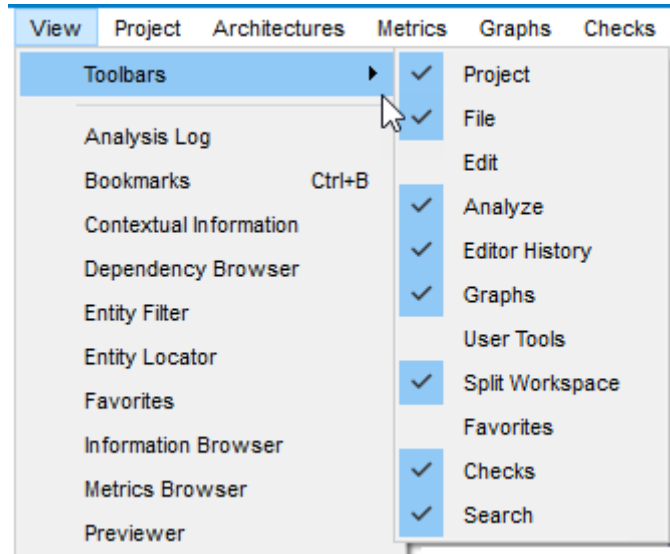
Using the Selector is a convenient way to perform actions—such as Close—on multiple windows by selecting multiple windows from the list, right-clicking, and choosing **Close Selected Window(s)** or **Close Unselected Window(s)**.

If you have created bookmarks in your source code ([page 199](#)), you can use the **View > Bookmarks** command to open the list of bookmarks.

Displaying Toolbars

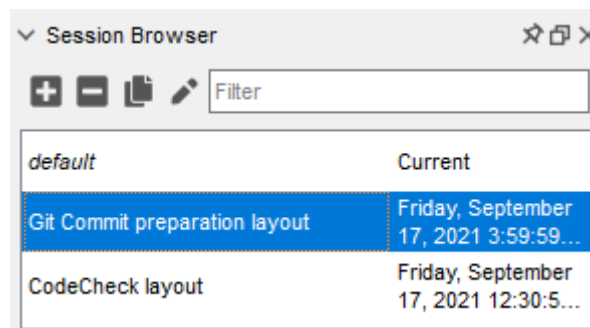
You can hide or display categories of toolbar icons in the main *Understand* window by right-clicking on the toolbar or menu bar and choosing a category. The toolbar is separated into the following categories: Project, File, Edit, Analyze, Editor History, Graphs, User Tools, Browse, Split Workspace, 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 175](#)). 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**.

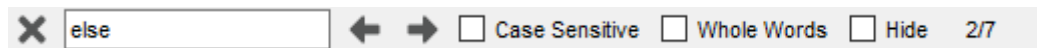
Session information is stored in the `session.json` file in the project folder and may be shared with other users.

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. Up to 1024 matches are also highlighted in the scroll bar; see the Radar Scroll Bar area in *Editor > Advanced Category* on [page 123](#). If the string does not exist in the file, the search field turns red.

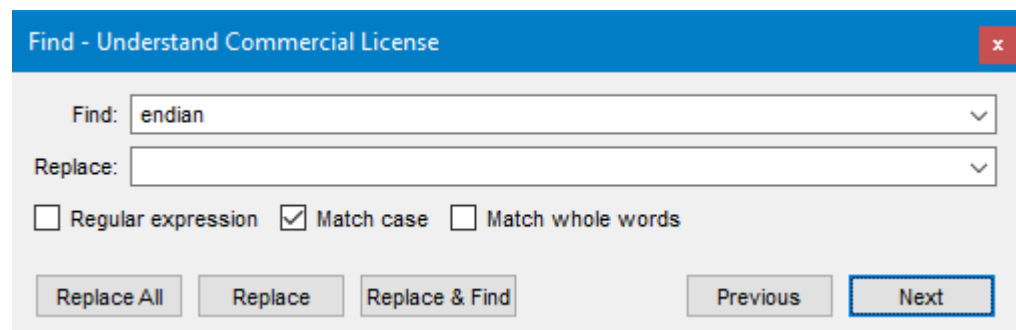
Use Ctrl+F to move to the next occurrence. Use Ctrl+Shift+F to find the previous occurrence.

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 170](#).

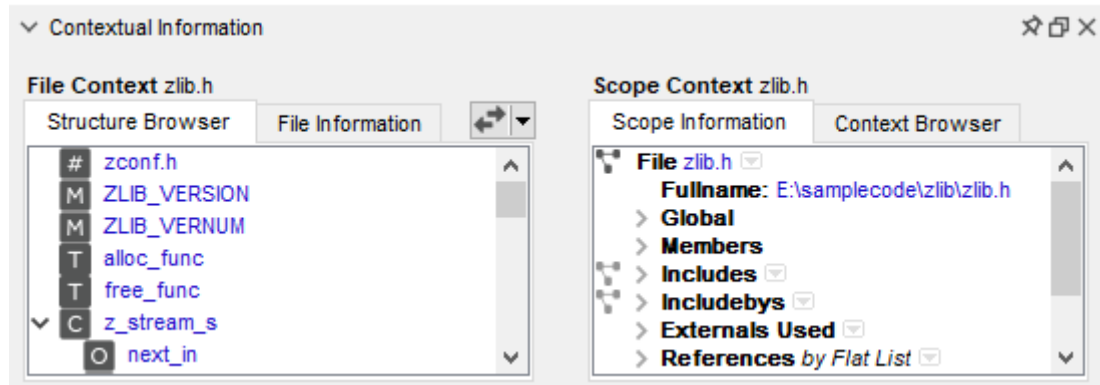
If you want to replace the string you are finding, type that in the **Replace** field.

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 163](#).


Contextual Information

The Contextual Information view is similar to the Scope List (see [page 180](#)), 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.

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

This chapter contains the following sections:

Section	Page
Source Editor	179
Saving Source Code	184
Refactoring Tools	185
Other Editing Features	194
Annotations	203
Printing Source Views	208

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.

```

egrep.c X
{} ... v
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 */
498
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 121](#)). 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 123](#)) lets you control the format when you print from the Source Editor, highlights in the scroll bar (radar) when you search, copy-paste behavior, auto-completion, auto-indenting, and more.

The **Editor > Styles** category of the Understand Options dialog (see [page 127](#)) lets you change the colors and styles used for different types of source code. The **Key Bindings** category (see [page 116](#)) shows a list (and lets you modify the list) of keystrokes you can use in the Editor.

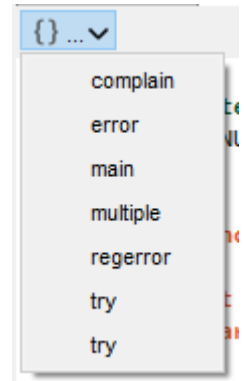
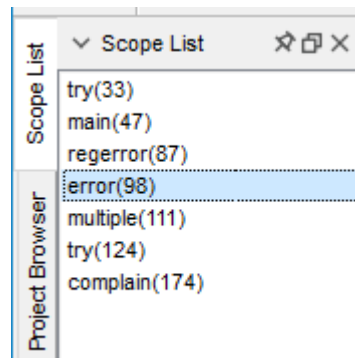
File Icons

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. If the file has been deleted in the file system, the filename in the tab is shown as strikethrough text.

Scope List

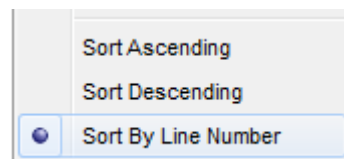
Scope lists are useful for jumping around in large files. You can jump to a particular function, procedure, or other language-specific construct in the current source file by selecting from the scope drop-down list in the toolbar. Constructs are listed alphabetically. The drop-down list shows all such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

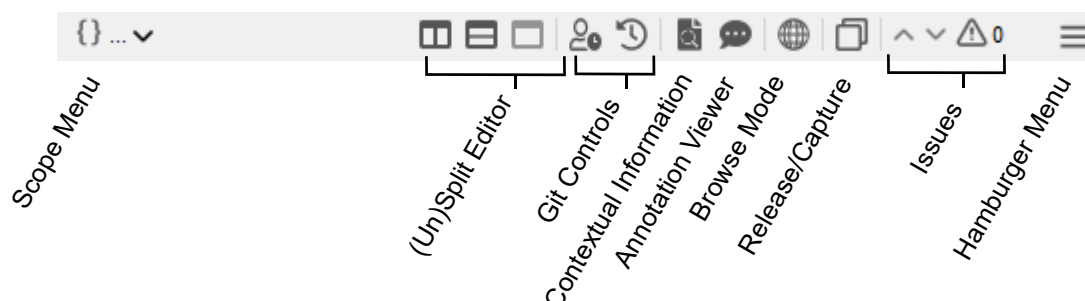
You can right-click on the Scope tab to choose a sort order from the context menu. The ascending and descending orders sort alphabetically or reverse alphabetically. The default is to sort by line number.



For more power than the scope list, use *Contextual Information* on [page 177](#).

Toolbar

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

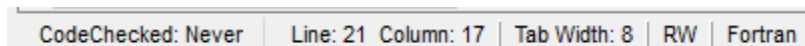


You can hide or display all of the icons in the Source Editor toolbar using the **Toolbar Sections** category in the hamburger menu. The toolbar sections available are:

- **Scopes:** Select a function or similar entity to jump to. See *Scope List* on [page 180](#).
- **Split Editor:** Split the Source Editor window horizontally, vertically, or remove a split. See *Splitting the Editor Window* on [page 197](#).
- **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 326](#) for details.
- **Dock Windows:** Show/hide the Contextual Information view and the Annotations viewer. See *Using Understand Windows* on [page 19](#).
- **Browse Mode:** Toggle Browse mode on or off. See [page 182](#).
- **Released Windows:** Release/capture window within the main window. See *Managing Source Editor Tabs* on [page 201](#).
- **Issues:** Move to next/previous issue found in source code. These issues may be detected by CodeCheck or project analysis. Click the **Caution** icon to open the Issue sidebar, which provides issue descriptions. See *Viewing Results in Violation Browser* on [page 302](#).

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.

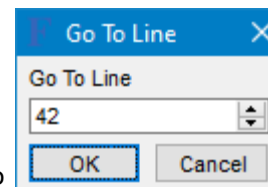


If you click the line number in the status bar (or choose **Search > Go to Line** from the menus), you see the Go To Line dialog.

If you click the Tab Width, you can change the tab width for this file. See [page 197](#).

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 <ctype.h>
46      #include <sys/types.h>
47      #include <sys/stat.h>
48      #include "regex.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** icon in the editor toolbar or choosing **View > Browse Mode** from the menus. When you are in Browse mode, the icon is highlighted.



If you do not see the **Browse** icon in the Source Editor toolbar, choose **Toolbar Sections > Browse Mode** from the Source Editor's  hamburger menu.

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 128](#) 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 139](#))
- Graphical Views (see Chapter 11)
- View Dependencies (see [page 148](#))
- Interactive Reports (see [page 343](#))
- Add to Favorites (see [page 155](#))
- Compare (see [page 319](#))
- Analyze (see [page 102](#))
- Add/Remove to Architecture (see [page 215](#))
- Annotate (see [page 203](#))
- Explore (see [page 147](#))
- Find in... (see [page 163](#))
- Browse Metrics (see [page 248](#))
- Hide Inactive Lines (see [page 196](#))
- Add Bookmark (see [page 199](#))

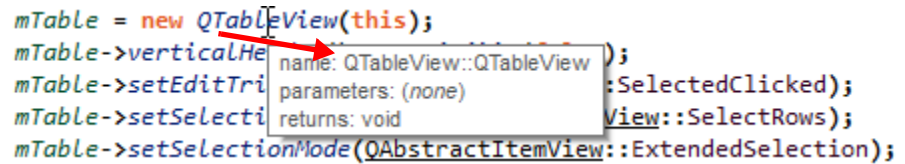
The following *editing features* are also typically included in the context menu:

- Refactor (see [page 185](#))
- Undo / Redo
- Cut / Copy / Paste (see [page 182](#))
- Copy Full Name (see [page 182](#))
- Select All (see [page 182](#))
- Remove from Project (see [page 51](#))
- Edit Files (see [page 142](#))
- Jump to Matching Brace (see [page 195](#))
- Select Block (see [page 195](#))
- Hide/Show Inactive Lines (see [page 196](#))
- Fold All (see [page 196](#))
- Soft Wrap (see [page 198](#))
- Comment Selection / Uncomment Selection (see [page 196](#))
- Sort Selection (see [page 198](#))
- Change Case (see [page 196](#))
- Reindent Selection/File (see [page 197](#))
- Revert (see [page 184](#))

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->verticalHeader()->setEditTriggers(QAbstractItemView::SelectedClicked);
mTable->setSelectionMode(QAbstractItemView::ExtendedSelection);
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**.

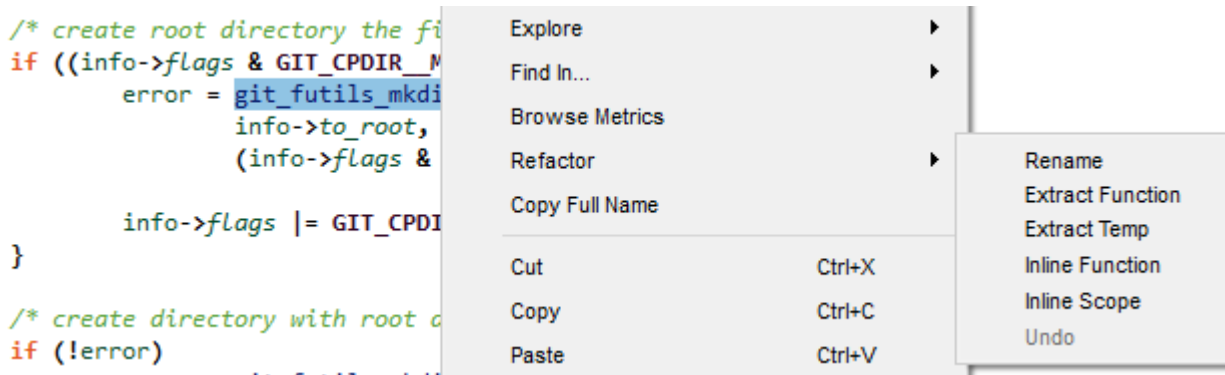
You can close the current source file by choosing **Window > Close <current_file>** from the menus. You can also middle-click on the tab above the source file area to close that tab (if your mouse has a middle button).

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.

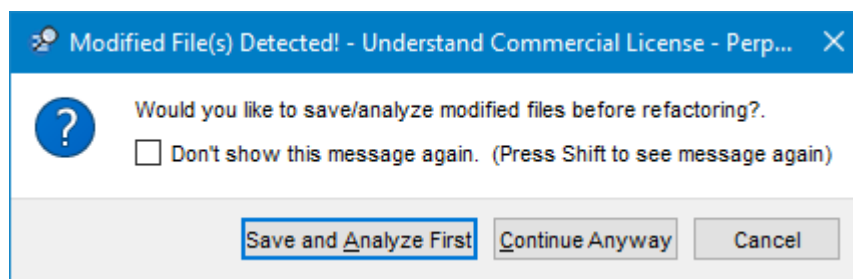
The refactoring tools allow you to preview the changes using a code comparison window. Refactoring changes can have significant effects on code, and should be reviewed before committing to make certain that the changes are correct.



The following refactoring tools are available if they apply to the selected entity or code:

- **Rename Entities:** [page 186](#) (various languages)
- **Rename Files:** [page 187](#)
- **Inline Function:** [page 189](#) (C/C++)
- **Extract Function:** [page 190](#) (C/C++)
- **Inline Scope:** [page 191](#) (C/C++)
- **Inline Temp:** [page 192](#) (C/C++)
- **Extract Temp:** [page 193](#) (C/C++)
- **Ignore Violations:** [page 307](#) (all languages)

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 contains 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

If you want to rename an entity throughout your project, we recommend that you use the **Refactor > Rename** command instead of **Replace in Files** ([page 166](#)).

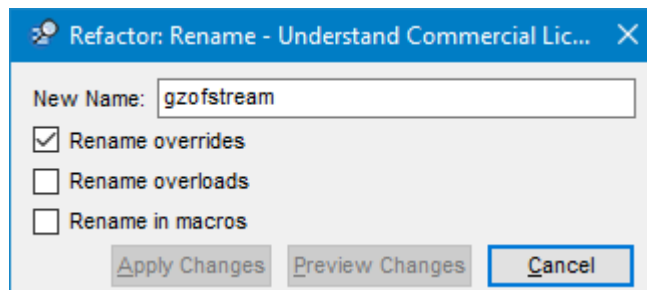
The difference between the two is that the **Refactor > Rename** command determines which uses apply to that specific entity. This makes it a better way to rename such things as variables that are used locally, entities in applications with a variety of namespaces, and entities with names that may be a part of another entity name (for example, renaming a “src” variable without renaming “srcTimer”).

Note that the Refactor command does not modify code comments that mention the entity.

(You can use the same Rename command to rename files; see *Renaming Files* on [page 187](#).)

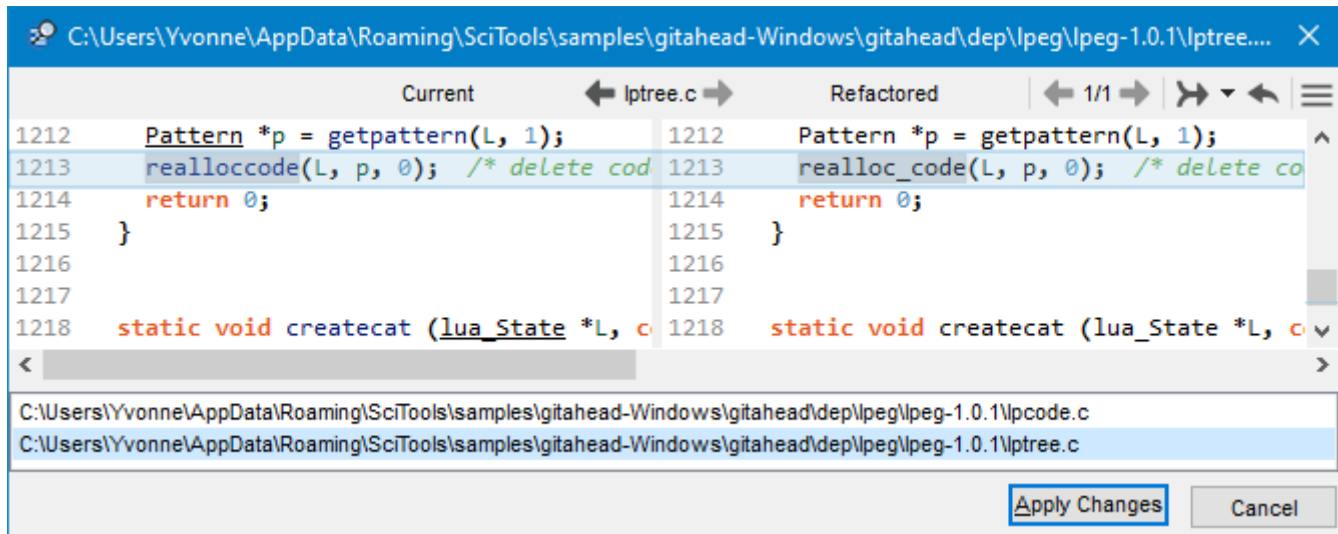
To use the **Refactor > Rename** command, follow these steps:

- 1 Highlight the name of the entity (for example, a function or argument) you want to change.
- 2 Right-click and choose **Refactor > Rename** from the context menu.
- 3 Type a new name for this entity in the dialog.
- 4 If the selected entity is a C++ override or a macro, additional options are provided:
 - If the selected entity is a class method for which the base class is declared as virtual and a derived class also implements the method, you can check **Rename overrides** to rename the method override.
 - If the selected entity is a class method for which the override has the same name but different parameters, you can check **Rename overloads** to rename the derived method.
 - If the selected entity has a macro with the same name, you can check **Rename in macros** to rename both the entity and its matching macro.



- 5 If you are absolutely sure you want to perform the renaming operation, click **Apply Changes**. Otherwise, click **Preview Changes**.

- 6 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 328](#) describes the icons and drop-down menus in this dialog.



- 7 If you are sure you want to make all the changes, click **Apply Changes**.

Renaming Files

If you want to rename a file throughout your project, you can use the **Refactor > Rename** command. In addition to renaming the file in the *Understand* project, the file is renamed in your file system and moved to a new location if you choose one when renaming the file.

If a file you are renaming is open and has been modified, the file will be saved and changed files will be analyzed as part of refactoring.

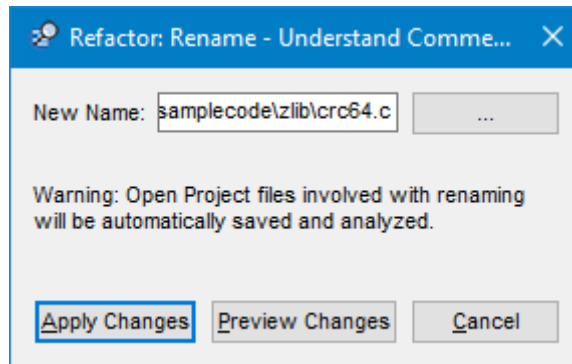
Note that the Refactor command does not modify code comments that mention the file name.

(You can use the same Rename command to rename entities; see *Renaming Entities* on [page 186](#).)

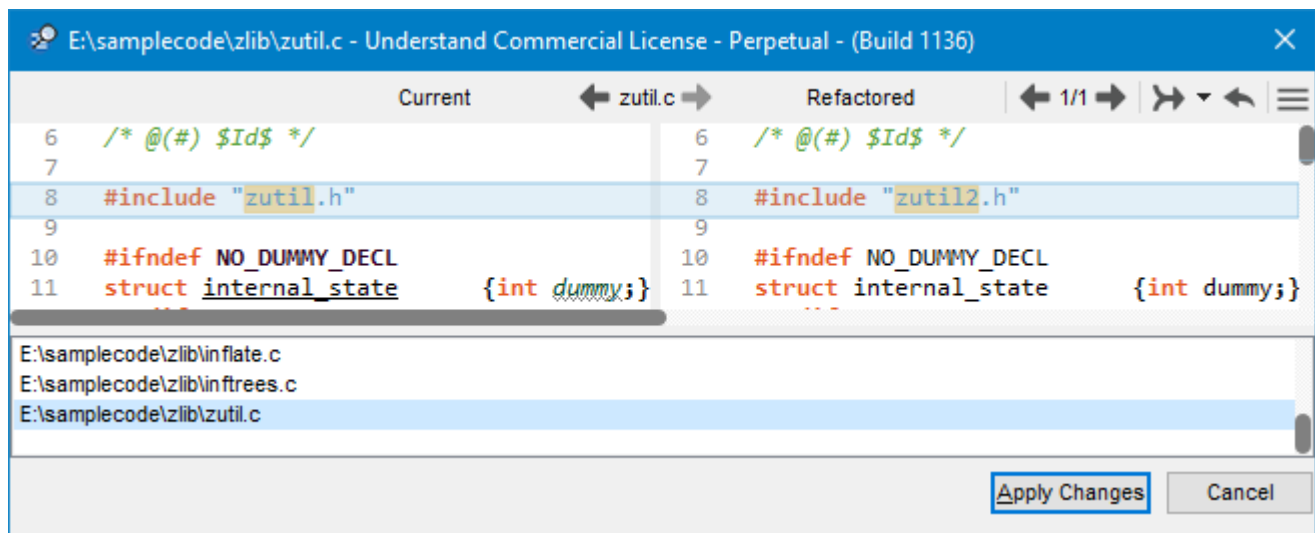
To rename or relocate a file, follow these steps:

- 1 Highlight the name of the file you want to change anywhere in *Understand*. For example, you can select file names in the Project Browser, Architecture Browser, Information Browser, or a source code file.
- 2 Right-click and choose **Refactor > Rename** from the context menu.

- 3 In the Refactor: Rename dialog, edit the name of the file or the full file path as needed. Or click the ... button and browse to the new location for the file.



- 4 If you are absolutely sure you want to perform the renaming operation, click **Apply Changes**. Otherwise, click **Preview Changes**.
- 5 A dialog opens that lets you examine all instances where the file name is used in source code and how it will be changed. *Exploring Differences on page 328* describes the icons and drop-down menus in this dialog. If the file name is not used within source code files, the dialog will be empty, but the Apply Changes button still works.



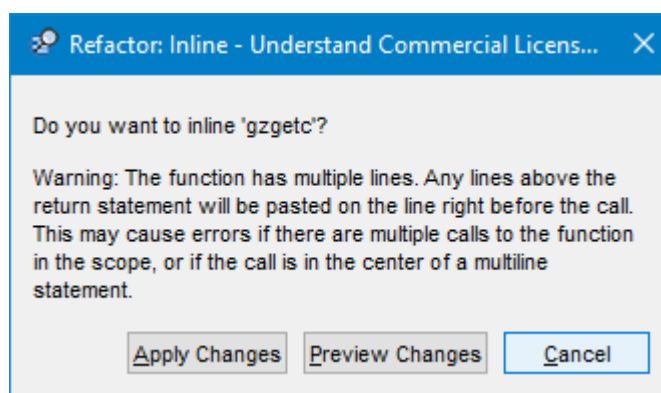
- 6 When you are sure you want to make all the changes, click **Apply Changes**. In addition to any code changes, the file will be renamed and/or moved in your file system.

Inlining Functions

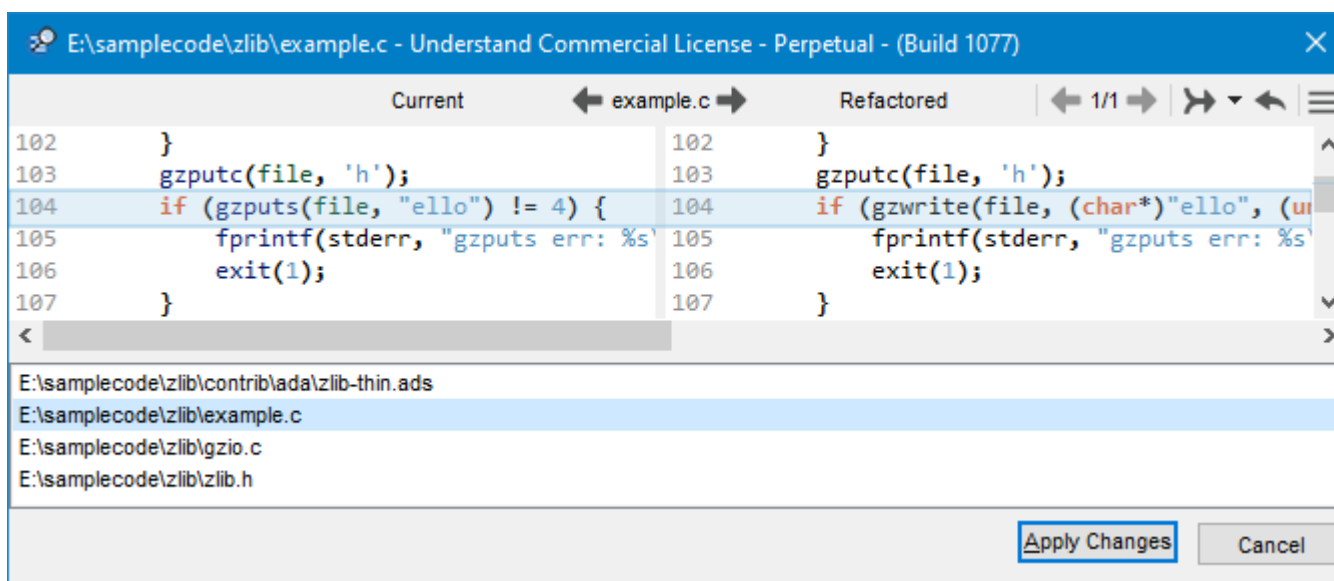
Inlining functions is a common optimization technique that places the code of a function at the location where it is called instead of in a separate function. In compiled languages, this can often be performed through compiler optimization, but inlining the source code may be useful for various reasons, 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 328](#) describes the icons and drop-down menus in this dialog.



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

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.

Function Name:

Return:

Parameters:

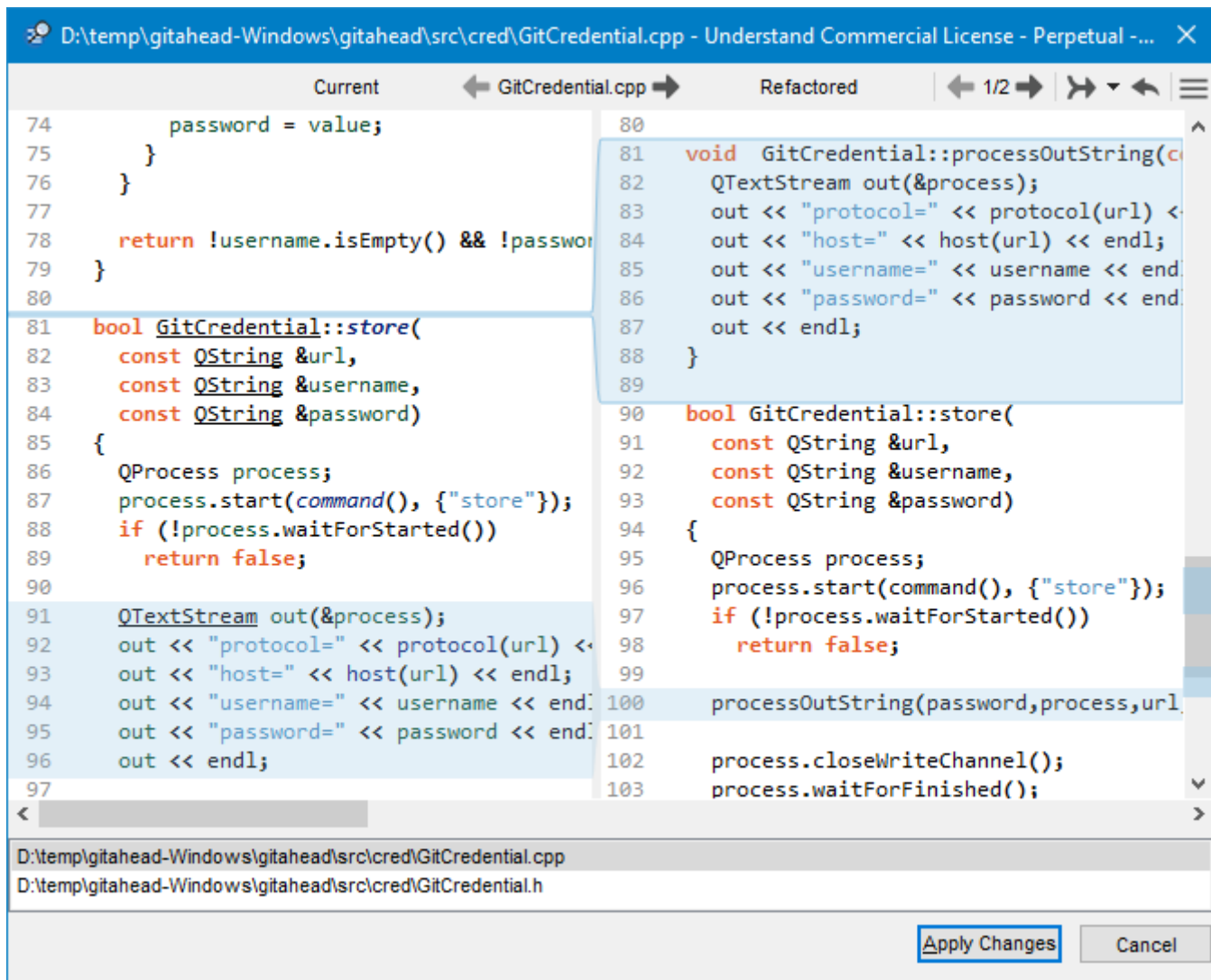
Entity	Name	Const	&
free_func zfree	zfree	<input type="checkbox"/>	<input type="checkbox"/>
voidpf opaque	opaque	<input type="checkbox"/>	<input type="checkbox"/>
alloc_func zalloc	zalloc	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Move Up
Move Down

Apply Changes Preview Changes Cancel

- 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 328* describes the icons and drop-down menus in this dialog.



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

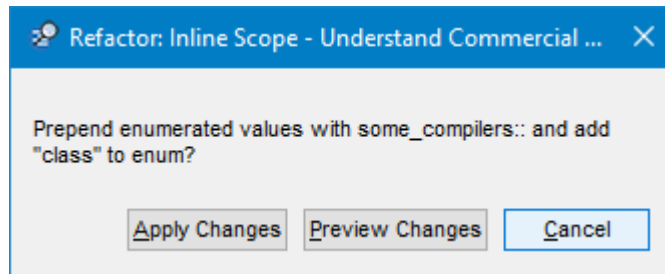
Inlining Scope

Inline scope refactoring can be used to change a regular enum to an enum class or to remove "using namespace" statements. The enum declaration will be changed to an "enum class" declaration and the name of the enum will be prepended to enumerated values wherever they are used.

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

- 1 Highlight the name of an enum in source code or a tool such as the Information Browser.

- 2 Right-click and choose **Refactor > Inline Scope** from the context menu.



- 3 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 that will occur. *Exploring Differences* on [page 328](#) describes the icons and drop-down menus in this dialog.

Inline Temp

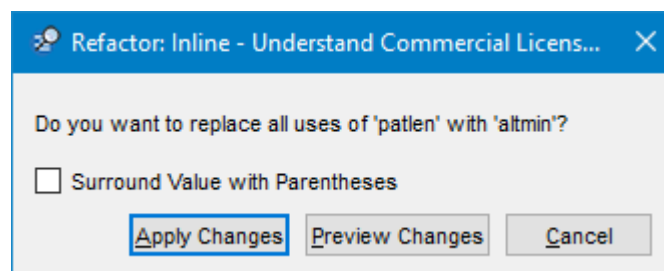
Inline temp refactoring can be used with a local or temporary variable that is initialized and never set after that. The inlining replaces uses of that variable with the expression to which it is initialized. In the following example, `patlen` could be inlined as `altmin`, so long as the value of `patlen`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

```
int patlen = altmin;

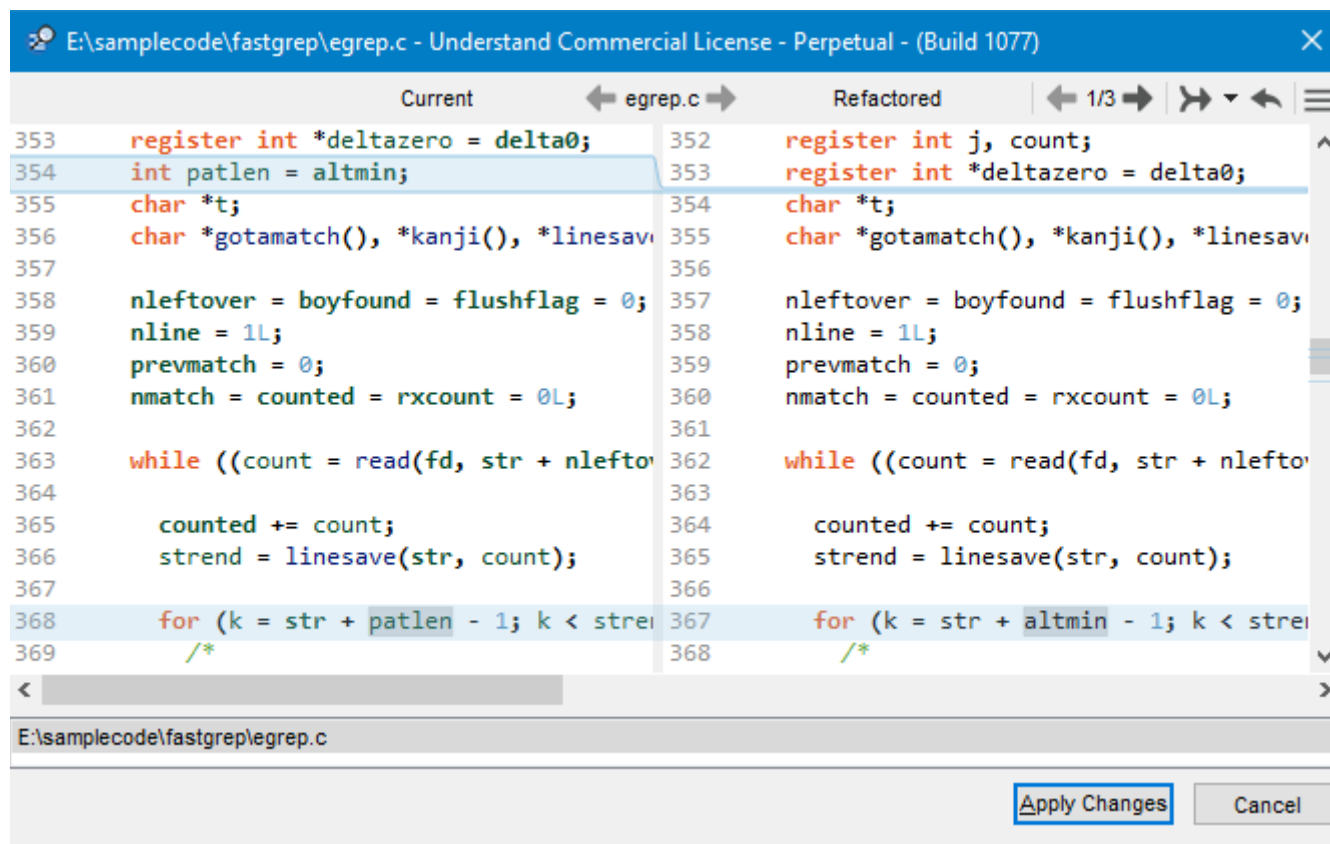
for (k = str + patlen - 1; k < strend;) {
    ...
}
```

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 328* describes the icons and drop-down menus in this dialog.



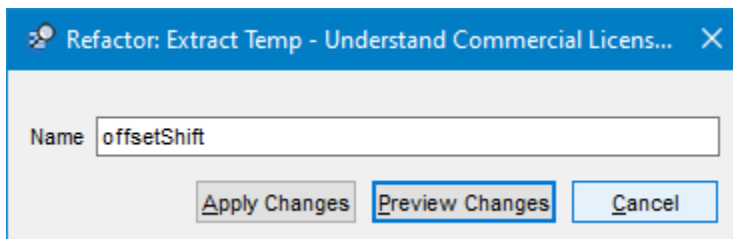
- 6 If you are sure you want to make all the changes, click **Apply Changes**.

Extract Temp

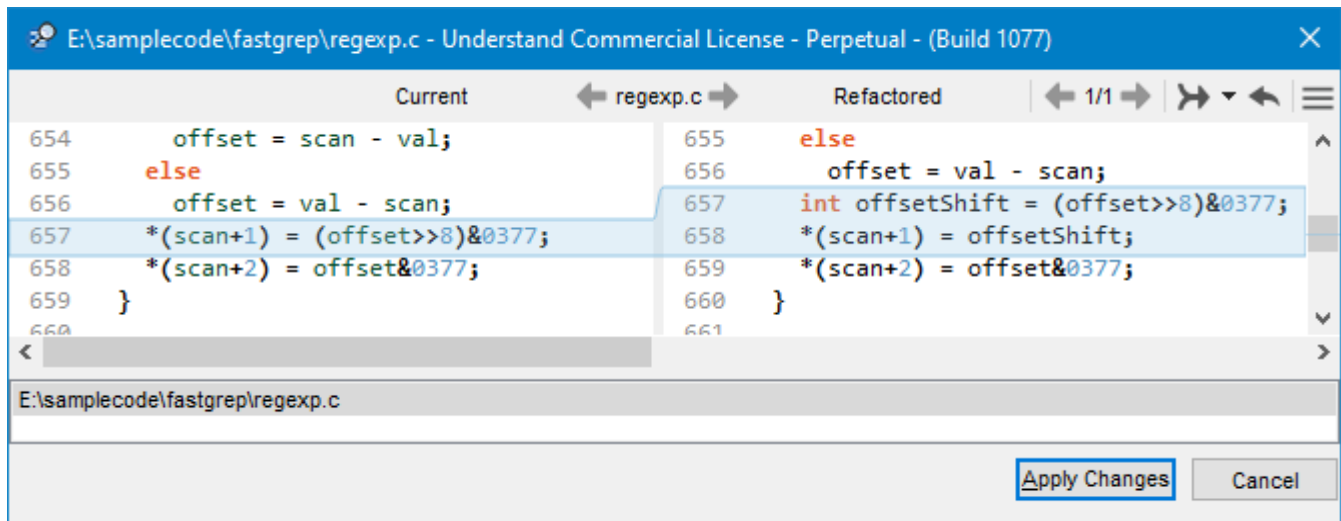
The opposite of inline temp is extract temp. If you have a complicated expression, you may want to assign a part of that expression to a local or temporary variable that can be reused within that function wherever the expression you select is used.

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 328](#) describes the icons and drop-down menus in this dialog.



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

Other Editing Features

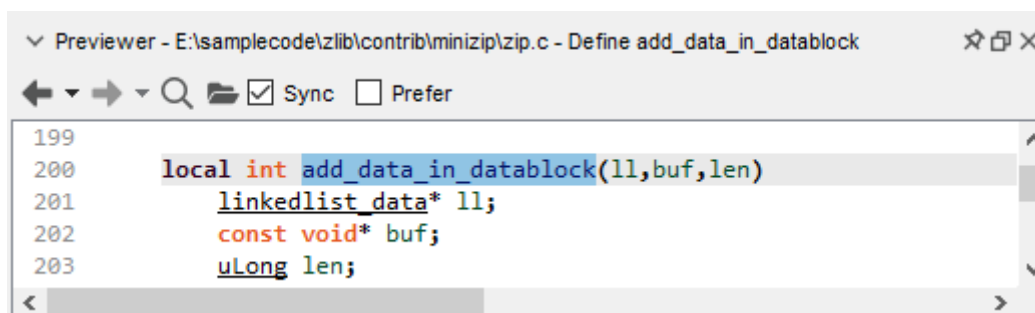
The Source Editor also provides several other options for displaying and editing files:

- *Previewer* on [page 195](#)
- *Bracket Matching* on [page 195](#)
- *Folding and Hiding* on [page 196](#)
- *Changing the Source Code Font Size* on [page 196](#)
- *Splitting the Editor Window* on [page 197](#)
- *Changing Case* on [page 196](#)
- *Commenting and Uncommenting* on [page 196](#)
- *Indentation* on [page 197](#)
- *Line Wrapping* on [page 198](#)
- *Insert and Overtyping Modes* on [page 198](#)
- *Sorting Lines Alphabetically* on [page 198](#)
- *Keyboard Commands* on [page 198](#)
- *Recording, Playing, and Saving Macros* on [page 198](#)
- *Creating and Opening Files* on [page 199](#)
- *Bookmarking* on [page 199](#)
- *Managing Source Editor Tabs* on [page 201](#)
- *Spell Checking* on [page 202](#)

Previewer

The Previewer window is similar to a Source Editor window. To open the Previewer window, highlight a filename and 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 to 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 121](#)).

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

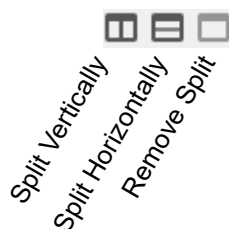
Commenting and Uncommenting

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.

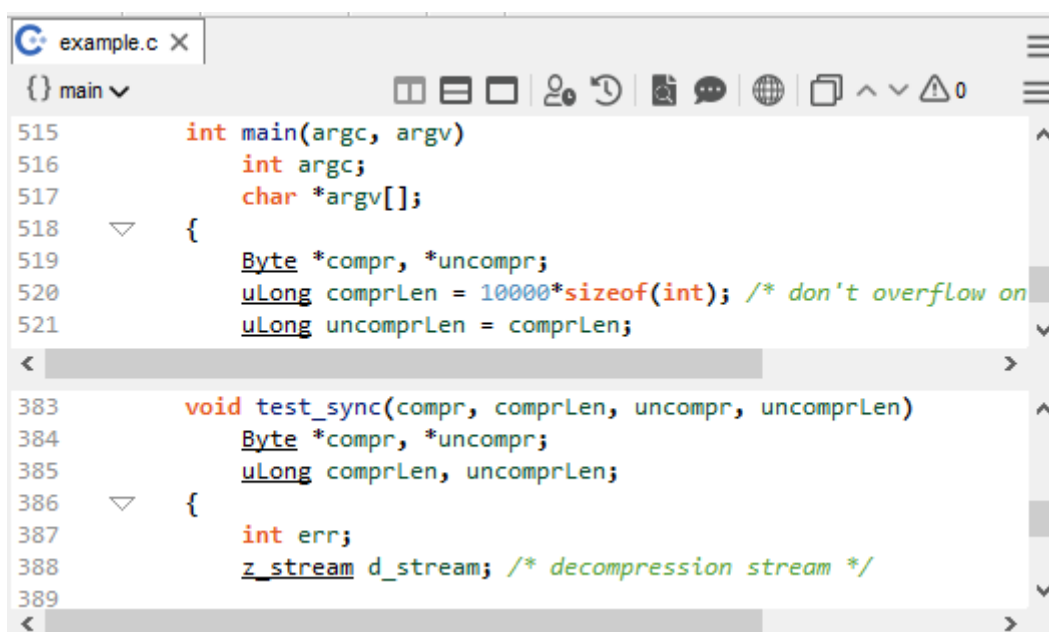
Comments are not analyzed when a project is analyzed.

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:

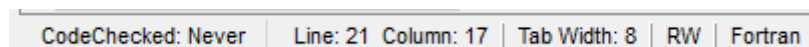


Click the **Split Vertically** or **Split Horizontally** icon to divide the source editor into two or more separately scrollable panes. You cannot split a Source Editor window in both directions, but you can split it multiple times in the same direction. Click the **Remove Split** icon to remove a split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



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 121](#)).



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 123](#)).

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 123](#) to change the wrap mode for source code printing.

Insert and Overtyping Modes

Normally, text to the right of your typing cursor is shifted as you type. This is called Insert mode. To switch between Insert mode and Overtyping mode, in which text to the right of the cursor is replaced character-by-character as you type, press the **Insert** key or choose **Edit > Toggle Overtyping** from the menus.

Sorting Lines Alphabetically

To sort a group of lines into alphabetical order, select the lines, right-click and choose **Sort Selection**.

Keyboard Commands

To see a list of keystrokes that work in the Source Editor, choose **Tools > Options** and go to the **Key Bindings > 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.

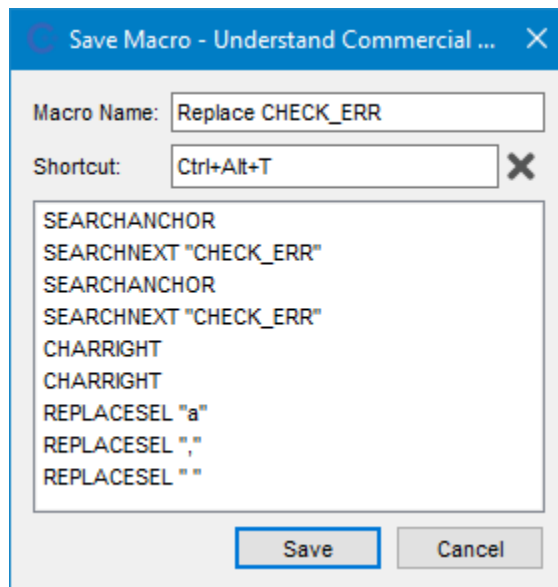
Recording, Playing, and Saving Macros

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 127](#) 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.

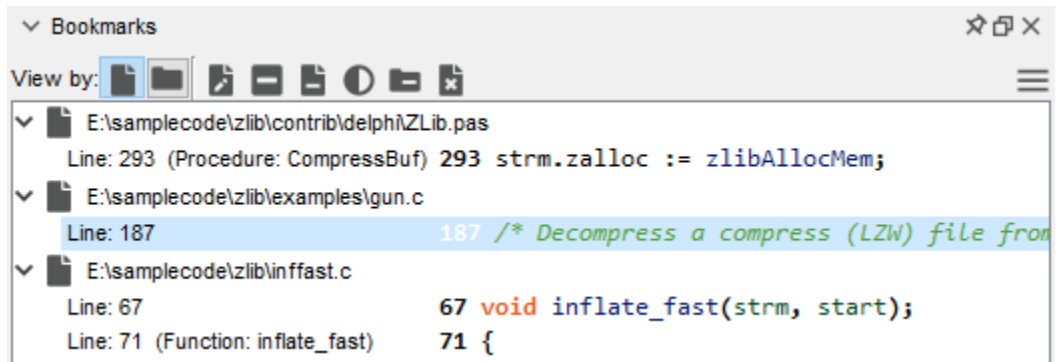
Bookmarking

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 155) are stored locally for the user. If you want to share code locations with others who use the same project, see *Annotations on page 203*.

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

View by: 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 and the category itself.

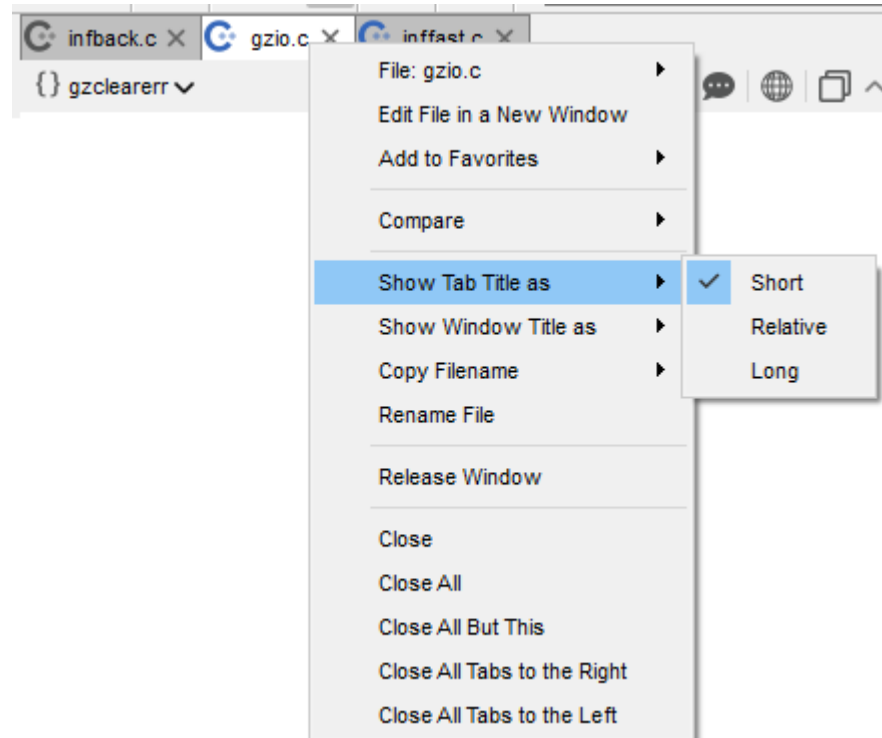
Click this icon to delete all your bookmarks.

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.

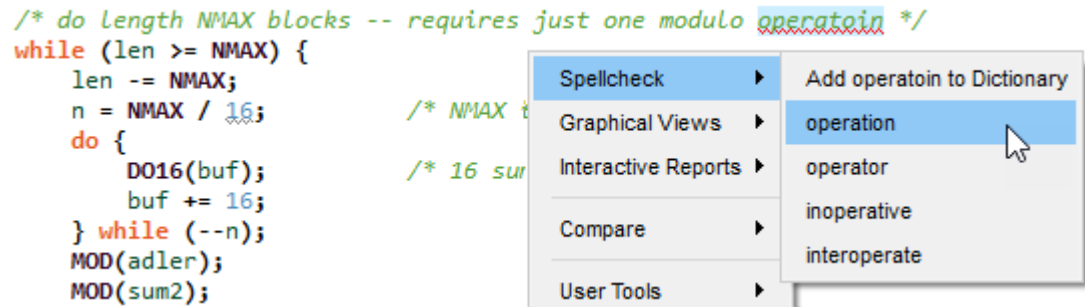
If you choose **Release Window**, the tabbed area changes to a separate window that can be moved around your screen. Click the  icon in the upper-right corner of a released window and choose **Capture Window** to return the window to a tab within the *Understand* window.

Spell Checking

By default (in new installations) *Understand* checks the spelling of both comments and quoted strings in your code automatically.

Any words that are not found in the dictionary and are not the names of entities in your analyzed project have a red, squiggly underline. The underline may take a little while to show up after you misspell a word, but you do not need to save a file or analyze a file in order for spelling to be checked.

To correct a misspelling, right-click on the underlined word and select the correct word from the list in the **Spellcheck** menu. Or you can add the word to the dictionary.



You can disable spell checking in comments or quoted strings by unchecking the **Enable Spellchecking in Comments** and/or **Enable Spellchecking in Strings** box in the **Editor > Advanced** category of the **Tools > Options** dialog (see [page 123](#)).

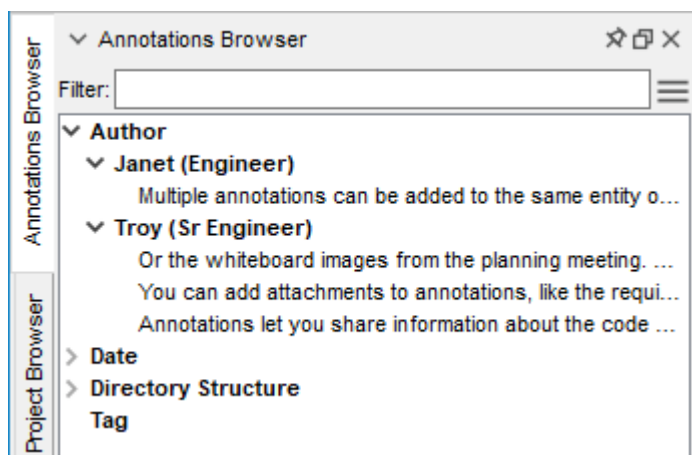
Currently, *Understand* provides spell checking using an English dictionary. If you would like to spell check using another language, please contact us at support@scitools.com.

Annotations

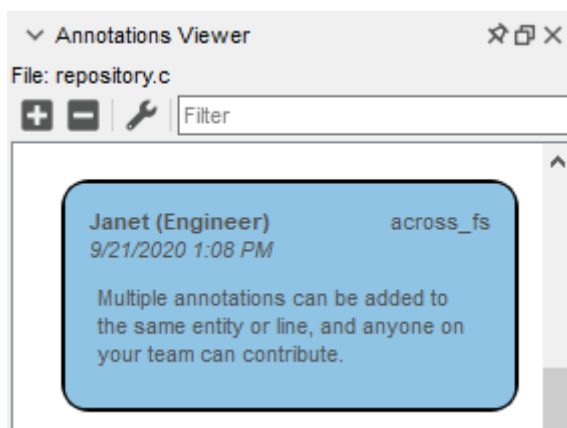
Annotations are an easy and convenient way for Understand users to add documentation without making changes to the source code directly, yet still have the comment where you can see it quickly, right in context with the code. Annotations can apply to lines of code, entities, or files. You can attach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

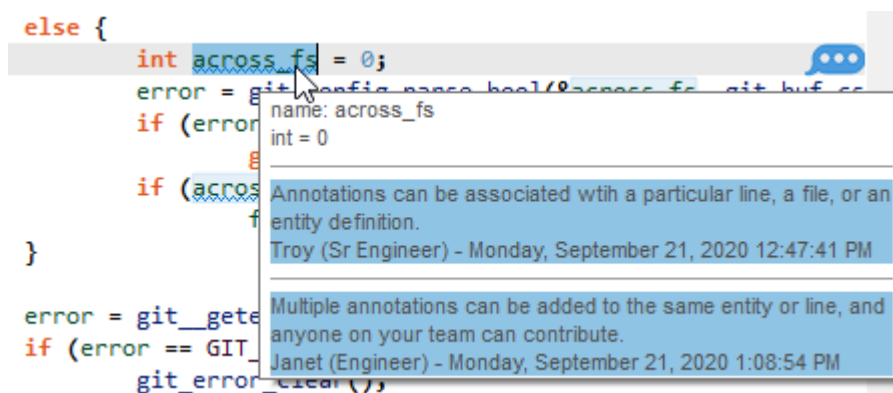
Annotations Browser



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 132](#)) 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 35](#) 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.

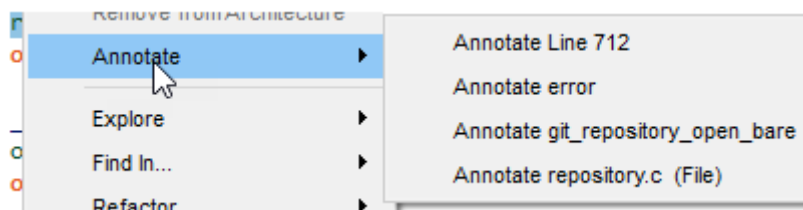
You can also search for annotations in the Annotations Viewer by typing in its **Filter** field. Note that the Annotations Viewer shows only annotations in the currently viewed source file, while the Annotations Browser shows all annotations in the project.

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

Adding Annotations

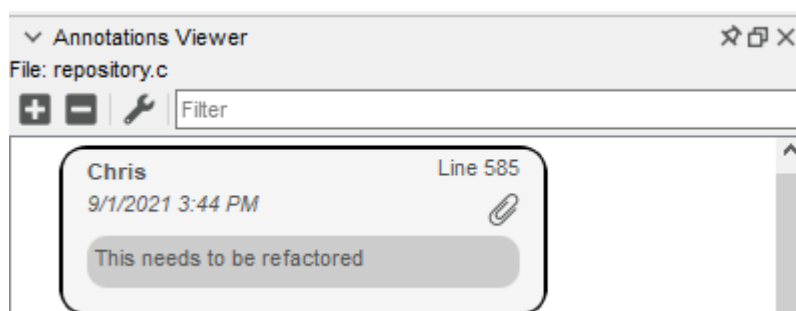
You can add an annotation in the following ways:

- 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.

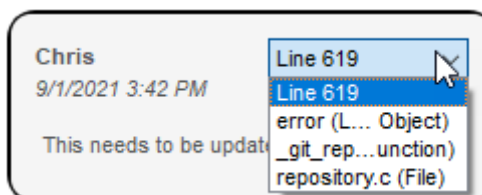
Whenever you create an annotation, the Annotations Viewer opens and a new annotation is added. Type text for your annotation, which is saved automatically when you click outside the annotation.



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 207](#)).

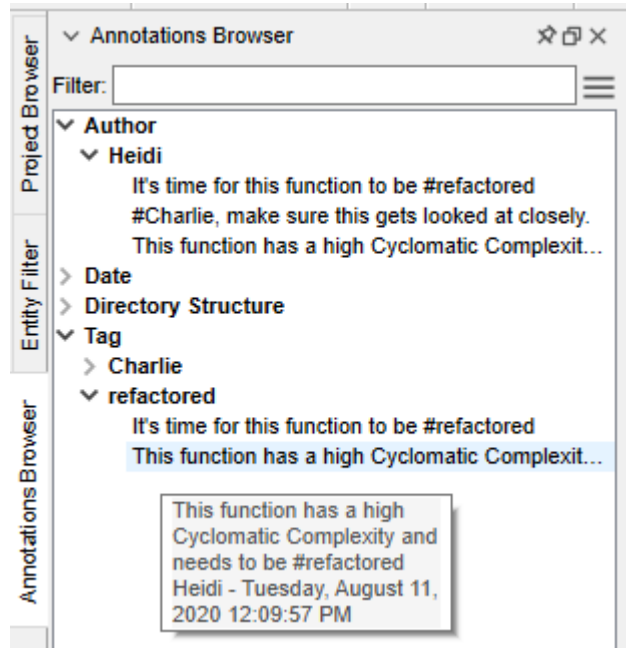
You can change the context for annotations you created. Click on the line or entity in the upper-right corner of an annotation and select the new context.




Tagging Annotations

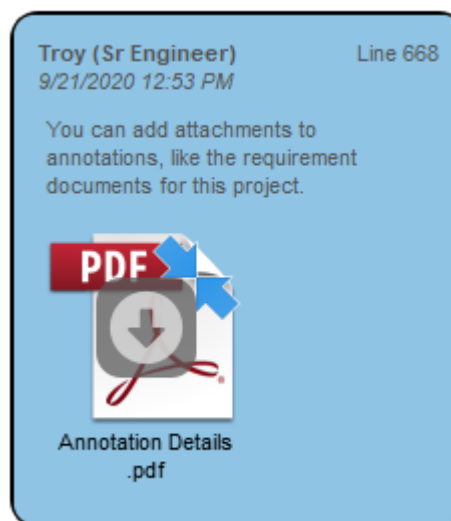
Annotations let you add tags that provide an easy way to create to-do lists, prioritize work, and assign tasks. To add a tag, simply type # and a keyword (as you would in social media applications). For example, you might use #Project7 or #Refactor.

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 204](#) for details on how these files are stored and shared.



Deleting Annotations

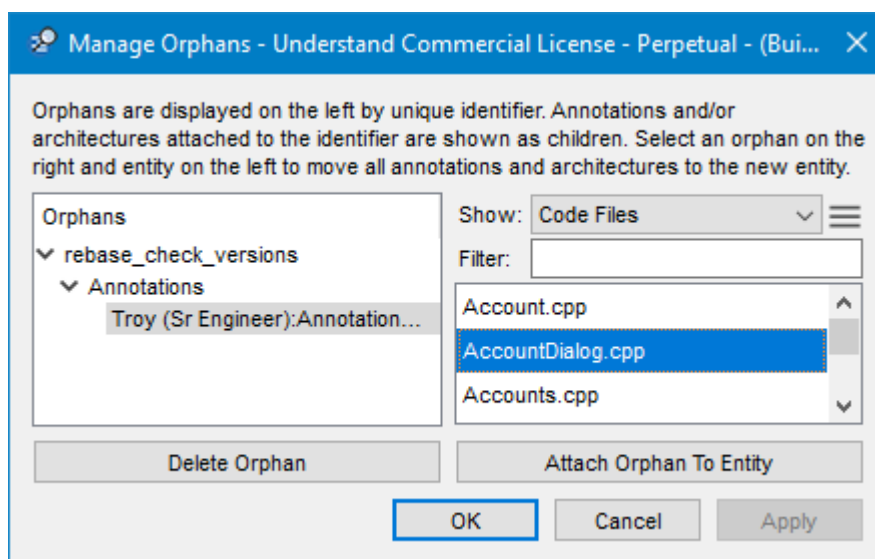
To delete an annotation, select it in the Annotations Viewer and click the  minus icon.

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**. (This command is only shown if orphaned annotations exist in the current project.)
- 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.

Using Annotations to Ignore Violations

You can use keywords in annotations to automatically ignore CodeCheck violations the same way you can with keywords in code comments. See *Adding Automatic Ignores to Code* on page 309 for details.

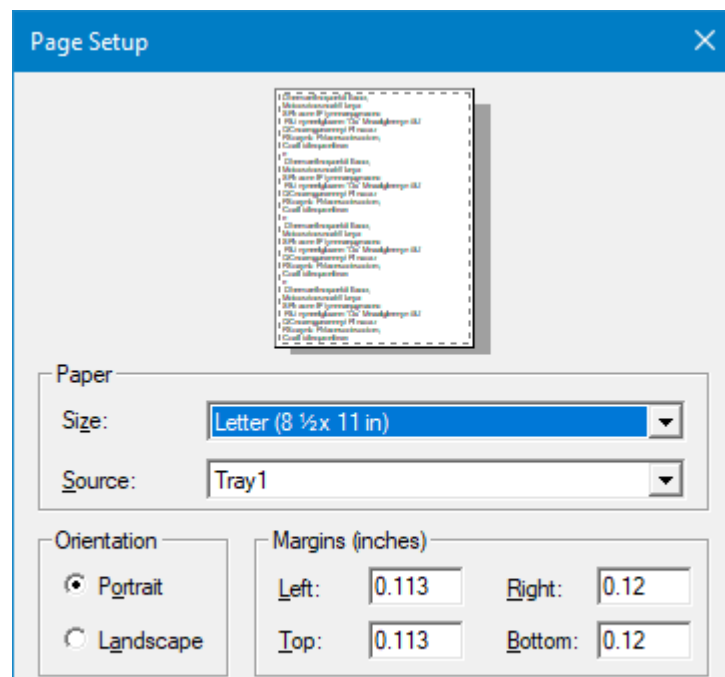
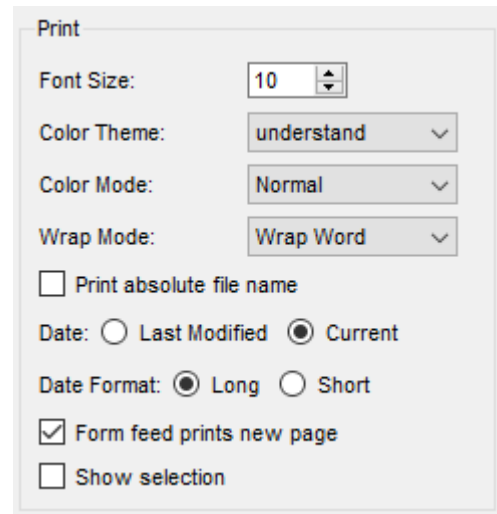
Printing Source Views

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 123](#) 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 203](#).

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:

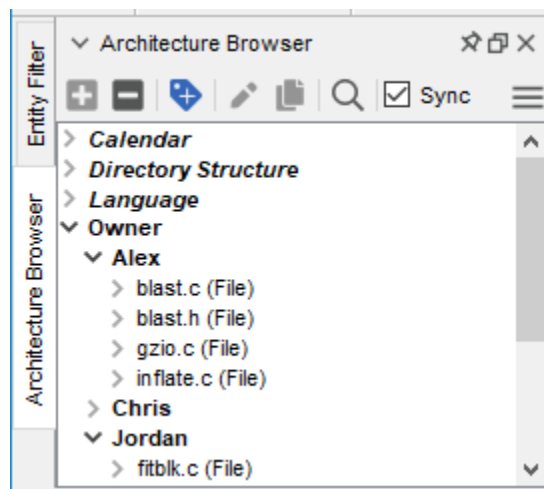
Section	Page
About Architectures	210
Browsing Architectures	211
Creating and Editing Custom Architectures	214
Sharing Architectures	219
Using Automatic Architectures	219
Viewing Architecture Graphs	220
Checking Dependencies	222
Viewing Architecture Metrics	223

About Architectures

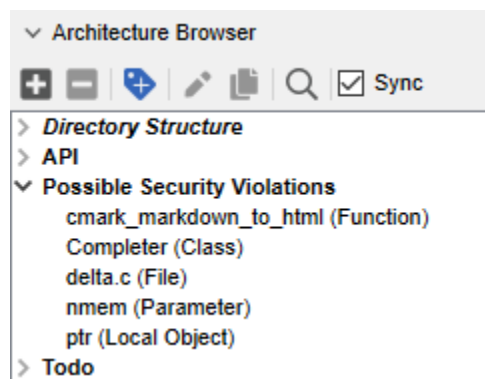
An *architecture* in *Understand* is an abstract hierarchy layered onto a body of source code. An architecture creates a hierarchy of source code units (entities). Architectures allow you to name and organize parts of a software project or ways of looking at software hierarchically.

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.



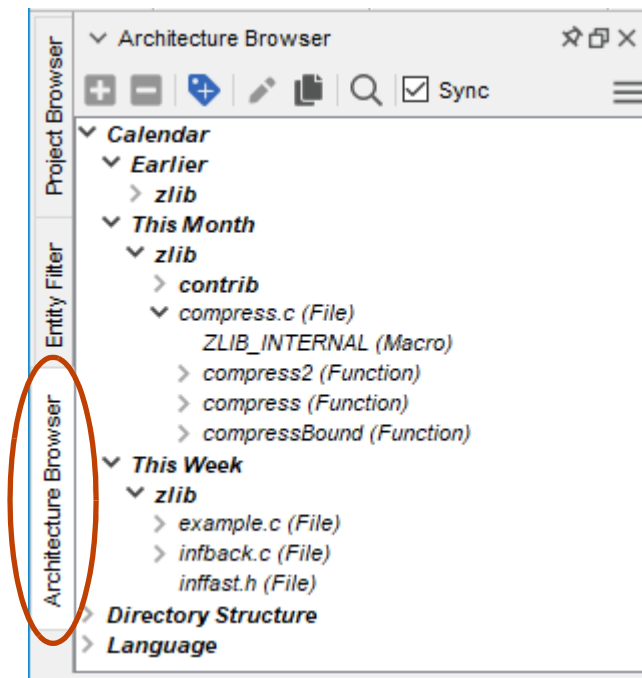
Architectures are saved as part of a project and so can be used collaboratively with others who work with the same project.

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.

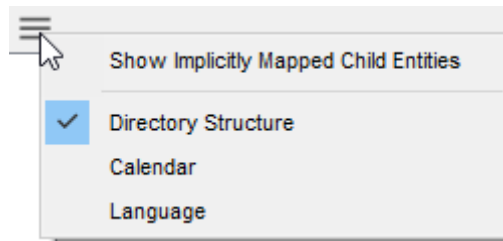
You can choose whether to display long or short names for files and entities by clicking the  hamburger menu icon and changing the settings for **Show File Name As** and **Show Entity Name As**.

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 163](#)) to search within architectures. See [page 214](#) 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

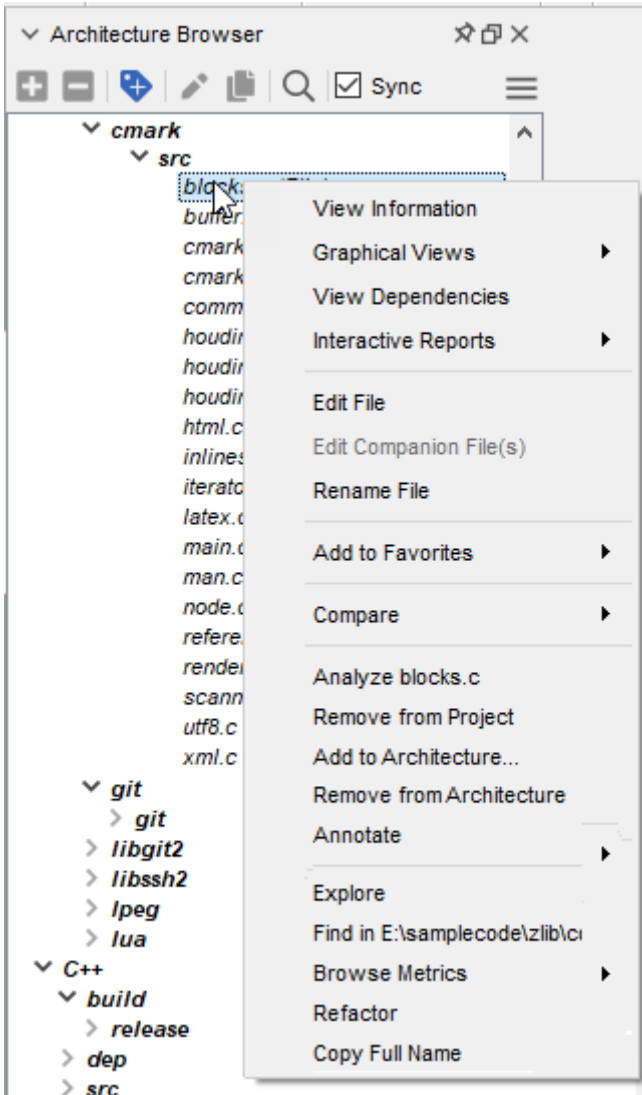
The context menu for an architecture node (such as a filesystem directory or “This Quarter”) contains some extra items not available in other context menus:

- **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 221](#).
- **Graphical Views > Internal Dependencies:** Shows the dependencies between architecture nodes. See [page 220](#).

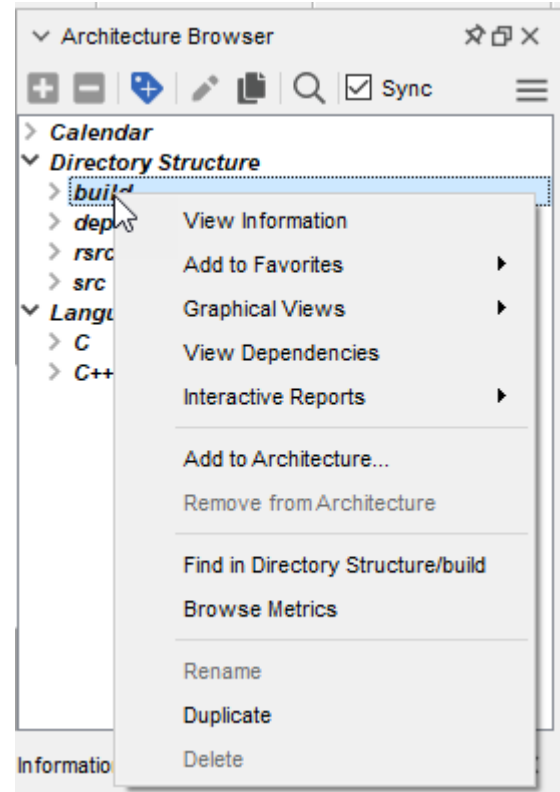
- **Add to Architecture:** Adds the contents of a selected node, file, or entity to another architecture node. You cannot edit the auto-architectures provided with *Understand*. See [page 215](#).
- **Remove from Architecture:** Removes the selected node or file from the architecture or node you select. You cannot remove files from the auto-architectures provided with *Understand*. See [page 215](#).
- **Browse Metrics:** Shows metrics for entities within the selected node (but not including entities in sub-nodes lower in the hierarchy. See [page 248](#).
- **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 215](#).
- **Duplicate:** Opens a Duplicate Architecture window that lets you type a name for a duplicate copy of the selected architecture or architecture node. See [page 214](#).
- **Delete:** Delete the selected architecture node. See [page 215](#).

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 217](#).


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 215](#).
-  **Delete Architecture Node:** Custom architectures only. See [page 215](#).
-  **Tag Entities:** See [page 215](#).
-  **Edit architecture:** Rename the selected architecture or node. Custom architectures only. See [page 215](#).
-  **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 at the same level as the architecture or node of which it is a copy. 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.

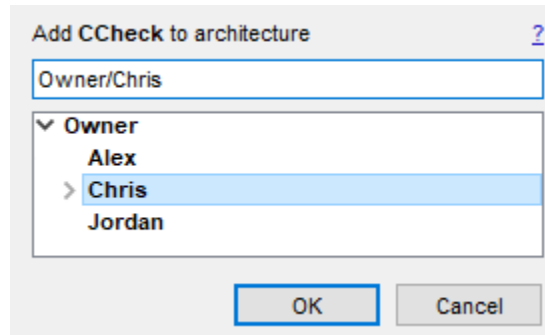
Adding Files and Entities to Nodes

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.

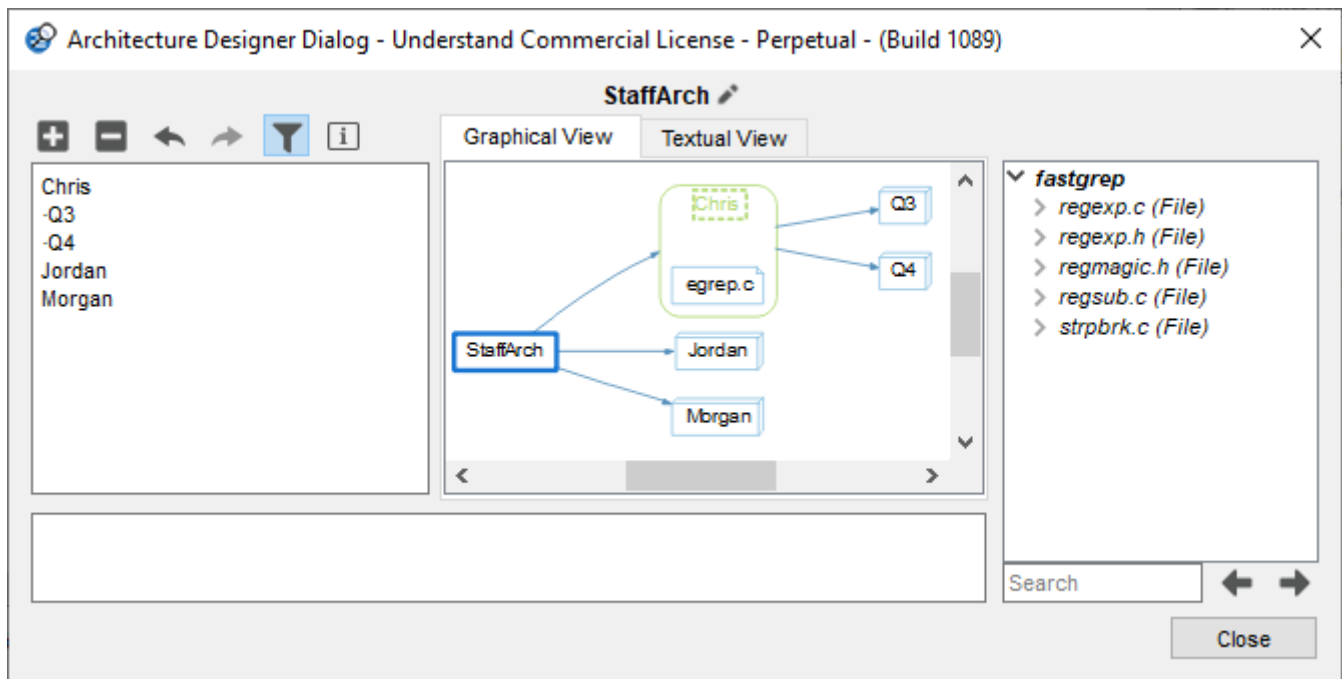
Managing Orphan Architecture Nodes



Architecture nodes can become “orphans” if their parent node is deleted. To delete or attach orphaned nodes, choose **Architectures > Manage Orphans** from the menus. (This command is only shown if orphaned architecture nodes exist in the current project.) See [page 207](#) 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 or right-clicking and choosing **Expand Architecture** or **Collapse Architecture**. Duplicate a node by selecting a node, right-clicking, choosing **Copy Architecture Node for Paste**, selecting the location for the duplicated node, right-clicking, and choosing **Paste Architecture Node for previous Copy**.
- **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 You can delete a selected node in the Graphical View tab by pressing the Delete key. If that node has children, you are asked if you also want to delete the children. If you choose **No**, the children are attached to the node above the node you deleted.
- 6 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.
- 7 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.

Using Automatic Architectures


Automatic architecture plugins are Python scripts that run every time you open an *Understand* project. They allow you to easily create your own architectures that are automatically updated. For example, an architecture can query your version control system to identify the most often modified files. Or, an architecture can separate third-party libraries from your own code. Or, it can work with your file naming conventions to create a custom structure.

Example automatic architectures are provided in the `plugins/Arch` directory of your *Understand* installation and the Arch subdirectory of the plugins Git repository at <https://github.com/stinb/plugins>. For example, the Visual Studio Structured Solution architecture can be used with projects with code managed by Visual Studio.

To install an Architecture plugin, drag its `.upy` file into the Understand GUI, which will ask if you want to install the plugin. Alternatively, you can manually place it in one of the following locations:

- **Windows:** `C:\Program Files\SciTools\conf\plugin\User\Arch` or `C:\Users\<user>\AppData\Roaming\SciTools\plugin\Arch`
- **Linux:** `/home/<user>/.config/SciTools/plugin/Arch`
- **MacOS:** `/Users/<user>/Library/Application Support/SciTools/plugin/Arch`

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category* on [page 109](#)).

After installing an architecture plugin, choose **Tools > Clear Script Cache** from the menus or restart *Understand*. Then, select the name of the architecture plugin in the  hamburger menu of the Architecture Browser.

Some plugins import common files provided in the `plugins/Solutions` directory of your *Understand* installation. For example, Git-related architectures use the `git` solution and visibility matrix architectures use the `visibilityMatrix` solution. To allow a plugin to use a solution, copy the whole directory to the same location relative to the plugin. For example, on Windows, copy the `plugins/Solutions/visibilityMatrix` directory to `conf/plugin/Solutions/visibilityMatrix` so that the plugin can find the common tool.

Note: A project does not finish opening until any architecture scripts finish running. During that time, `/visibilityMatrix` is essentially frozen. Your custom scripts should finish quickly to avoid a long wait when opening a project.

Viewing Architecture Graphs

You can generate graphs that show the hierarchy of an architecture. See [Chapter 11, 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 258](#) 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. Additional graph categories may be available for certain architecture structures and nodes. For example, for architectures with nodes created for functions where the functions share global variables, the Shared Objects in Functions graph is available.

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 263](#) for details about using the icons in the graphical view toolbar.

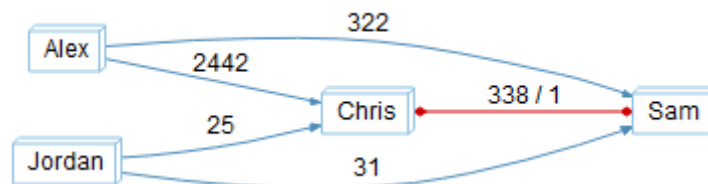
Architecture graphs can be quite large. Remember that you can use Ctrl+F to search within a graph.

You can save these graphs as PNG, JPEG, SVG, Visio XML, DOT, and PU files. See [page 285](#).

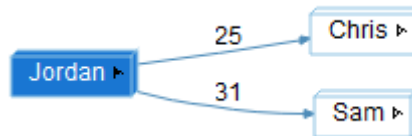
You can also use the Dependency Browser to see dependencies within an architecture. See [page 148](#).

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.



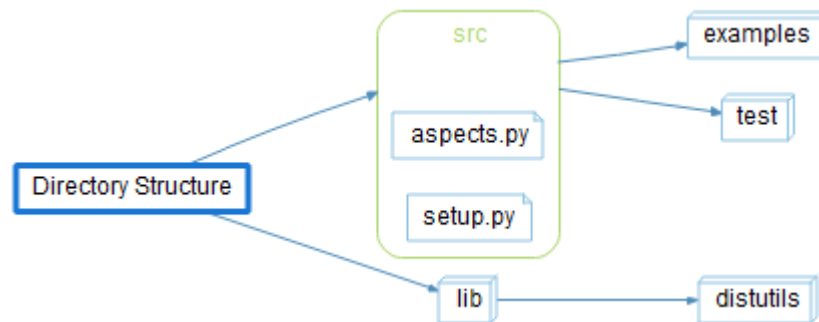
Use the **Variant** drop-down list to change the graph to show **Internal** dependencies within the selected node only, nodes **Depended On By** this node, or nodes that this node **Depends On**.

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 261](#)).

Architecture Structure Views

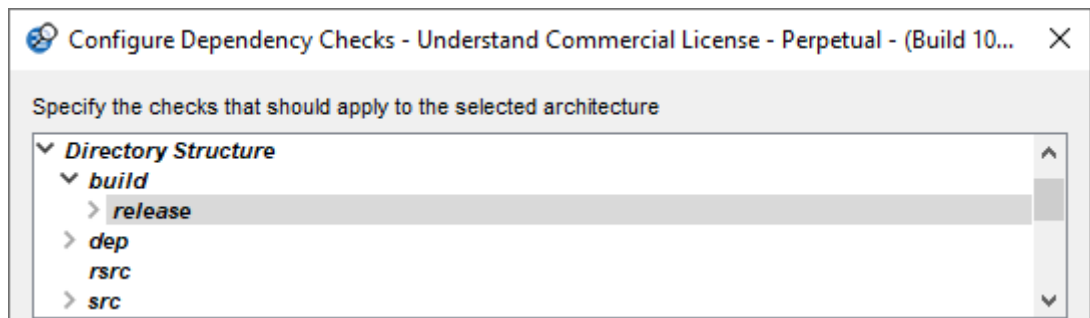
The **Graph Architecture** view is different from the Architecture Dependency graphs. It shows the structure of the architecture, rather than dependencies between entities in the architecture. Open the Graph Architecture view the same way you would open other graphical views (see [page 263](#)).



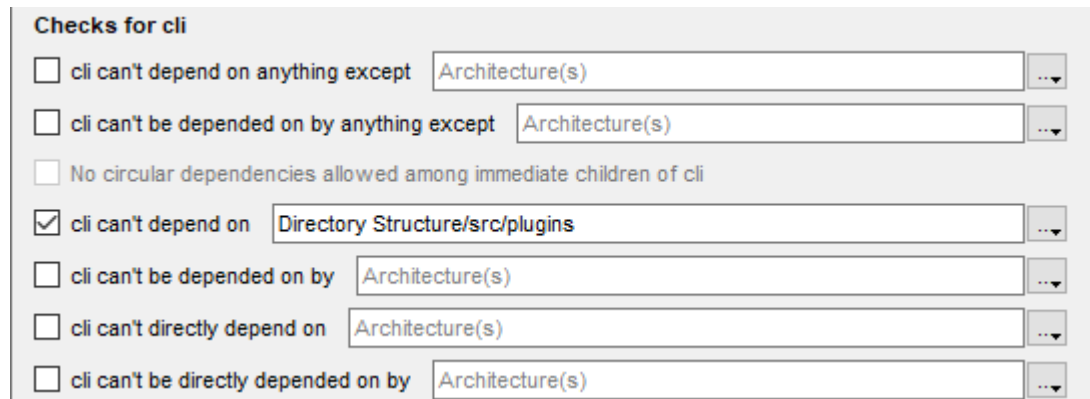
Checking Dependencies

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”.

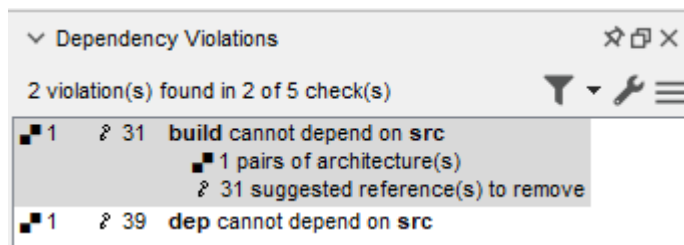
Configured Checks	
Architecture	Check
Directory Structure/src/cli	Cannot depend on Directory Structure/src/plugins
Directory Structure/src/plugins	Cannot be depended on by Directory Structure/src/cli

- 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. Any exceptions are included in the descriptions.



- 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 295](#).

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 10, Using Metrics](#).

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

Note: Access to HTML Reports is controlled by your *Understand* license. Please contact our support department if you would like to enable this feature.

This chapter contains the following sections:

Section	Page
Configuring Reports	225
Generating Reports	226
Viewing Reports	227
An Overview of Report Categories	228
Cross-Reference Reports	230
Structure Reports	234
Quality Reports	237
Metrics Reports	241
Importing Report Plugins	244

Configuring Reports

Understand provides a large number of reports you can generate about your code. These can be generated in HTML or text format. You can choose which reports and how to format them.

To configure how reports will be generated, choose **Reports > Configure Reports**. This opens the Project Configuration dialog with the **Reports > Output** category selected. From there, you can also configure the **Reports > Options** and **Reports > Selected** categories.

See *Report Options* on [page 56](#) for details on the **Reports > Output** category. In general, you can configure the following:

- **HTML reports:** The “home” file for the reports is index.html, but you can select an alternate title page. You may generate single or multiple HTML files for each report type. It is recommended that you split up the files for large projects. Choose *Alphabetic* to generate multiple HTML files per report that are split up alphabetically by the first letter of the entity name. Choose *Every n Entities* to generate multiple HTML files per report that are split up every “n” number of entities. By default, a single HTML file is generated for each letter of the alphabet.
- **Text reports:** You may generate one text file of the specified name (by choosing File). This one file will contain all the selected reports. Alternately, you may generate multiple text files (by choosing Separate Files) and specify a common filename prefix. The filenames of each text file identify the report.

The **Reports > Selected** category lets you select from the available reports for the languages used by your project. The specific reports available depend upon the source languages used in your project. This list shows the reports for all languages:

See *An Overview of Report Categories* on [page 228](#) for descriptions of the types of reports you can generate.

The screenshot shows a dialog box with a list of reports. All items in the list are checked with a checkbox. At the bottom, there are two buttons: 'All' and 'None'. The 'All' button is highlighted.

<input checked="" type="checkbox"/>	Data Dictionary
<input checked="" type="checkbox"/>	File Contents
<input checked="" type="checkbox"/>	Program Unit Cross Reference
<input checked="" type="checkbox"/>	Object Cross Reference
<input checked="" type="checkbox"/>	Type Cross Reference
<input checked="" type="checkbox"/>	Macro Cross Reference
<input checked="" type="checkbox"/>	Include File Cross Reference
<input checked="" type="checkbox"/>	Declaration Tree
<input checked="" type="checkbox"/>	Extend Tree
<input checked="" type="checkbox"/>	Invocation Tree
<input checked="" type="checkbox"/>	Simple Invocation Tree
<input checked="" type="checkbox"/>	Import
<input checked="" type="checkbox"/>	With Tree
<input checked="" type="checkbox"/>	Simple With Tree
<input checked="" type="checkbox"/>	Generic Instantiation
<input checked="" type="checkbox"/>	Exception Cross Reference
<input checked="" type="checkbox"/>	Renames
<input checked="" type="checkbox"/>	Program Unit Complexity
<input checked="" type="checkbox"/>	Project Metrics
<input checked="" type="checkbox"/>	Program Unit Metrics
<input checked="" type="checkbox"/>	File Metrics
<input checked="" type="checkbox"/>	File Average Metrics
<input checked="" type="checkbox"/>	Fortran Extension Usage
<input checked="" type="checkbox"/>	Class Metrics
<input checked="" type="checkbox"/>	Class OO Metrics
<input checked="" type="checkbox"/>	Implicitly Declared Objects
<input checked="" type="checkbox"/>	Uninitialized Items
<input checked="" type="checkbox"/>	Unused Variables and Parameters
<input checked="" type="checkbox"/>	Unused Objects
<input checked="" type="checkbox"/>	Unused Types
<input checked="" type="checkbox"/>	Unused Program Units
<input checked="" type="checkbox"/>	Uses Not Needed
<input checked="" type="checkbox"/>	Withs Not Needed

All None

Customizing Report Colors

HTML reports use Cascading Style Sheets (CSS) to set colors and font styles used for keywords, comments, strings, numbers, and more. The colors and styles used are defined in the `sourcestyles.css` file, which is created the first time you generate HTML reports in a particular location.

You can customize the `sourcestyles.css` file using a text editor. Any colors and font styles normally supported by CSS can be used in this file. For example:

```
span.comment{color:DarkSeaGreen;font-style:italic}
```

If you modify the stylesheet and want to use it for other reports you generate, you can copy the modified `sourcestyles.css` file to the locations of other HTML reports.

Generating Reports

Once you have specified formatting options and the types of reports to be generated, choose **Reports > Generate Reports** from the menus to open a dialog that lets you begin generating the selected reports. Click **Generate** to show the progress of the report generation.

On Windows, the ASCII text follows the DOS text file format (carriage return and line feed at the end of each line). On Unix, text files are created according to the Unix convention (lines end with a carriage return).

HTML reports are generated as HTML 3.0 format files. The generated HTML is not complex, the only HTML 3.0 (versus HTML 2.0) feature used is frames. Netscape 2.0 and higher, and Internet Explorer 3.0 and higher can display the files.

You can view the reports as described in *Viewing Reports on page 227*.

For large projects, reports can take a long time to generate. You can click **Cancel** to halt report generation. Clicking **Cancel** leaves the reports in a partially generated state.

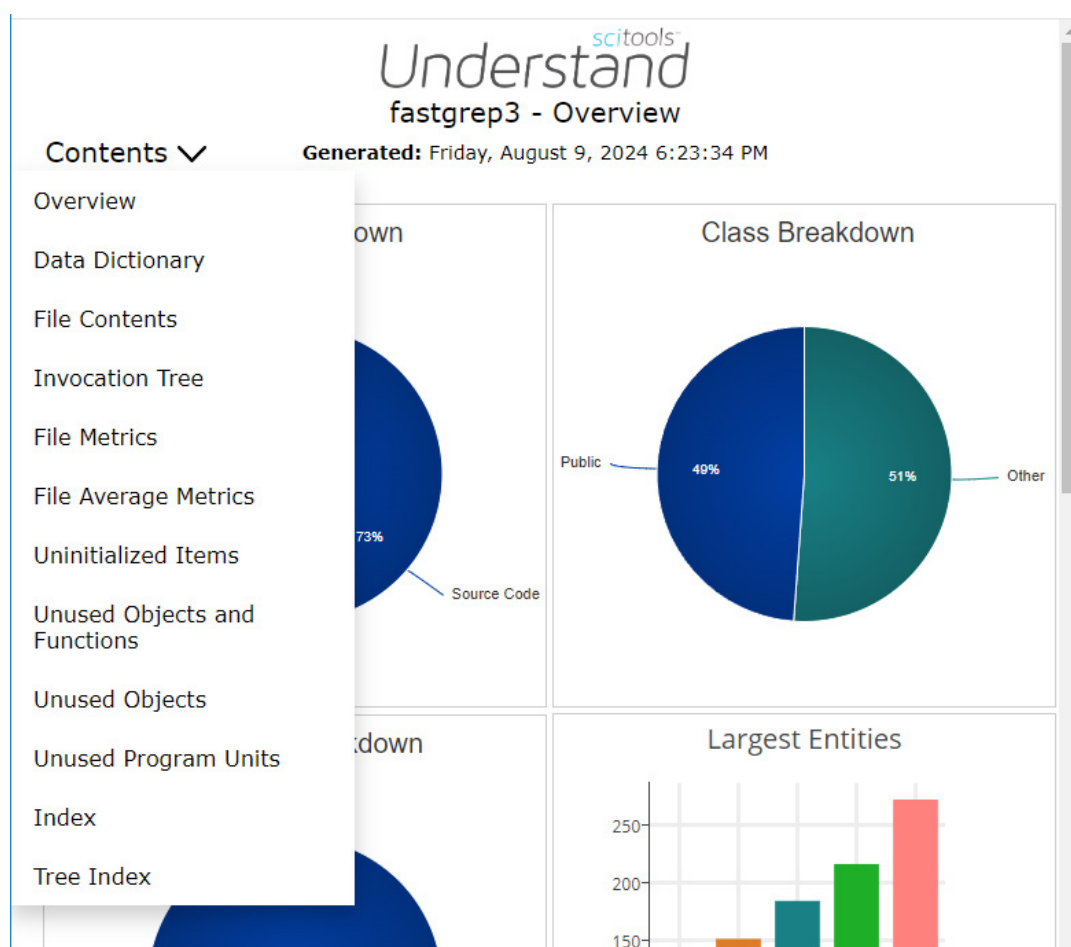
Note: You may want to temporarily toggle off anti-virus protection programs while reports are being generated. This may speed the process of creating reports. If you do this, be sure to turn on virus checking after report generation is finished.

Note: HTML, text, and project metrics reports may also be generated with the “und” command line program. Refer to [Chapter 15, Command Line Processing](#) for details.

Viewing Reports

To view generated reports, choose **Reports > View Reports**. Then choose the **HTML** or **Text** option. File names of reports generated vary based on the type and format of the report generated.

- For text files, a single text file containing all selected reports may be generated or separate files for each type of report may be generated. A single text file is named `<project_name>.txt`. For separate text files, the file name is the type of report.
- For HTML reports, you can generate either a single HTML file for each report type or smaller files divided either alphabetically by entity name or in groups of N number of entities. The main window page is named `index.html` and provides project summary graphs, such as code line type breakdown, largest entities and files, most complex entities and files, and comment ratio.



For HTML reports, the Index contains an alphabetic list of all entities found in all other generated HTML reports. The index links to the Data Dictionary report for each entity, which in turn links to the entity's declaration source code and the object cross reference report. The entity index can be accessed from the "index" link on the main HTML page.

An Overview of Report Categories

Understand generates a wide variety of reports. The reports fall into these categories:

- **Cross-Reference** reports show information similar to that in the *Information Browser*, except that all entities are shown together in alphabetic order. See *Cross-Reference Reports* on [page 230](#).
- **Structure** reports show the structure of the analyzed program. See *Structure Reports* on [page 234](#).
- **Quality** reports show areas where code might need to be examined. See *Quality Reports* on [page 237](#).
- **Metrics** reports show basic metrics such as the number of lines of code and comments. See *Metrics Reports* on [page 241](#).

The following table shows the type and page number for each report. Note that the specific reports available depend upon the source languages used in your project.

Report Type	Report Name and Page
Cross-Reference	<i>Data Dictionary Report</i> on page 230
Cross-Reference	<i>File Contents Report</i> on page 231
Cross-Reference	<i>Program Unit Cross-Reference Report</i> on page 231
Cross-Reference	<i>Object Cross-Reference Report</i> on page 232
Cross-Reference	<i>Type Cross-Reference Report</i> on page 232
Cross-Reference	<i>Macro Cross-Reference</i> on page 233
Cross-Reference	<i>Include File Cross-Reference</i> on page 233
Cross-Reference	<i>Exception Cross-Reference Report</i> on page 233
Structure	<i>Declaration Tree</i> on page 234
Structure	<i>Extend Tree</i> on page 235
Structure	<i>Invocation Tree Report</i> on page 235
Structure	<i>Simple Invocation Tree Report</i> on page 235
Structure	<i>Imports Report</i> on page 236
Structure	<i>With Tree Report</i> on page 236
Structure	<i>Simple With Tree Report</i> on page 236
Structure	<i>Generic Instantiation Report</i> on page 236
Structure	<i>Renames Report</i> on page 236
Quality	<i>Program Unit Complexity Report</i> on page 237
Quality	<i>Uninitialized Items</i> on page 238
Quality	<i>Unused Objects and Functions</i> on page 238
Quality	<i>Unused Objects Report</i> on page 239
Quality	<i>Unused Types Report</i> on page 239
Quality	<i>Unused Program Units Report</i> on page 240
Quality	<i>Uses Not Needed Report</i> on page 240
Quality	<i>Withs Not Needed Report</i> on page 240

Report Type	Report Name and Page
Quality	<i>Implicitly Declared Objects Report</i> on page 238
Quality	<i>Fortran Extension Usage Report</i> on page 238
Metrics	<i>Project Metrics Report</i> on page 241
Metrics	<i>Program Unit Metrics Report</i> on page 243
Metrics	<i>File Metrics Report</i> on page 243
Metrics	<i>File Average Metrics Report</i> on page 244
Metrics	<i>Class Metrics Report</i> on page 242
Metrics	<i>Class OO Metrics Report</i> on page 242

Augment with Plugins

The reports included with *Understand* have evolved over many years to accommodate common customer requests. However, we recognize that not all needs can be covered.

To help you develop custom reports we include both PERL and C interfaces to *Understand* databases.

For details on the PERL interface choose **Help > PERL API Documentation**. Also visit the support page on our website. Java API documentation is provided in the `doc/manuals/java` subdirectory of the *Understand* installation.

The **Reports > Project Interactive Reports** and **Graphs > Project Graphs** commands display a list of user-created plugins, which can be created using the Perl or Python API. See *Importing Report Plugins* on [page 244](#) for more information.

See the `plugins/IReport` directory in your *Understand* installation or the IReport folder in the Git repository at <https://github.com/stinb/plugins> for example report plugins and a Readme file. See the *Understand* blog for more information. The SciTools Support page at scitools.com/support contains information about plugins in the “Plugins/API” category.

Cross-Reference Reports

Cross-Reference reports show information similar to that in the References section of the Information Browser, except that all entities are shown together in alphabetic order. The following table shows the page that describes each type of cross-reference report.

Report Name
Data Dictionary Report on page 230
Program Unit Cross-Reference Report on page 231
File Contents Report on page 231
Object Cross-Reference Report on page 232
Type Cross-Reference Report on page 232
Macro Cross-Reference on page 233
Include File Cross-Reference on page 233
Exception Cross-Reference Report on page 233

Data Dictionary Report

The *Data Dictionary Report* lists all entities alphabetically. Each listing shows the entity name, what kind of entity it is (for example, macro, type, variable, function, include, file, or procedure), along with links to the location where each is declared in the source code.

gitahead-Windows - Data Dictionary

Contents ▾ & A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

year (Local Object)[\[xref\]](#)
[\[date.c, 38\]](#)

yFinalStop (Unresolved Parameter)[\[xref\]](#)

yil (Local Object)[\[xref\]](#)
[\[CommitList.cpp, 969\]](#)

Name

What kind of entity it is

What file/line it was declared in.

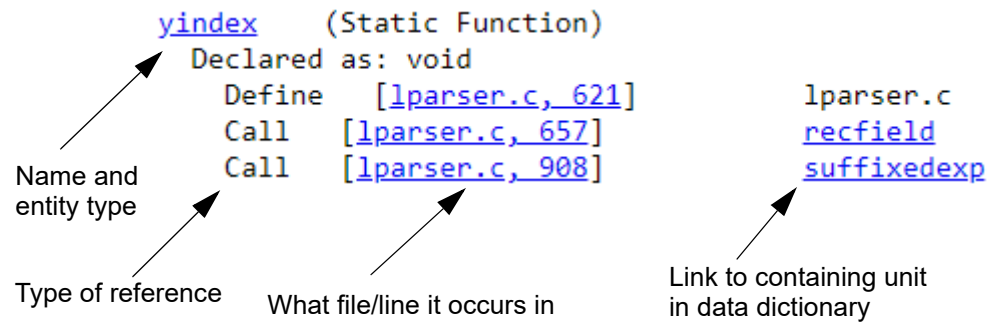
Quick link to cross-reference of this entity

Optionally break up report alphabetically

Program Unit Cross-Reference Report

The *Program Unit Cross-Reference Report* lists all program units (such as procedures and functions) analyzed in alphabetic order along with information about what they return (if anything), what parameters are used, and where they are used by other program units.

The HTML version offers hyperlinks to the Data Dictionary report entry and to the source code where each reference occurs.



You can create an additional Program Unit Index report to list all the program units in the project and show the file and line where each is declared. For text reports, this is stored in a *.pcn file.

File Contents Report

Lists functions declared within a source file and the line numbers where they are declared. HTML versions of this report permit hyperlinked jumping to the function in the source as well as viewing of the entire source file.



Object Cross-Reference Report

The Object Cross-Reference Report lists all objects (Fortran *variables*, *parameters*, *macros*) in alphabetic order along with declaration and usage references.

```

new_base      (Local Object)  Declared as: git_annotated_commit *=((void *)0)
  Define      [merge.c, 2265]      compute_base
  Init        [merge.c, 2265]      compute_base
  Addr Use    [merge.c, 2303]      compute_base
  Use         [merge.c, 2310]      compute_base
  Set         [merge.c, 2311]      compute_base
  Use         [merge.c, 2322]      compute_base

new_blob      (Parameter)    Declared as: const git_blob *
  Define      [patch_generate.c, 588] git_diff_blobs
  Use         [patch_generate.c, 600] git_diff_blobs

```

The HTML version of this report includes hyperlinks to the Data Dictionary Report and the source code where the reference occurs. This report was previously titled the Class and Interface Cross-Reference Report.

Type Cross-Reference Report

The Type Cross-Reference Report lists all declared types in alphabetic order, along with their declaration and usage information. The HTML version of the report offers hyperlinks to the Types data dictionary report entry, as well as the source code where the reference occurs.

```

X509_PUBKEY   (Typedef)
  Declared as: struct X509_pubkey_st
  Type        [openssl.c, 1477]      pubkey

xdalgoenv_t    (Typedef)
  Declared as: struct s_xdalgoenv
  Type        [xdiffi.c, 335]         xenv
  Type        [xdiffi.c, 277]         xenv
  Type        [xdiffi.c, 53]          xenv
  Type        [xdiffi.c, 72]          xenv

```


Macro Cross-Reference

The Macro Cross-Reference Report lists all macros analyzed in the source code in alphabetic order along with information about where they are declared and where they are used. The HTML version offers hyperlinks to the macro's Data Dictionary report entry and to the source code where each reference occurs.

[WAIT_TIMEOUT](#)

Declared as: 258L

Use [\[relauncher.cpp, 35\]](#) [main](#)

[WAS_NEWLINE](#)

Declared as: ((NLBLOCK->nlttype != NLTYPE_FIXED)? ((p) > NLBLOCK->PSSTART &

Use [\[pcre_dfa_exec.c, 871\]](#) [internal_dfa_exec](#)

Use [\[pcre_dfa_exec.c, 3508\]](#) [pcre_dfa_exec](#)

Use [\[pcre_exec.c, 2093\]](#) [match](#)

Use [\[pcre_exec.c, 6812\]](#) [pcre_exec](#)

Include File Cross-Reference

The Include File Cross-Reference Report lists all include files analyzed in the source code in alphabetic order with information about which files include them. The HTML version offers hyperlinks to the source code where each reference occurs.

[version.h](#)

Include [\[path.c, 16\]](#) [path.c](#)

[VisualStyle.h](#)

Include [\[EditModel.cxx, 40\]](#) [EditModel.cxx](#)

Include [\[MarginView.cxx, 41\]](#) [MarginView.cxx](#)

Include [\[EditView.cxx, 47\]](#) [EditView.cxx](#)

Include [\[VisualStyle.cxx, 27\]](#) [VisualStyle.cxx](#)

Include [\[Editor.cxx, 46\]](#) [Editor.cxx](#)

Exception Cross-Reference Report

The *Exception Cross-Reference Report* documents the declaration and usage of all exceptions. Each declaration and any *raises* or *handles* are documented. In the HTML version each raise or handle may be visited in the source, as well as the declaration point of the Exception (if visible).

adaCoreVSS - Exception Cross Reference

Contents ▾ [&](#) [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#)
[N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

[VSS.Magic_Error](#) (Exception)

Declare [\[vss.ads, 28\]](#) [VSS](#)

Structure Reports

Structure reports are designed to help you understand the relationships between various entities. The following table shows the page in this chapter that describes each type of structure report.

Report Name and Page
<i>Declaration Tree</i> on page 234
<i>Extend Tree</i> on page 235
<i>Invocation Tree Report</i> on page 235
<i>Simple Invocation Tree Report</i> on page 235
<i>With Tree Report</i> on page 236
<i>Simple With Tree Report</i> on page 236
<i>Generic Instantiation Report</i> on page 236
<i>Renames Report</i> on page 236
<i>Imports Report</i> on page 236

Declaration Tree

The *Declaration Tree* shows the declaration nesting of each program unit analyzed. Each nesting level is indicated by an indent with a vertical bar used to help align your eyes when viewing. Each nesting level is read as “declares”. In the HTML version of the report each program unit name is a hyperlink to its entry in the *Program Unit Cross-Reference Report*.

```
Package Body Occupants
| Procedure Get
| Function May_I_Get
| Procedure Drop
| Function May_I_Drop
| Procedure Inventory
| Procedure Go
| | Block
```

In the above example, *Package Body Occupants* is the top level program unit. It has declared within it, *Put_View*, *Look*, *Get*, *May_I_Get*, *Drop*, *May_I_Drop*, *Inventory*, and *Go*. Nested within *Go* is an unnamed declare block.

The Declaration Tree report shows a representation of the declaration tree in each file.

```
Misc Package
| MedianOfRunningArray Public Class
| | MedianOfRunningArray Public Constructor
| | insert Public Method
| | median Public Method
| | main Public Static Method
| PalindromePrime Public Class
| | main Public Static Method
| | prime Public Static Method
| | reverse Public Static Method
```

Extend Tree

The *Extend Tree* report shows the nesting of class declarations in the files analyzed. Each nesting level is indicated by an indent with a vertical bar to help align your eyes when viewing. Each nesting level is read as “extends”. In the HTML version of the report each class name is a hyperlink to its entry in the *Data Dictionary and Interface Cross-Reference Report*.

Invocation Tree Report

The Invocation Tree Report shows a textual representation of the invocation tree for each program unit analyzed. The report shows what each program unit calls. Levels are indicated by tabs and are lined up with vertical bars. Each nesting level is read as “calls”. The HTML version has hyperlinks to the corresponding Data Dictionary report entries.

```

p_getcwd
|  _wgetcwd
|  git_utf16_to_8
|  *** Repeated Subtree ***
|  GetLastError
|  _errno

pack
|  lua_gettop
|  lua_createtable
|  *** Repeated Subtree ***
|  lua_rotate
|  *** Repeated Subtree ***
|  lua_seti
|  |  index2addr
|  |  luaH_getint
|  |  luaC_barrierback_
|  |  luaV_finishset
|  |  *** Repeated Subtree ***
|  lua_pushinteger

```

Simple Invocation Tree Report

The Simple Invocation Tree Report shows the invocation tree to only one level for each program unit that has been analyzed. The invocation level is indicated by an indent and a vertical bar and is read as “calls”.

```

zcalloc
|  malloc
|  calloc

zcfree
|  free

```

With Tree Report

Structured identically to the other hierarchy reports, the *With Tree* report shows a textual version of the With Tree for each program unit that is not Withed by another.

As with the other textual hierarchy reports, indents show level with a vertical bar helping align your eye. For this report, each line is read as “Withs”.

```
Package Body Occupants
|  Package Rename Text_IO
|  |  Package Text_IO
|  |  |  Package IO_Exceptions
|  |  |  Package System
|  |  |  Package Parameters
```

In the above example, the package body *Occupants* Withs package *Text_IO*, which in turn Withs *IO_Exceptions*, *System*, and *Parameters*.

Simple With Tree Report

The *Simple With Tree* report is similar to the *With Tree* report. It shows a textual representation of the With Tree for each program unit that is not Withed by another. However, it shows only one level of withs. For example:

```
Package Body Occupants
|  Package Rename Text_IO
```

Generic Instantiation Report

This report lists each package that was created through instantiation.

In the HTML version, the source where it was instantiated and its Data Dictionary Report entry may be visited from hyperlinks.

```
My_Int_IO  Package Instantiation
          FILE: board.adb  LINE:12
Instantiated From =>  INTEGER_IO  Generic Package
```

Renames Report

The *Renames Report* cross-references the use of the Ada command “renames”, as in:

```
function Rouge return Color renames Red;
```

This report lists program units that have been renamed in alphabetic order. Each rename shows the program unit it renames, and in the HTML report a hyperlink to the rename instance in the source is provided.

The Information Browser also identifies packages and program units that rename others or are renamed.

Imports Report

The *Imports* report lists all source files that import other files and the files they import. The HTML version offers hyperlinks to the data dictionary entry for each imported file.

Quality Reports

Understand's quality reports are designed to provide information about areas of the analyzed source that might not meet standards or that hold the potential for trouble. They also identify areas where extra programming has been done but not needed. This sometimes identifies areas that aren't yet complete, or that haven't been maintained completely.

The following table shows the page in this chapter that describes each type of quality report.

Report Name and Page
<i>Program Unit Complexity Report</i> on page 237
<i>Fortran Extension Usage Report</i> on page 238
<i>Implicitly Declared Objects Report</i> on page 238
<i>Uninitialized Items</i> on page 238
<i>Unused Objects and Functions</i> on page 238
<i>Unused Objects Report</i> on page 239
<i>Unused Types Report</i> on page 239
<i>Unused Program Units Report</i> on page 240
<i>Uses Not Needed Report</i> on page 240
<i>Withs Not Needed Report</i> on page 240

The complete list of quality metrics available in *Understand* changes frequently - more frequently than this manual is reprinted. A complete and accurate list is always available on our website: scitools.com/support/metrics_list/.

Program Unit Complexity Report

The *Program Unit Complexity Report* lists every procedure and function or similar program unit in alphabetic order along with the McCabe (Cyclomatic) complexity value for the code implementing that program unit.

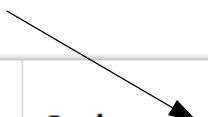
The Cyclomatic complexity is the number of independent paths through a module. The higher this metric the more likely a program unit is to be difficult to test and maintain without error.

The Modified column shows the cyclomatic complexity except that each case statement is not counted; the entire switch counts as 1.

The Strict column shows the cyclomatic complexity except && and || also count as 1.

The Nesting column shows the maximum nesting level of control constructs in this program unit.

Click column header for explanation of each metric



Entity	Cyclomatic	Modified	Strict	Essential	Nesting	Path Count	Path Count Log
joinktables	4	4	6	1	1	4	1
jumponcond	3	3	3	3	2	3	0
jumptohere	1	1	1	1	0	1	0
jumptothere	2	2	2	1	1	2	0

Fortran Extension Usage Report

This report lists anywhere your source code has non-standard Fortran extensions. The report factors in what variant (F77, F90, F95) you chose on your project configuration.

Implicitly Declared Objects Report

The *Implicitly Declared Objects Report* lists any variables or parameters that were implicitly declared using Fortran's implicit declaration mode. Using implicitly declared variables is considered a risky practice, and this report helps you weed out where the practice is occurring in your code.

The HTML version offers hyperlinks to the function's Data Dictionary report entry.

Uninitialized Items

The *Uninitialized Items* report lists items such as variables that are not initialized in the code. The report is organized by file. Each uninitialized item within the file is listed by name along with the line number on which the item is declared. The HTML version offers hyperlinks to the location where the item is declared.

Unused Objects and Functions

The *Unused Objects and Functions* report lists items that are declared (and perhaps initialized) but never referenced other than that. The report is organized by file. Each unused item is listed by name along with the type of item and the line number on which the item is declared. The function or similar container is shown after the list of unused items within it. Types of items may include functions, parameters, variables, and objects. The HTML version offers hyperlinks to the location where each unused item is declared.

Unused Objects Report

The *Unused Objects Report* lists objects (for example, variables, parameters, constants) that are declared but never used. The HTML version links to the function's Data Dictionary report entry and to the source line where the object is declared.

gitahead-Windows - Unused Objects		
Contents	▼ & A B C D E F G H I J K L M	
	N O P Q R S T U V W X Y Z	
xdiffi.c		
rchg1		297
rchg2		291
xhistogram.c		
histindex::key_shift		67
histindex::records_size		63

Unused Types Report

The *Unused Types Report* lists types that are declared but never used. The HTML version has links to the function's Data Dictionary report entry and the source where the type is declared.

TabWidget.cpp		
DefaultWidget::QPrivateSignal		34
ThemeDialog.cpp		
ThemeButton::QPrivateSignal		27

Unused Program Units Report

The *Unused Program Units Report* identifies program units that are declared but never used.

Note that this listing in this report doesn't mean the system doesn't need this program unit. For instance, interrupt handlers that are called by system interrupts are often never "used" within the other source of the program.

gitahead-Windows - Unused Program Units		
Contents ▾	& A B C D E F G H I J K L M	
	N O P Q R S T U V W X Y Z	
zutil.c		133
zError		32
zlibCompileFlags		27
zlibVersion		

Uses Not Needed Report

The *Uses Not Needed Report* identifies any unneeded "use" statements that provide access to a module's public specifications and definitions. To remove unneeded access, you may add only clauses to use statements.

Withs Not Needed Report

This report lists, any With statements a program unit has but does not need (by not using items made public by the With statement).

Note that this covers only direct usage in the program unit and doesn't account for side effects that may be needed by the program to operate correctly. For instance, sometimes a package can be Withed just to start a task or to execute code in its begin/end block.

Metrics Reports

Metrics provide statistical information about your project and entities, such as the number of lines of code and the complexity of various entities.

Understand provides a number of ways to gather metrics information. This section describes reports that provide metrics. See [page 246](#) for other ways to gather metrics.

The following table shows the page that describes each type of metrics report.

Report Name and Page
<i>Project Metrics Report</i> on page 241
<i>Class Metrics Report</i> on page 242
<i>Class OO Metrics Report</i> on page 242
<i>Program Unit Metrics Report</i> on page 243
<i>File Metrics Report</i> on page 243
<i>File Average Metrics Report</i> on page 244

The complete list of metrics available in *Understand* changes frequently—more frequently than this manual is reprinted. A complete and accurate list is always available on our website: scitools.com/support/metrics_list/.

Project Metrics Report

The *Project Metrics Report* provides metric information about the entire project. The metrics reported are:

- Total number of classes (if applicable)
- Total number of files
- Total number of program units
- Total number of source lines
- Total number of blank lines
- Total number of comment lines
- Total number of inactive lines
- Total number of executable statements
- Total number of declarative statements
- Ratio of comment lines to code lines

Some of these metrics are also reported on the title page of the HTML report.

Classes:	129
Files:	479
Program Units:	9894
Lines:	302627
Lines Blank:	41638
Lines Code:	195473
Lines Comment:	31200
Lines Inactive:	31866
Executable Statements:	129450
Declarative Statements:	33814
Ratio Comment/Code:	0.16

.....

Class Metrics Report The *Class Metrics Report* provides the following metrics for each class that has been analyzed:

- Total number of lines
- Total number of blank lines
- Total number of lines of code
- Total number of lines that contain comments
- Average number of lines per class
- Average number of comment lines per class
- Average complexity per class
- Maximum complexity within class
- Ratio of comment lines to code lines

Class	Lines	Lines Blank	Lines Code	Lines Comment	Average Lines	Average Lines Comment	Average Complexity	Maximum Complexity	Ratio Comment/Code
WildcardQuery	17	2	15	0	6	0	1	1	0.00%
WindowPanel	205	36	158	13	451	26	3	8	0.08%

.....

Class OO Metrics Report The *Class OO Metrics Report* provides the following object-oriented metrics for each class that has been analyzed:

- **LCOM (Percent Lack of Cohesion):** 100% minus the average cohesion for class data members. A method is cohesive when it performs a single task.
- **DIT (Max Inheritance Tree):** Maximum depth of the class in the inheritance tree.
- **IFANIN (Count of Base Classes):** Number of immediate base classes.
- **CBO (Count of Coupled Classes):** Number of other classes coupled to this class.
- **NOC (Count of Derived Classes):** Number of immediate subclasses this class has.
- **RFC (Count of All Methods):** Number of methods this class has, including inherited methods.
- **NIM (Count of Instance Methods):** Number of instance methods this class has.
- **NIV (Count of Instance Variables):** Number of instance variables this class has.
- **WMC (Count of Methods):** Number of local methods this class has.

Program Unit Metrics Report

The *Program Unit Metrics Report* provides information on various metrics for each program unit that has been analyzed.

	Lines	Comments	Blanks	Code	Lines-exe	Lines-dec	Stmt-exe	Stmt-dec	Ratio Comment/Code
Query::parseQuery	13	2	1	10	6	3	6	2	0.20
querycap	11	4	0	11	6	2	6	1	0.36
queue_difference	22	0	2	20	12	3	8	3	0.00
queue_objects	95	6	19	71	49	10	46	10	0.08
quotes_for_value	17	0	4	13	8	2	9	1	0.00

The following metrics are provided for each program unit:

- **Lines:** Total number of lines in the function.
- **Comment:** Number of comment lines in the function.
- **Blank:** Number of blank lines in the function.
- **Code:** Number of lines in the function that contain any code.
- **Lines-exe:** Lines of code in the function that contain no declaration.
- **Lines-decl:** Lines of code in the function that contain a declaration or part of a declaration.
- **Stmt-exe:** Number of executable statements in the function.
- **Stmt-decl:** Number of declarative statements in the function. This includes statements that declare classes, structs, unions, typedefs, and enums.
- **Ratio Comment/Code:** Ratio of comment lines to code lines.
(comment_lines/code_lines)

Note: code+comment+blank != lines because some lines contain both code and comments.

File Metrics Report

The *File Metrics Report* provides information similar to that in the *Program Unit Metrics Report*. However, it is organized by file rather than by *program unit*.

Click on each metric column to get a detailed description of it.

Note: code+comment+blank != lines because some lines contain both code and comments.

File	Lines	Comments	Blanks	Code	Lines-exe	Lines-dec	Stmt-exe	Stmt-dec	Ratio Comment/Code	Units
zstream.c	210	14	44	148	98	19	89	23	0.09	13
zutil.c	325	21	20	68	41	18	53	7	0.31	5

File Average Metrics Report

The *File Average Metrics Report* provides averages for the functions within a file. All lines outside any function are ignored when calculating the averages. The following metrics are provided for each function:

- **Cyclomatic:** The average number of independent paths through the functions in this file. The higher this metric the more likely a program unit is to be difficult to test and maintain without error.
- **Modified:** Same as Cyclomatic complexity except that each case statement is not counted; the entire switch statement counts as 1.
- **Strict:** Same as Cyclomatic complexity except that && and || also count as 1.
- **Essential:** Measures the amount of unstructured code in a function.
- **Lines:** Average number of lines in the functions in this file.
- **Code:** Average number of lines that contain any code in the functions in this file.
- **Comment:** Average number of comment lines in the functions in this file.
- **Blank:** Average number of blank lines in the functions in this file.

Importing Report Plugins

Plugins are special Perl or Python scripts that can be imported to produce customized reports. Several report plugins are available in the `plugins/IReport` directory in your *Understand* installation or the IReport folder in the Git repository at <https://github.com/stinb/plugins>.

See the *Understand* blog for more information. The SciTools Support page at scitools.com/support contains information about plugins in the “Plugins/API” category.

To install a report plugin, drag the .upy or .upl file into the Understand GUI, which will ask if you want to install it. Alternatively, you can place it in the following location:

- **Windows:** `C:\Program Files\SciTools\conf\plugin\User\IReport` or `C:\Users\<user>\AppData\Roaming\SciTools\plugin\IReport`
- **Linux:** `/home/<user>/.config/SciTools/plugin/IReport`
- **MacOS:** `/Users/<user>/Library/Application Support/SciTools/plugin/IReport`

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category* on [page 109](#)).

Then choose **Tools > Clear Script Cache** from the menus or restart *Understand*. Use the **Reports** menu to open your plugin report.


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

This chapter contains the following sections:


Section	Page
About Metrics	246
Home	247
Metrics Browser	248
Metric Definitions	249
Exporting Metrics	250
Metrics in the Information Browser	252
Metrics Treemap	253
Metric Plugins	256

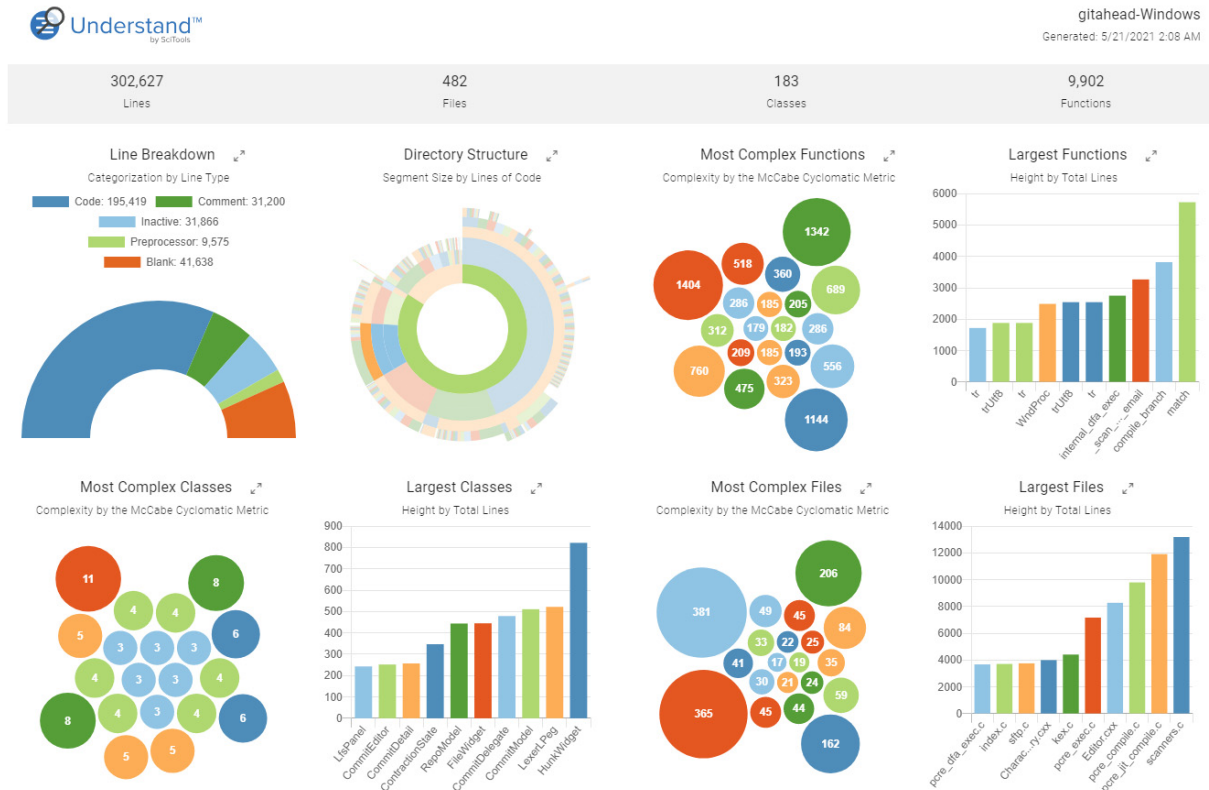
About Metrics

Understand provides a number of ways to gather metrics information:

- **Home:** Choose **Project > Home** to see graphs of metrics for the overall project, including the most complex and largest functions, classes, and files. See [page 247](#).
- **Configure Available Metrics:** *Understand* provides a large number of metrics. To shorten the lists of metrics (which can improve performance), see [page 63](#).
- **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 248](#).
- **Metric Definitions:** 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. See [page 249](#).
- **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 143](#).
- **Export to HTML:** You can use the Metrics Browser to export the full list of metrics for all architecture nodes and files. See [page 250](#).
- **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 250](#).
- **Python and Perl APIs:** 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 343](#).

Home


Choose **Project > Home** from the menus or click the  Home icon in the main toolbar to see overview graphs of metrics for the project, including the most complex and largest functions, files, classes (for languages that support classes), and packages (for languages that support packages). Graphs of the project by file dependencies, line type, and directory structure are also shown. Hover your cursor over a colored area to see what it represents.



If you analyze a project, including using the recommended CodeCheck checks, statistics about accuracy and errors found are added to the top of the overview. In addition, graphs of errors and warnings (grouped by type) and violations per line of code are added to the overview.

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

Grades are shown for project metrics such as Function Complexity and Technical Debt (violations per line of code). Click the links to find ways to improve your project.

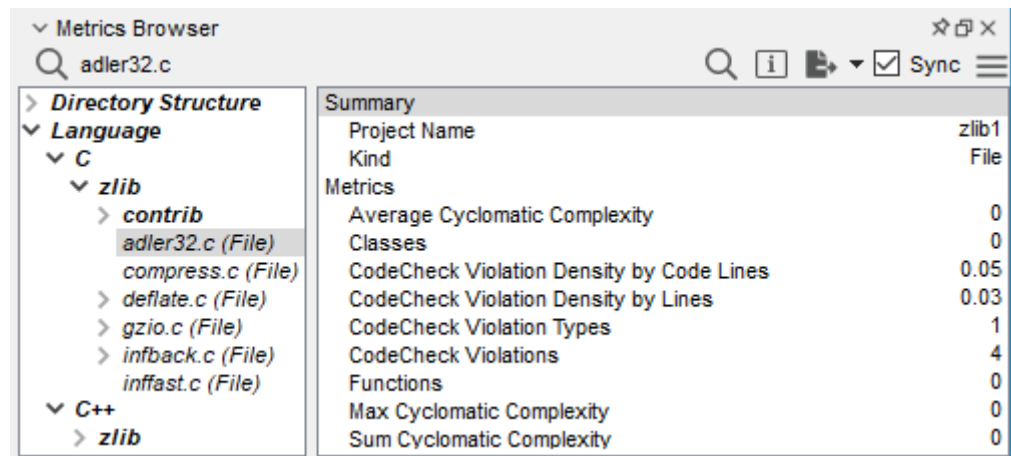
You can copy a graph from the Home tab to your clipboard by clicking the  icon near that graph.

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 architectures and entities in your project and select any architecture node, file, or entity. By default, the list on the right shows code size, complexity, and CodeCheck violation 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 249](#) for details.

Click the **Export Metrics** icon in the Metrics Browser toolbar to save metrics to a file. See [page 250](#) 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 control the metrics available here using the *Metrics Options* on [page 63](#). You can further adjust the display by selecting options from the hamburger menu:

- **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 “RatioCommentToCode”.

- **Show Friendly Name:** Displays the metrics using descriptive names, such as “Comment to Code Ratio”. The list is sorted differently depending on whether you are viewing it by API Name or Friendly Name.

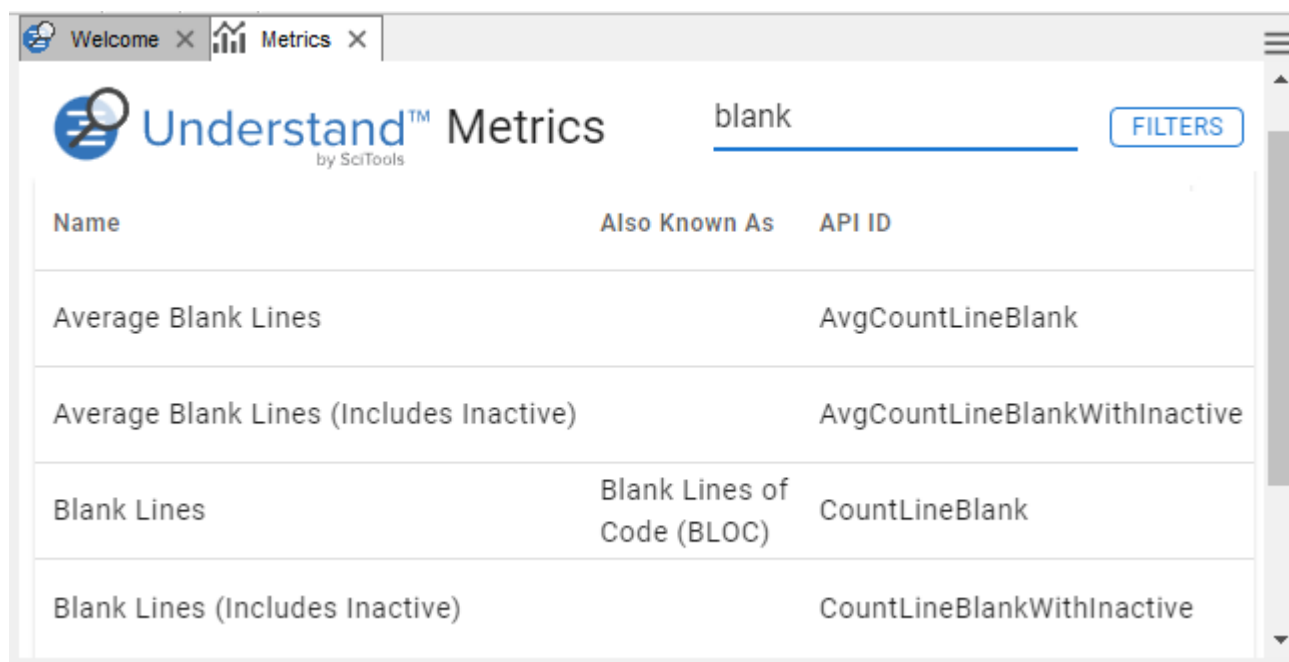
Metric Definitions

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.

This list shows all metrics supported by *Understand*, including those not shown in the Metrics Browser and other windows due to the project configuration (see [page 63](#)).

You can filter this list by typing in the **Search** field or by clicking the **Filters** button and choosing a category of metrics (Complexity, Count, or Object-oriented) or a language.




The screenshot shows the 'Understand Metrics' window. At the top, there's a search bar with the text 'blank' and a 'FILTERS' button. Below the header, there's a table with three columns: 'Name', 'Also Known As', and 'API ID'. The table lists several metrics related to blank lines.

Name	Also Known As	API ID
Average Blank Lines		AvgCountLineBlank
Average Blank Lines (Includes Inactive)		AvgCountLineBlankWithInactive
Blank Lines	Blank Lines of Code (BLOC)	CountLineBlank
Blank Lines (Includes Inactive)		CountLineBlankWithInactive

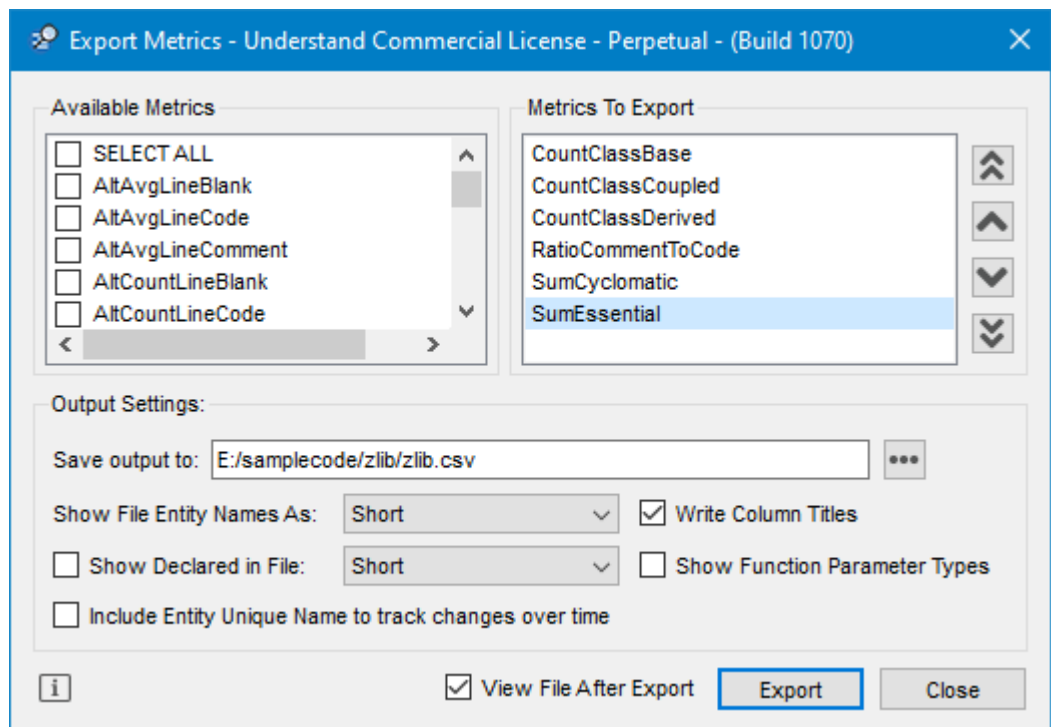
Information about the metrics is also provided in PDF format in the “doc/manuals” folder of the *Understand* installation.

Note: The API Names for some metrics were changed in *Understand* 6.4 to make the naming more consistent. See the *Understand* blog for details.

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 in the Metrics Browser 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).

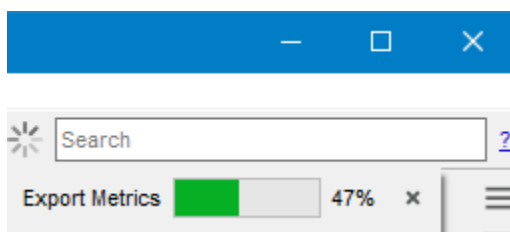
- **Show Declared in File:** Check this box if you want the file in which each entity is declared to be included in the output. You can specify whether you want these files displayed with **Short** names, **Full** names, or **Relative** names.
- **Write Column Titles:** Check this box if you want column 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_UniqueName” 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, the Metrics Browser 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 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 you click **Export**, a progress bar in the upper-right portion of the *Understand* window shows how much of the process is complete.




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.

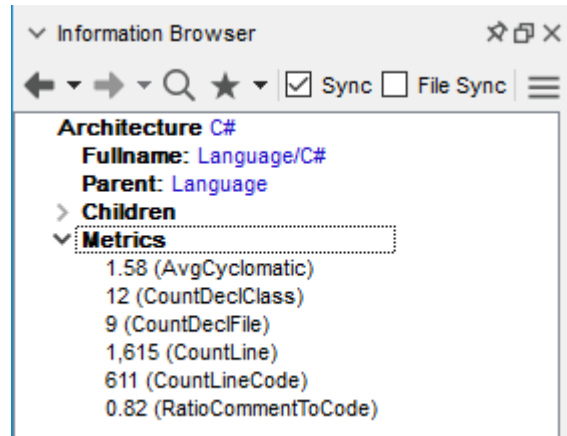
Metrics in the Information Browser

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 248](#) for details.

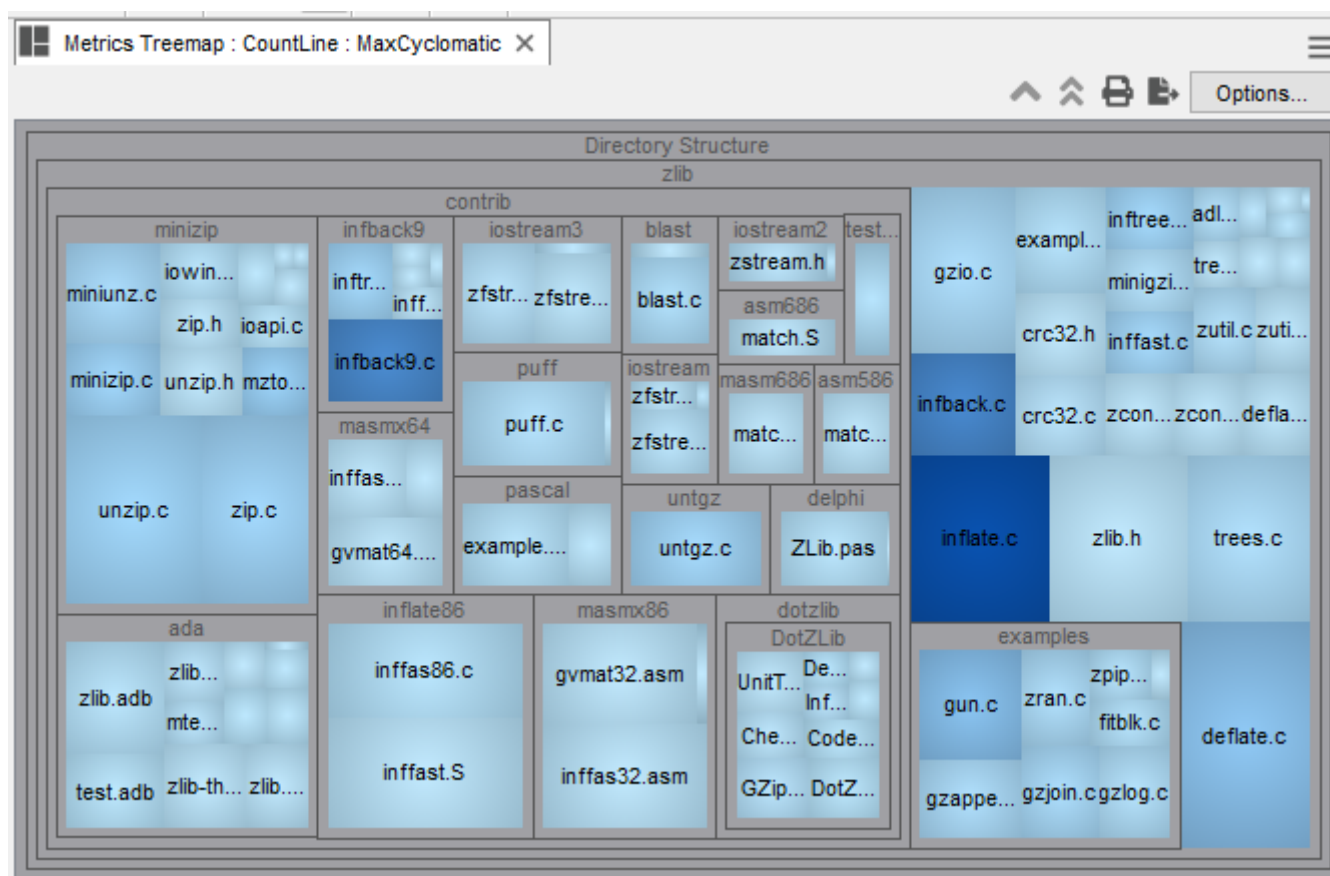
By default, the Metrics node in the Information Browser is collapsed whenever a new entity is viewed. If you want the node to stay expanded, you can click the small arrow to the right of the word “Metrics” and choose **Allow Pre-expansion**. Viewing new entities in the Information Browser may be slower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

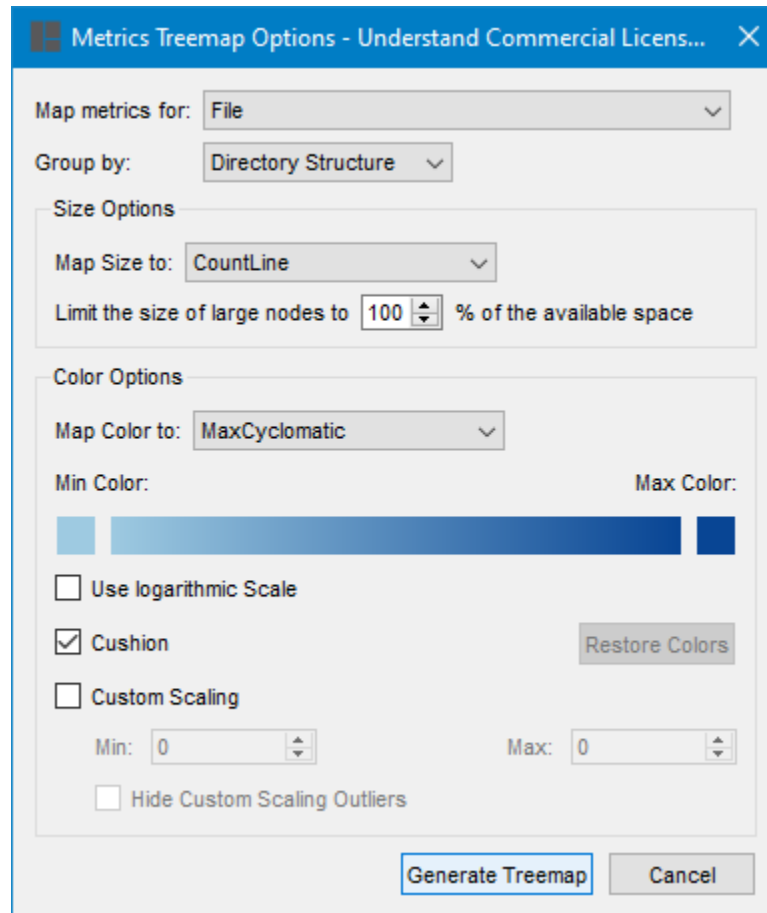


So we learn that `unzip.c` is large, but not particularly complex, while `inflate.c` is large and highly complex.

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.

Metric Plugins

Metric plugins are Python scripts that can be installed similarly to any other *Understand* plugin. (The Perl API is not supported for metric plugins.) Once installed, the custom metrics are listed in any metric list where they are applicable.

See the `plugins/Metric` directory in your *Understand* installation or the Metric folder in the Git repository at <https://github.com/stinb/plugins> for example metric plugins. For example, there are sample metric plugins related to code coverage, the amount of code churn based on Git commits, and the Halstead Maintainability Index. See the *Understand* blog for more information.

A metric plugin should expect an object with information about the requested metric, a database, and a target. The target can be an entity, architecture, or database.

A metric plugin must contain a “value” function that returns a number. The other functions in a metric plugin provide information about the metric (ID, name, description) and when it is available.

If a metric plugin is slow, remember that metrics are generally calculated by background threads. See *Analyzing the Code on page 102* for information about monitoring background processing.

To install a metric plugin, drag the .upy file into the Understand GUI, it will ask if you want to install it. Alternatively, you can manually place it in one of the following locations:

- **Windows:** `C:\Program Files\SciTools\conf\plugin\User\Metric` or `C:\Users\<user>\AppData\Roaming\SciTools\plugin\Metric`
- **Linux:** `/home/<user>/.config/SciTools/plugin/Metric`
- **MacOS:** `/Users/<user>/Library/Application Support/SciTools/plugin/Metric`

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category on page 109*).

Then choose **Tools > Clear Script Cache** from the menus or restart *Understand* to see your plugin.


This chapter covers the graphical views in *Understand* and their options.

This chapter contains the following sections:

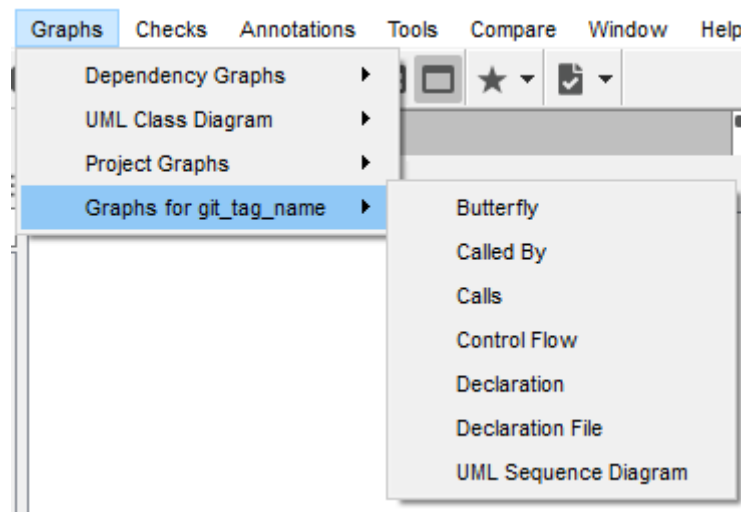
Section	Page
Opening Graphs	258
Choosing Graph Variants	261
Controlling Graphical Views	263
Controlling Graph Styles	271
Using Context Menus for Graphs	277
Saving and Exporting Graphical Views	285
Printing Graphical Views	286
Importing Graphical View Plugins	287

Opening Graphs

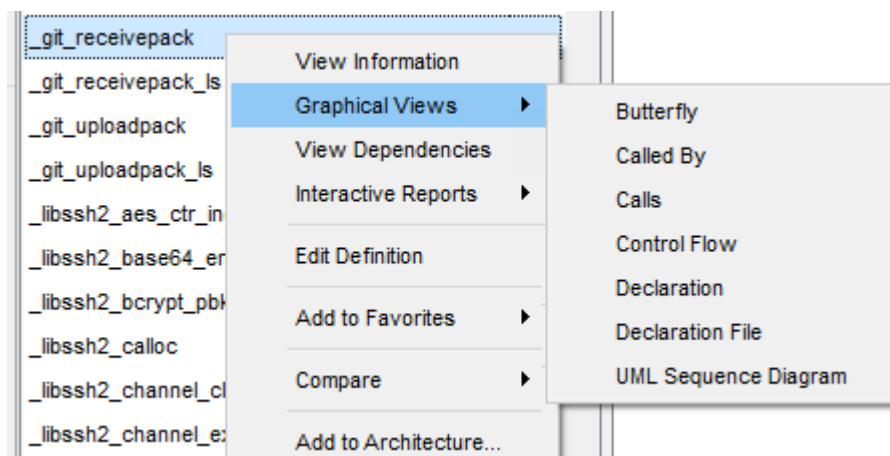
Understand provides many types of graphs for visually exploring and explaining your project. The graph categories available change based on the type of entity or architecture node you select. You can select entities or nodes almost anywhere in *Understand*. The list of graph categories is provided for 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.

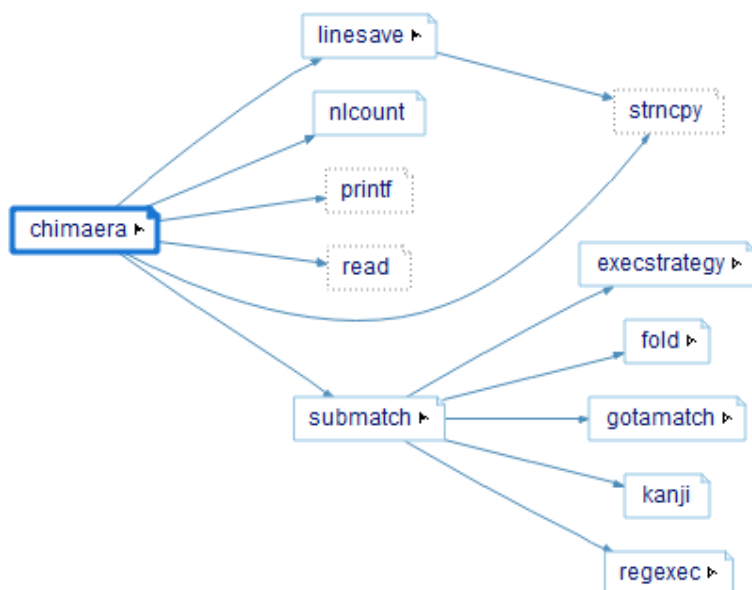


The same graph categories are listed when you right-click on an entity and choose **Graphical Views**:

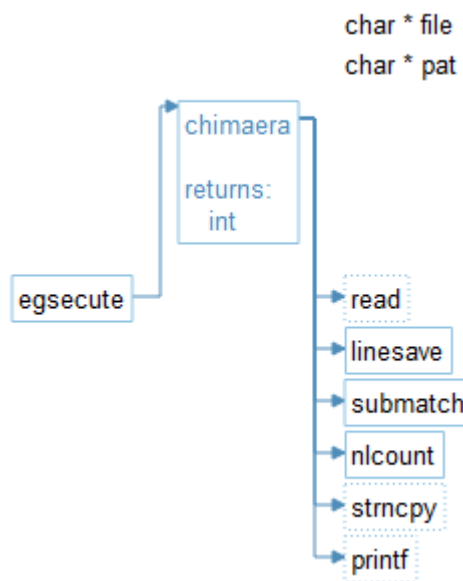


There are many graph types available in *Understand*. Graphs that show the hierarchy of calls and the structure of types are available for all languages that support calls and types. For example, here are two graphs for a C function called “chimaera”:

Calls Graph (simplified)



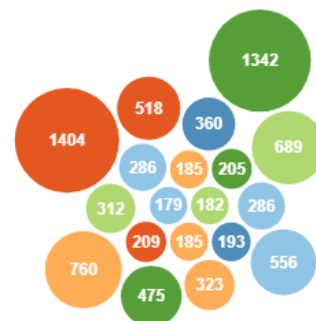
Declaration Graph



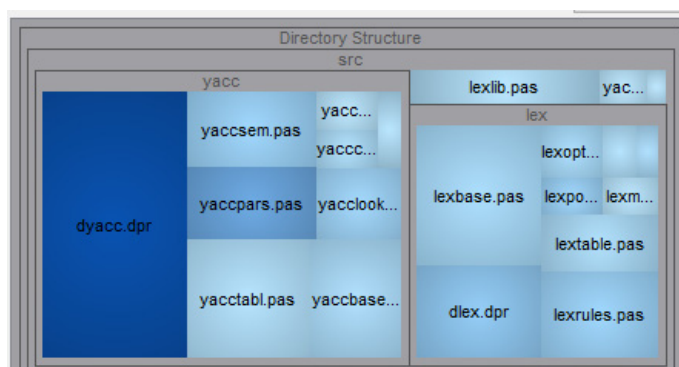
Some graphs apply to certain programming languages. For example, there are Derived Classes and Base Classes graphs for languages that support class inheritance, such as C++. Likewise, there are graphs to show the hierarchy of “include”, “with”, “use”, or similar statements, depending on the keywords 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 > Home** from the menus. These graphs show project-level metrics, such as relative function size and complexity. See [page 247](#).



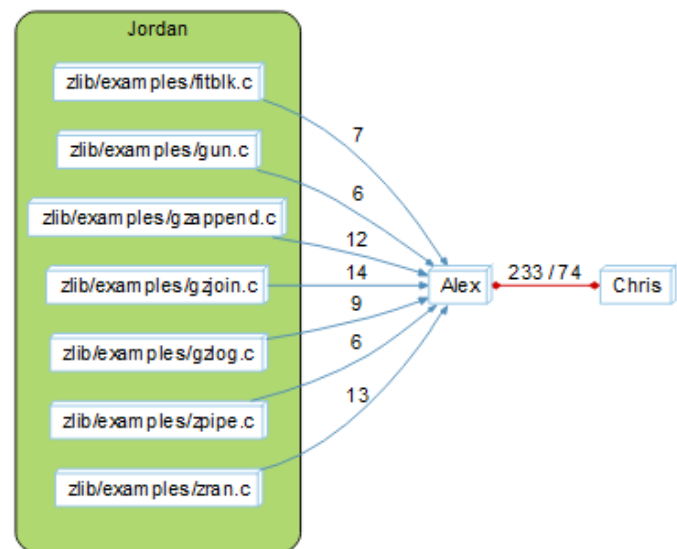
Metrics treemaps can be opened by choosing **Metrics > Metrics Treemap** from the menus. These are special graphs that can be generated using any two metrics. See [page 253](#).



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 220](#).

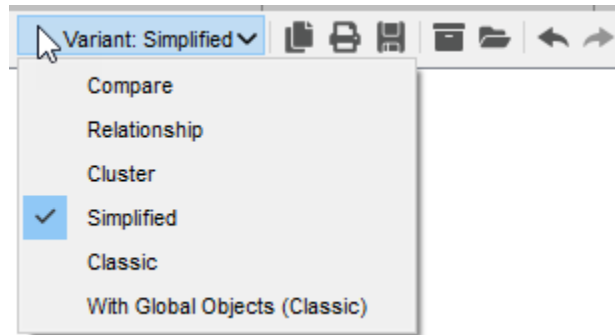


Additional graph types include Variable Tracker graphs (for objects) and Data Flow graphs (for objects and files).

Plugin graphs can be created using the Perl API or the Python API. To open plugin graphs you have installed, choose from the **Graphs** list. See [page 343](#).

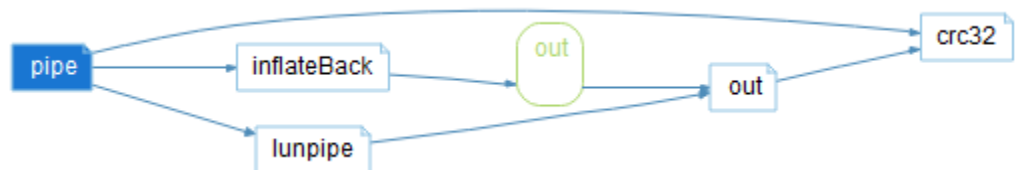
Choosing Graph Variants

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:

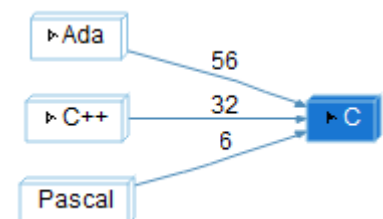


Choose a variant to view that graph. The variants available depend on the entity and graph category you selected, but these are some common variants:

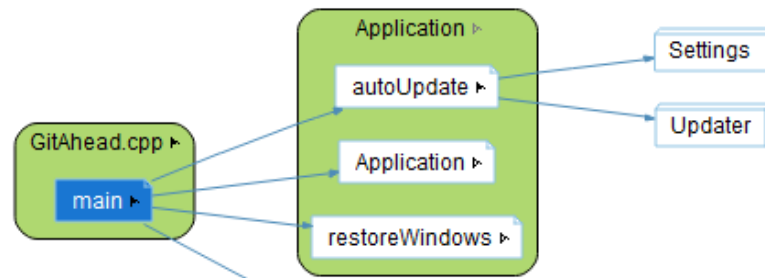
- **Simplified:** This is the default variant. (See [page 130](#) to change this.) For functions, it presents a simple view of the call, call by, or butterfly call tree. For objects or types, it presents a simple view of its members. You can right-click on the graph to control a number of display options. See [page 277](#) 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 277](#) 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()`.



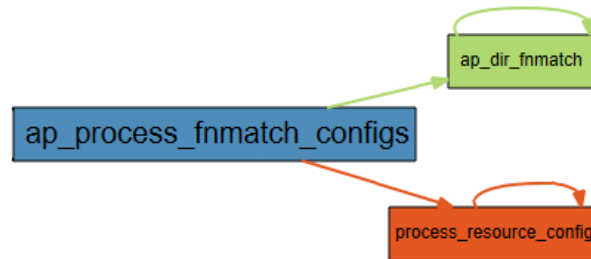
- **Dependencies:** This variant contains nodes drawn as 3D boxes (like those in this figure) that can be expanded to show the nodes they contain by double-clicking on them. You can keep expanding nodes until you get to the entity level.



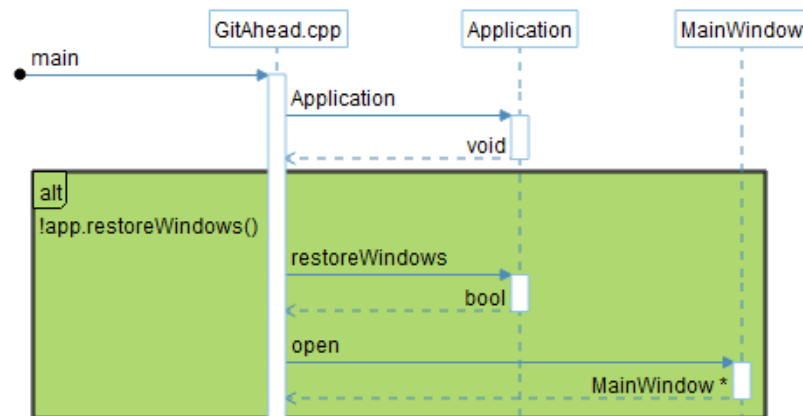
- **Cluster:** This variant provides a view of relationships that is clustered by classes or files. These graphs can be expanded or collapsed by double-clicking on a box in the graph. Cluster graphs can be accessed from the function, class, file, or architecture level.



- **Compare:** This variant shows how changes to a project affect the call structure, data structure, control flow, or UML Sequence. To view these graphs, you need to point to a previous version of the same project. See [page 323](#).



- **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. This graph is also supported for Ada. 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 344](#).

Controlling Graphical Views

The following sections describe controls for using graphical views.


- *Graph Toolbar on page 263*
- *Graph Display Options on page 265*
- *Mouse Controls on page 266*
- *Classic Context Menu Controls on page 268*
- *Customization Panel Controls on page 266*

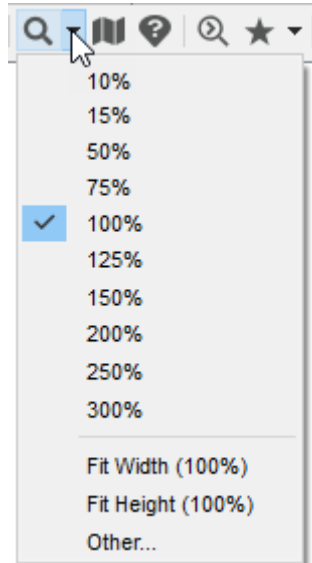
Graph Toolbar




The toolbar for graphs may contain the following icons. Some of the sections of the toolbar may be hidden by default or by your settings. Use the **Graph** category of the **Tools > Options** dialog to control which toolbar icons are shown ([page 130](#)).




- **Variant: Simplified** **Variant list:** Select a variant of this graph category to view. See [page 261](#).
- **Copy Image to Clipboard:** Allows you to paste the graph as an image into another application. See [page 285](#).
- **Print Graph:** Opens the Print Options dialog to print the graph. See [page 286](#).
- **Save Graph to File:** Allows you to save a graph as a JPEG, PNG, SVG, Visio, DOT, or PU file. See [page 285](#).
- **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**. This icon is shown in the graph toolbar if you enable the Archive toolbar in the Graphs category of the **Tools > Options** dialog; it is hidden by default.
- **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**. This icon is shown in the graph toolbar if you enable the Archive toolbar in the Graphs category of the **Tools > Options** dialog; it is hidden by default.
- **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.

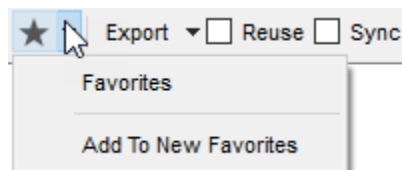
-  **Zoom:** You can zoom in or out using this drop-down list or the mouse scroll wheel. If you do not want the mouse scroll wheel to control the zoom level, you can disable that behavior in the Graphs category of the Options dialog ([page 130](#)).









-  **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 and can be used to customize the graph styles extensively (see [page 271](#)). 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 the same way you use it in the Source Editor to find entities by name or other text in the current graphical view. As you type search text, all instances of the string are highlighted in the graphical view. See [page 176](#).



-  **Favorites:** Select 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 155](#). This icon is shown in the graph toolbar if you enable the Favorites toolbar in the Graphs category of the **Tools > Options** dialog; it is hidden by default.



-  **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. This icon is shown in the graph toolbar if you enable the Reuse toolbar in the Graphs category of the **Tools > Options** dialog; it is hidden by default.
-  **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. This icon is shown in the graph toolbar if you enable the Released Windows toolbar in the Graphs category of the **Tools > Options** dialog; it is hidden by default.
-  **Configure Graph Settings:** Open the Graphs category of the **Tools > Options** dialog ([page 130](#)).
-  **Progress:** This icon is shown to the right of the Configure icon if a graph is currently being updated. A progress bar is also shown if needed.
-  **Graph Settings:** The hamburger menu provides the same commands as right-clicking on the background of the graph ([page 277](#)). In addition, use the **Toolbar Sections** submenu to hide or display icons in the graphical view toolbar ([page 130](#)) and the **Release Window / Capture Window** commands to move the graph to or retrieve it from a separate window.

Graph Display Options

To control the font used in graphs, set the **Application Font** in the **General** category of the **Tools > Options** dialog. See *General Category* on [page 109](#).

The **Legend** in each graph provides extensive control over colors, shape fills, shapes, and arrows used in graphs. See *Controlling Graph Styles* on [page 271](#). (Previously, some of these options were available in the Cluster Graph Styles section of the **Graphs** category in the **Tools > Options** dialog).

The **Graphs** category in the **Tools > Options** dialog lets you control graph behaviors such as what happens when you hover the mouse over a graph, zoom in or out, or double-click on a node or cluster. You can also use that dialog to set the default graph variant, the icons shown in the graph toolbar, and whether clusters in cluster graphs are filled. See *Graphs Category* on [page 130](#).

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 277).

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 277). 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 130.

In a control flow graph, right-click on any box in the flow chart and choose **Go To Source** to open the source code to the line that performs this action. In the Source Editor, right-click on any line of code in a function or method and choose **Graphical Views > Control Flow** to open a control flow graph with this action selected.

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.

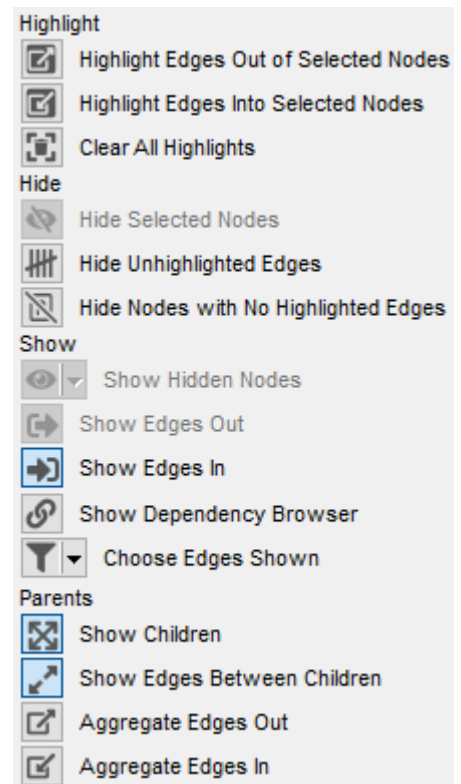
Double-click on an entity in a graph to expand that entity if possible.

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.

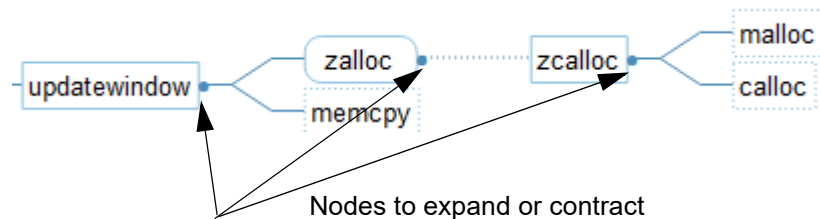
- **Clear All Highlights.** Click this button to turn off all node and edge highlighting.
- **Hide Selected Node(s).** Removes all the nodes that are currently selected from the graph and reorganizes the graph as needed. (You can later restore the hidden nodes by clicking Show Hidden Nodes.)
- **Hide 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. You can override this global setting for individual entities by selecting an entity and choosing **Show Edges Out** from its context menu. The next time the global setting is changed, any settings for individual entities are discarded.
- **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. You can override this global setting for individual entities by selecting an entity and choosing **Show Edges In** from its context menu. The next time the global setting is changed, any settings for individual entities are discarded.
- **Show Dependency Browser.** This button opens the Dependency Browser ([page 148](#)) 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.

Classic Context Menu Controls

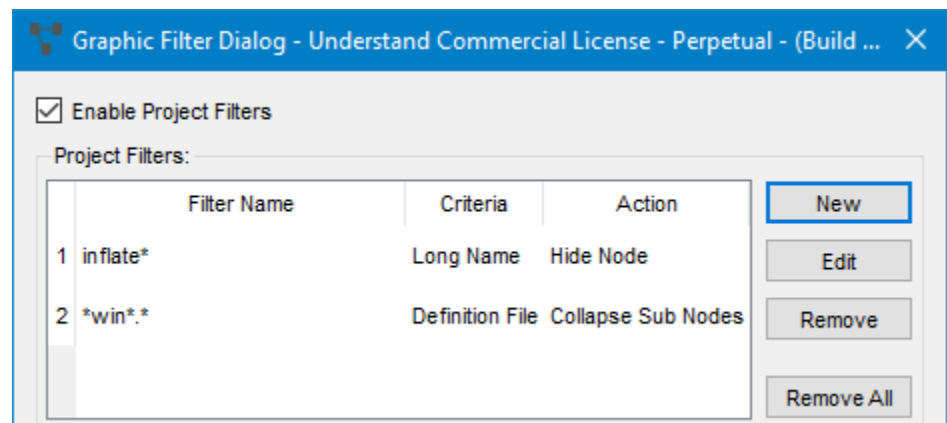
In Classic graph variants and some other graph types, commands besides the **Entity** submenu and **Graph** submenu may be provided for actions like the following:

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**.



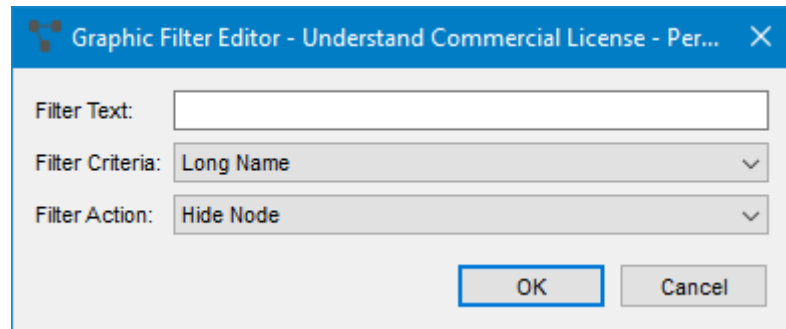
Filtering graph contents: In Classic graph variants, you can apply filters to hide certain entities in some graphical views. To create such a filter, follow these steps:

- 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.



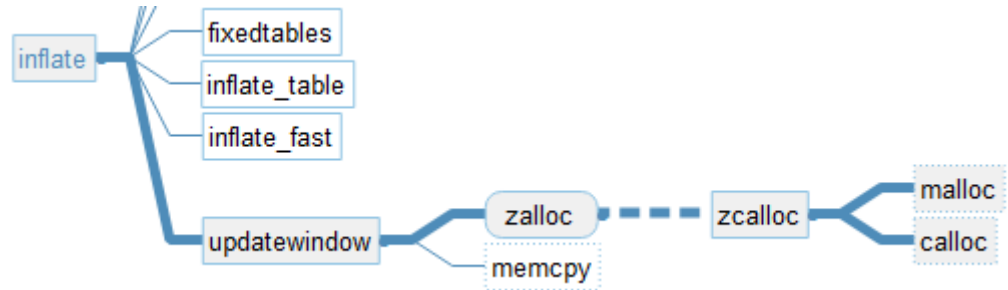
- 4 Type a filter in the **Filter Text** field. For example, use `gr*` to match entity names beginning with `gr`. Filters are case-sensitive.
- 5 In the **Filter Criteria** field, select whether to compare the filter to long names, definition files, the type text of entities, or parameter text. For example, 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.

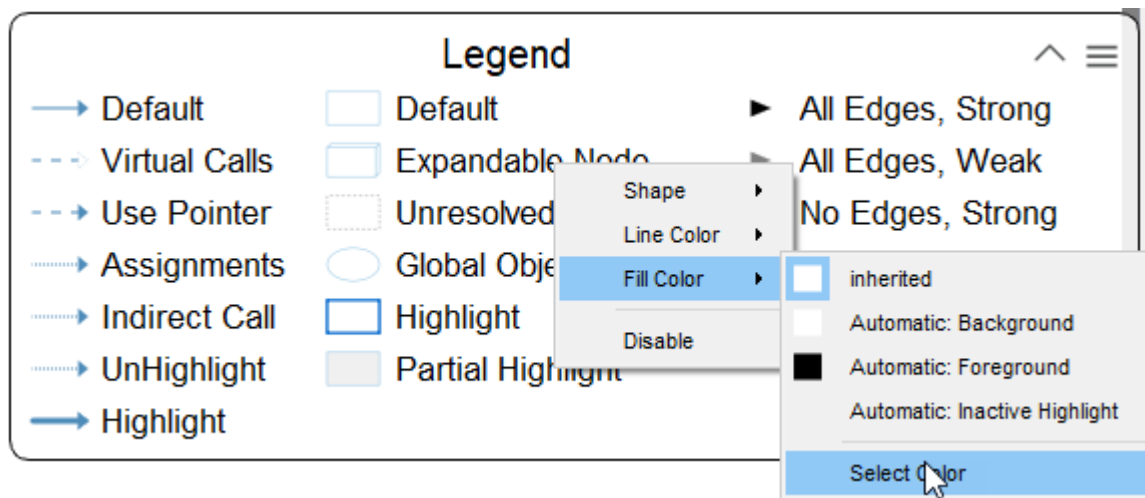
Path highlighting: In certain graph variants with a single root node and direction, you can highlight the path to one or more entities in a tree view (such as a Callby view) by selecting the entity and right-clicking. In the context menu, choose **Highlight Path**. Hold down the Ctrl key to select multiple entities for path highlighting.



Controlling Graph Styles


The **Legend** in each graph provides extensive control over colors, shape fills, shapes, and arrows used in graphs. You can access these controls by right-clicking on items and by using the  hamburger menu.


You can manage collections of styles (themes), share themes with other users, and use different styles with different types of graphs.



(Previously, some of these options were available in the Cluster Graph Styles section of the **Graphs** category in the **Tools > Options** dialog).

Viewing the Legend



If you do not see a Legend in a graph, toggle the  **Show Legend** icon on in the graph toolbar to on. If you do not see the Show Legend icon, use the **Graphs** category in the **Tools > Options** dialog to enable the Navigation toolbar (see [page 130](#)).


To minimize the legend in a graph, click the up arrow  next to the hamburger menu. To maximize a legend that has been minimized, toggle the down arrow.

Legends are available for most graph types. You can drag the legend to reposition it within the graph. The legend identifies shapes and line styles used in the graph.

Using the Legend: Summary

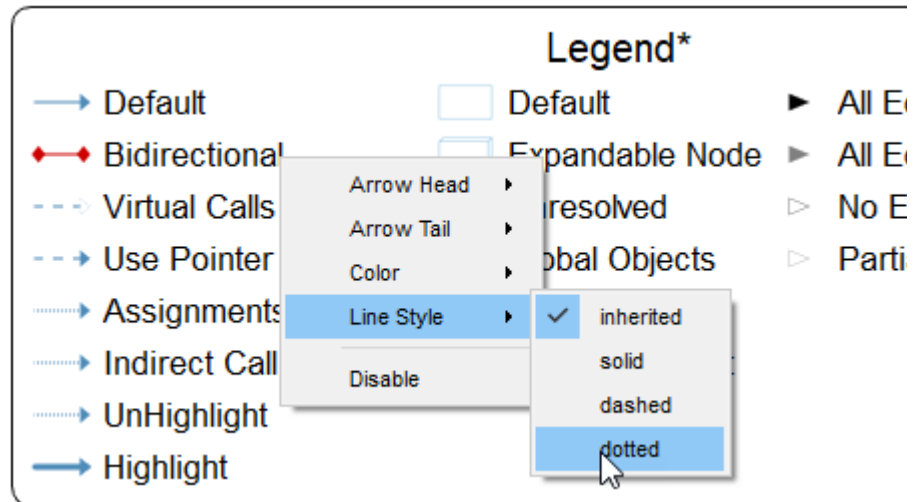
You can use the legend to customize graph styles extensively:

- **Change line style, color, and arrow type:** Right-click on any line style in the first column of the legend. See [page 272](#).
- **Change shapes and shape fill and line colors:** Right-click on any shape in the second column of the legend. See [page 272](#).
- **Add node style:** Click the  menu, and choose **Add Node Style**. See [page 272](#).
- **Add edge style:** Click the  menu, and choose **Add Edge Style**. See [page 273](#).
- **Hide node or edge style:** Right-click on an item in the first or second column and choose **Disable**. See [page 272](#).

- **Add a color gradient scale:** Click the  menu, and choose **Add Metric Color Scale**. See [page 274](#).
- **Manage graph themes:** See [page 275](#).

Modifying Styles in the Legend

Edges (lines): To change the style, color, or arrow type of an edge type, right-click on that type and choose a new setting for Arrow Head, Arrow Tail (for Bidirectional edges), Color, or Line Style.



Nodes (shapes): To change the shape, line color, or fill color of a node type, right-click on that type and choose a new setting for Shape, Line Color, or Fill Color.


Defaults and Inheritance: For edges, there are separate Default, Highlight, and UnHighlight types. For nodes, there are separate Default, Highlight, and Partial Highlight types. Most other types inherit their settings from one of these types. If you change a setting for the Default type, other types will also change if they inherit those settings. For colors, automatic foreground and background settings may come from your computer's system settings.


Graphs are updated to use new settings.

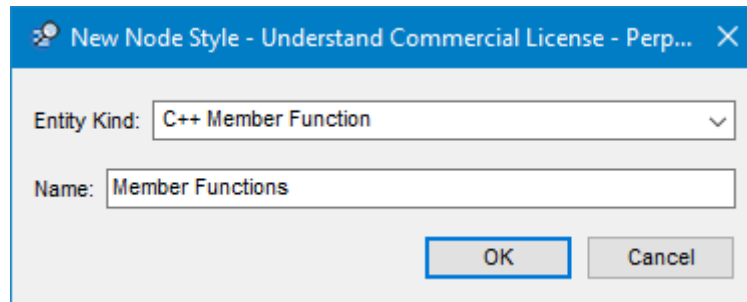
Adding Node Styles

You can add node styles to control how those types or nodes are shown. Adding styles does not change which kinds of nodes are shown in the graph—that is controlled by the Graphical View and Variant you have selected. Instead, adding node styles lets you highlight various subtypes of entities. For example, for C++ projects you can choose different ways to display classes, functions, member functions, unresolved functions, and virtual functions.

To add node styles:

- Click the  hamburger menu, and enable a style from the **Global Node Styles** list. The styles available depend on the graph type.

- Click the  hamburger menu, and choose **Add Node Style**.



The dialog box titled "New Node Style - Understand Commercial License - Perp..." has a close button (X) in the top right. It contains two input fields: "Entity Kind:" with a dropdown menu showing "C++ Member Function" and a small downward arrow, and "Name:" with a text box containing "Member Functions". At the bottom right are "OK" and "Cancel" buttons.

The **Entity Kind** determines to which entities a style applies. You can select from the drop-down list or type a custom kind. See the “Kind Filters” topic in either the Perl API or Python API documentation for a list of the available entity kinds. Look for the “<language> Entity Kinds” heading for a language used in your project.

The **Name** will be the text shown in the legend for this node style.



When you add a node style, the new style appears in the legend immediately, but it initially inherits all values from the default style. So, you will not see a change in the graph unless you change the display style for the node type you added as described in *Modifying Styles in the Legend* on [page 272](#).

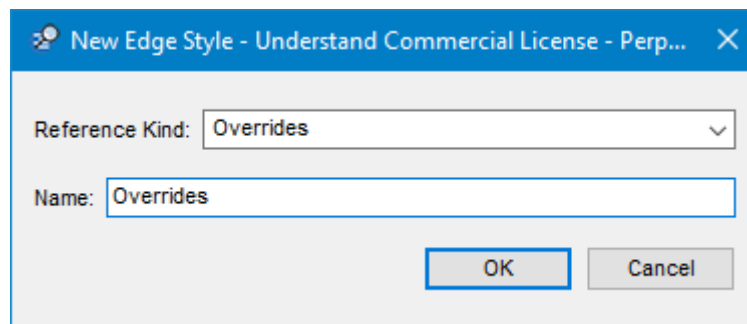
If an entity matches more than one node kind, it uses the last style listed in the legend. For example, a header file is both an expandable node and a header file. If you have styles for both of these, the style listed last determines the shape and color of header files in a graph.

Adding Edge Styles

Adding an edge (line) style is similar to adding a node style. Edge styles can be added only to graphs with references as edges. For example, they do not apply in architecture graphs or control flow graphs. The rule for the style is based on reference kinds:

To add edge styles:

- Click the  hamburger menu, and enable a style from the **Global Edge Styles** list. These styles available depend on the graph type.
- Click the  hamburger menu, and choose **Add Edge Style**.



The dialog box titled "New Edge Style - Understand Commercial License - Perp..." has a close button (X) in the top right. It contains two input fields: "Reference Kind:" with a dropdown menu showing "Overrides" and a small downward arrow, and "Name:" with a text box containing "Overrides". At the bottom right are "OK" and "Cancel" buttons.

The **Reference Kind** determines to which relationships a style applies. You can select from the drop-down list or type a custom kind. See the “Kind Filters” topic in

either the Perl API or Python API documentation for a list of the available entity kinds. Look for the “<language> Reference Kinds” heading for a language used in your project.

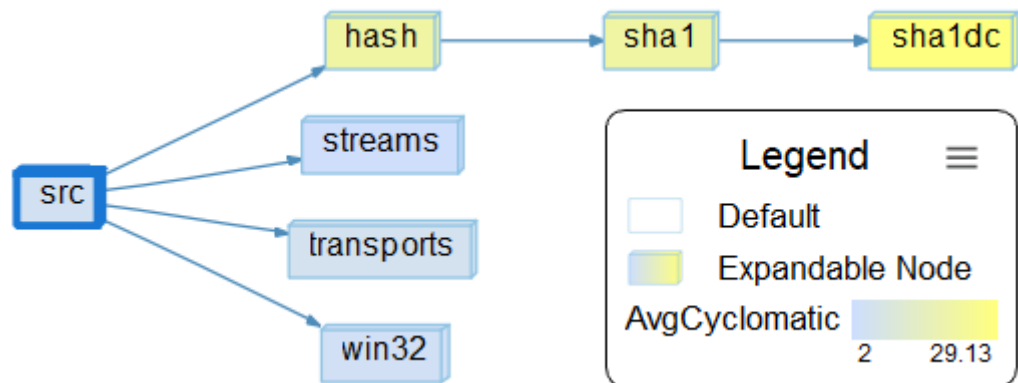
The **Name** will be the text shown in the legend for this edge style.

When you add an edge style, the new style appears in the legend immediately, but it initially inherits all values from the default style. So, you will not see a change in the graph unless you change the display style for the edge type you added as described in *Modifying Styles in the Legend* on [page 272](#).

If a relationship matches more than one edge kind, it uses the last style listed in the legend.

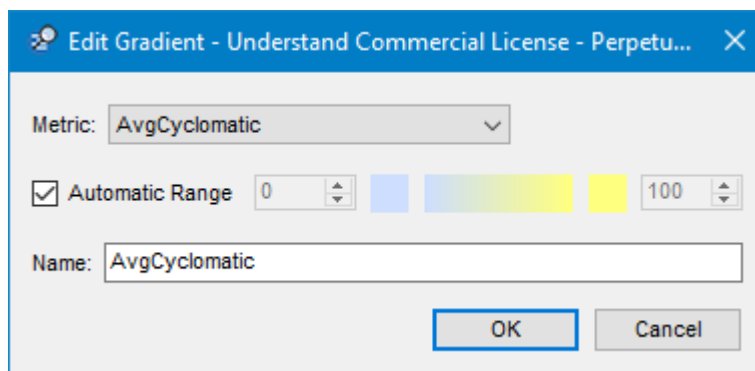
Shading Nodes or Edges Using a Metric

You can shade nodes (line color or fill color) or edges in a graph according to the value of a metric. For example, this graph shows that the hash node and its subnodes in this architecture have the highest AvgCyclomatic complexity. The minimum and maximum complexities in the graph are shown in the legend:



To shade nodes or edges, follow these steps:

- 1 Click the hamburger menu, and choose **Add Metric Color Scale**.
- 2 In the Edit Gradient dialog, choose a Metric for shading nodes or edges. For descriptions of individual metrics, open the Metrics Browser ([page 248](#)) and click the **Show Metrics Definitions** information icon in the toolbar.




- 3 By default, the range for the gradient automatically adapts to the minimum and maximum values of nodes or edges in your graph. You can uncheck the **Automatic Range** box and set a minimum value (lower values will all use the minimum value color) and a maximum value (higher values will all use the maximum value color).
- 4 To change the colors of the gradient, click on the squares to the left and right of the gradient bar. The default colors are white and blue.
- 5 Click **OK** when you have finished choosing settings. You can come back to make changes by right-clicking on the metric bar added to the legend and choosing **Edit**.
- 6 Notice that the new scale appears in the legend, but your graph doesn't change at all. This is because you still need to apply the gradient to a node or edge style.
- 7 Right-click on a node or edge style in the legend and choose **Fill Color** or **Line Color** for a node style or **Color** for an edge style. Choose the scaled metric you added.

You can create multiple metric color scales for different metrics and apply them to different node and edge styles.



For example, you might want to use a metric color scale related to CodeCheck violations or code coverage for graph edges. Such metrics are available in the metrics plugins in the `plugins` directory of your *Understand* installation.

You cannot apply a metric color scale to the Default node or edge style. This is because the default is needed for nodes and edges to which the metric cannot apply. For example, Cyclomatic does not apply to files. Other node and edge types may inherit the metric color scale from the node or edge style to which you apply the gradient.

Removing Node and Edge Styles

Simplify the legend by hiding any styles that do not match any nodes or edges in the current graph by toggling **Show Unused Styles** in the  hamburger menu.

To remove node or edge styles from the theme:


- Right-click on a style in the legend and choose **Disable** for styles shown by default or added from a Global Styles list. These styles are built in and cannot be deleted, only disabled.
- Right-click on a style in the legend and choose **Delete** for styles you added with a Add Node/Edge Style command.
- Deselect an option in the **Global Node Styles** list in the  hamburger menu.
- Deselect an option in the **Global Edge Styles** list in the  hamburger menu.

Saving, Importing, and Exporting Styles

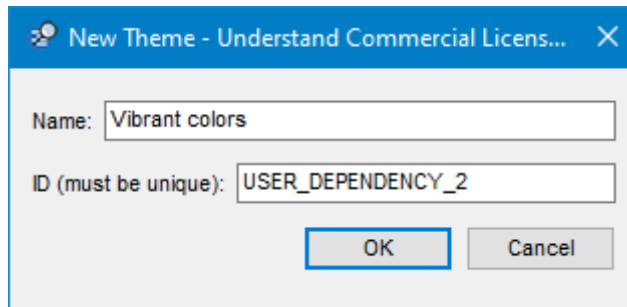
After you make a change using the legend, notice that the title has an asterisk (*) next to the word "Legend" to indicate that there are unsaved changes to the set of graph styles (called a "theme").

All graph types have a built-in "Basic" theme. Many graph types also have a default theme that is specific to the types of entities shown in that graph. For example, the "Calls" or "Object References" theme. Some graphs have extra variant-specific themes, like the "Compare" theme. You can customize and save all of these themes.

The changes to themes that you make and save are available in any project you open and are saved separately from projects. Each theme applies to a graph type. For example, changing a dependency graph theme has no impact on the display of control flow graphs.

You can save styles to named themes, switch themes, and import and export themes. Use these commands in the  hamburger menu to manage graphical view themes:

- **Save current theme:** Choose **Theme > Save <name>**. All other graphs that use the same theme will have the saved changes.
- **Revert to the default settings:** Choose **Theme > Restore Default for <name>** to reset all the styles to their last saved settings.
- **Create a new theme:** Choose **Theme > New Theme**. If you have changed any styles, you will be asked if you want to save your changes before creating the new theme. If you choose **Save**, your current styles are saved both as part of the current theme and the new theme. If you choose **Discard**, your current styles are saved only as part of the new theme. Type a **Name** for the new theme. An **ID** for the theme is provided, and should not duplicate other themes.

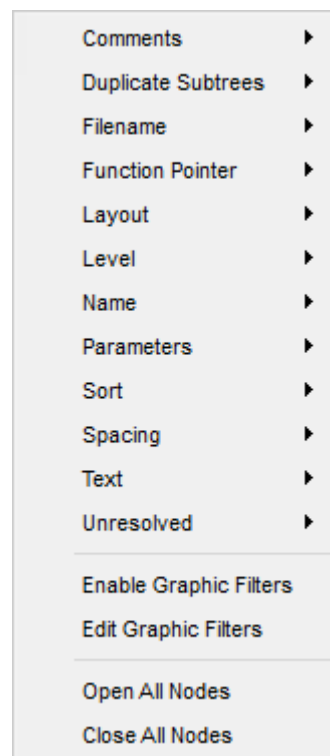


- **Switch to a different theme:** Choose **Theme > <name>** to display the current graph using a different saved theme.
- **Rename a theme:** Choose **Theme > Rename <name>** and type a new name for this theme ID.
- **Remove a theme:** Choose **Theme > Remove <name>** to remove the current theme and switch to the previous theme in the list.
- **Export a theme:** Choose **Theme > Export <name>** and choose the location and filename to store the theme as a JSON file. You can share this file with other users.
- **Import a theme:** Choose **Theme > Import <name>** and browse for the location of a JSON file that contains a theme you want to use. If the ID for a theme you import matches an existing theme, you will be asked if you want to overwrite your existing theme. Importing a theme does not change your selected theme. After you import a theme, you can switch to it.

Using 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. This section describes many, but not all, of those submenus, which 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. You can override this global setting for individual entities by selecting an entity and choosing **Show Annotations** from its context menu. The next time the global setting is changed, any settings for individual entities are discarded.
- The **Architecture Labels** menu controls whether architecture nodes are labeled in File Dependency graphs. The default is on.
- The **Architecture Name** menu controls how architecture names are displayed. The options are none, short, relative to the project, and long names. The default is relative.
- The **Base Classes** menu controls whether declaration graphs include base classes if applicable.
- The **Bidirectional Edges** menu controls separate or a single connection is shown between entities that have multiple relationships. The default is to show each edge separately. This affects Simplified, Cluster, Relationship, and Compare variants of certain graphs.
- The **Calls Depth** menu controls how many levels of calls are displayed. This setting may range from 0 to 10. The default is 3 levels.
- The **Called By Depth** menu controls how many levels of callbys are displayed. This setting may range from 0 to 10. 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 **Code Style** menu controls whether code shown in a graph is shown in normal text or with highlighting (for example, keywords and strings) as shown in the Source Editor. Additionally, you can choose to expand macro code. The default is unstyled.
- The **Collapse** menu controls whether or not to combine statements into a single box if there are no decision points between them in a control flow graph. The On Respect Hidden setting is like the On option except that hidden nodes remain hidden. The default is On.
- The **Comments** menu controls whether any comments associated with entities are shown. The default is off. You can override this global setting for individual entities by selecting an entity and choosing **Show Comments** from its context menu. The next time the global setting is changed, any settings for individual entities are discarded.
- The **Constants** menu controls whether to show constants in Declaration views. The default is on.
- The **Data Flow In Depth** menu controls how many incoming levels are shown in Data Flow graphs, which are available for global objects and files. You can right-click on the edges in a Data Flow graph for a list of links to the relevant code.
- The **Data Flow Out Depth** menu controls how many incoming levels are shown in Data Flow graphs, which are available for global objects and files.
- The **Debug** menu controls whether to show details about each item in control flow graphs. In order, the detailed 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. This setting may range from 0 to 10.
- The **Depends On Depth** menu controls the number of levels of entities this entity depends on shown. This setting may range from 0 to 10.
- 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** or **Name Format** 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 75](#)).
- 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 Name** menu controls whether the name of the function in which an assignment is made is shown in Assignments and Assigned To graphs.
- The **Function Pointer** menu controls whether function pointers are displayed as invocations in the Call and CallBy trees and UML Sequence Diagrams.
- 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. To change the line style of virtual edges, see *Modifying Styles in the Legend* on [page 272](#).
- 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 in hierarchy and control flow graphs. For some graph types, the options are Horizontal and Vertical. For other graph types, the options are as follows:
 - **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 and can be used to customize the graph styles extensively (see *Controlling Graph Styles* on [page 271](#)).
- 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 10 may be set. This is available only in hierarchy views.

- The **Library** menu lets you exclude standard library entities from Ada call graphs.
- The **Local** 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/method 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 **Parameter Calls** menu controls whether parameter calls are shown in variable tracker views.
- 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 **Returns** menu controls whether returned variables are shown in variable tracker views.
- The **Routines** menu controls whether to show routines (procedures, functions, etc.) in Declaration views. The default is on.
- The **Search Level** menu controls the maximum number of calls or other relationship levels that can be searched in relationship graphs. The default is all levels. Other settings range from 1 to 10 levels.
- The **Show Assign References** menu controls whether assignment references are shown in variable tracker views. The default is on.
- The **Show Class Details** menu controls whether details about each class are shown.
- 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 on for architecture cluster graphs but off by default for other types of graphs.
- The **Show Entities** menu controls whether entities are shown in addition to architecture nodes in architecture dependency graphs. The default is on.
- 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 an architecture or 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. Right-click on a node and select **Show Children** to see the children of an individual node. The default is off.
- The **Show Related Classes** menu controls whether classes related to this class are shown in UML diagrams. The default is on.
- The **Show Return Type** menu controls whether UML diagrams show the return type of class functions. The default is on.
- 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 Self Edges** menu controls whether diagrams show relationships to the entity itself. The default is on for call graphs and off for other types of graphs.
- The **Show Solo Classes** menu controls whether comments associated with statements are shown in control flow graphs and UML diagrams.
- 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 Source Locations** menu controls whether source locations are hidden or shown in variable tracker graphs. The default is off.
- The **Show Version Differences** menu 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 and variable tracker graphs. The formats defined in the **Editor > Styles** category of the Understand Options dialog ([page 127](#)) 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 the entity that “Withs” a given compilation unit is shown in declaration views. The default is on.

Saving and Exporting Graphical Views

Understand offers various ways to save or export your graphical views. The toolbar for each graphical view provides the following icons for copying and printing graphs.




You can 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.

You can print a graphical view as described in *Printing Graphical Views* on [page 286](#).

For cluster graphs, you can save and load customized graph settings ([page 263](#)).

To export a graphical view to a file:

- 1 Click the  icon or choose **File > Export** from the menus.
- 2 In the Export to File dialog, choose a directory location and filename.
- 3 Use the **Save as type** drop-down to choose the type of file to export. You can save graphical views as JPEG, PNG, SVG, Visio, DOT, and PU files.
 - **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.
 - **VDX and VSDX** are files used by Microsoft Visio, which 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 file. The VSDX format is a newer and more compressed file format than the VSD format.
 - **DOT files** are 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. If this option is not shown in the **Save as type** drop-down, the current graph cannot be exported to this format. (This format is not the same as DOT template files used by Microsoft Word.)
 - **PU files** are used by PlantUML, an open-source tool for creating diagrams from plain text.


Note that if the graph will result in an image larger than 200 MB, the graph will be resized to a smaller size.

Printing Graphical Views

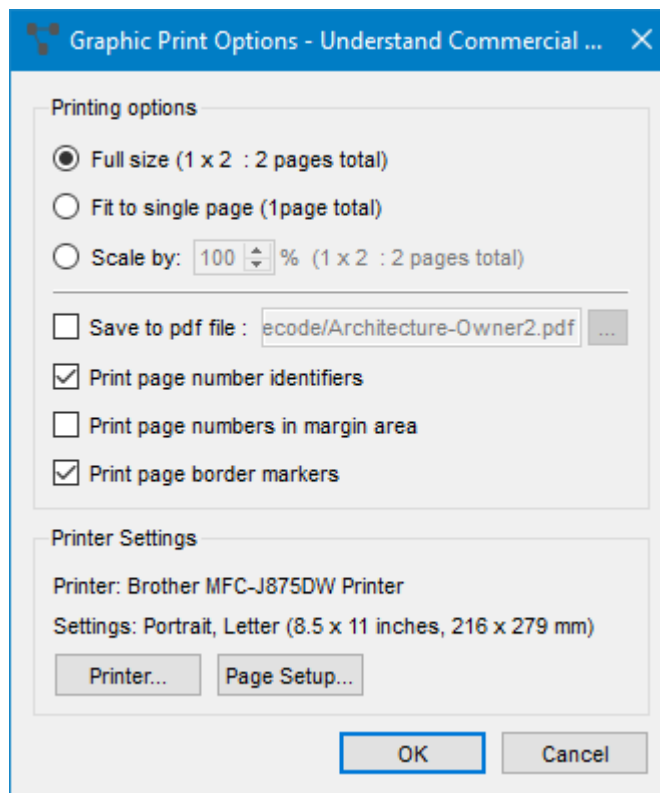
Understand has these printing modes:

- Source File printing sends a text file to the printer using 66 lines of source per page. See *Printing Source Views* on [page 208](#).
- Graphical view printing provides options for how to fit the image to a page. See *Graphical View Printing* on [page 286](#).

Graphical View Printing

To print the current graphical view, you can click  **Print** icon on the graphical view toolbar. Or choose **File > Print Entity Graph** from the menus.

When you choose to print a graphical view, you see the Graphic Print Options dialog.



You can choose to print the image at one of the following sizes:

- **Full size** uses the default scaling of 100%. The dialog shows the number of pages in width x height format. The page size selected with Page Setup is used.
- **Fit to a single page** scales the image to fit on the selected page size.
- **Scale by** lets you choose the sizing percentage and shows the number of pages that will be printed.

Check the **Save to PDF** file box if you want the image saved to an Adobe Acrobat file rather than being sent to a printer. This PDF printing feature does not require that you have third-party PDF generating software installed on your computer.

Check the **Print page number identifiers** box if you want page numbers on each page in the upper-left and lower-right corners. The page numbers are in “(column, row)” format. For example, (1,3) indicates that the page goes in the leftmost (first) column of the third row when you piece the pages together. The page number is not printed if the view fits on a single page.



Check the **Print page numbers in margin area** to place the page numbers outside the borders of the graph. If this box is unchecked, page number indicators are printed just inside the border markers.

Check the **Print page border markers** box to place corner markers on each page.

Click the **Printer** button to open the 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 text files. The **Page Setup** button on the Graphic Print Options dialog saves settings separately.

Importing Graphical View Plugins

Graphical view plugins are Perl or Python scripts that can be imported to provide customized graphical views. 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.

To install a graphical view plugin, drag the file to the *Understand* window. For example, a Perl API plugin would be a .upl file, and a Python API plugin would be a .upy file. You can also copy the plugin file to the appropriate subdirectory:

- **Windows:** C:\Program Files\SciTools\conf\plugin\User\Graph or C:\Users\<user>\AppData\Roaming\SciTools\plugin\Graph
- **Linux:** /home/<user>/.config/SciTools/plugin/Graph
- **MacOS:** /Users/<user>/Library/Application Support/SciTools/plugin/Graph

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category on page 109*).

Then choose **Tools > Clear Script Cache** from the menus or restart *Understand* to see your plugin within *Understand*.

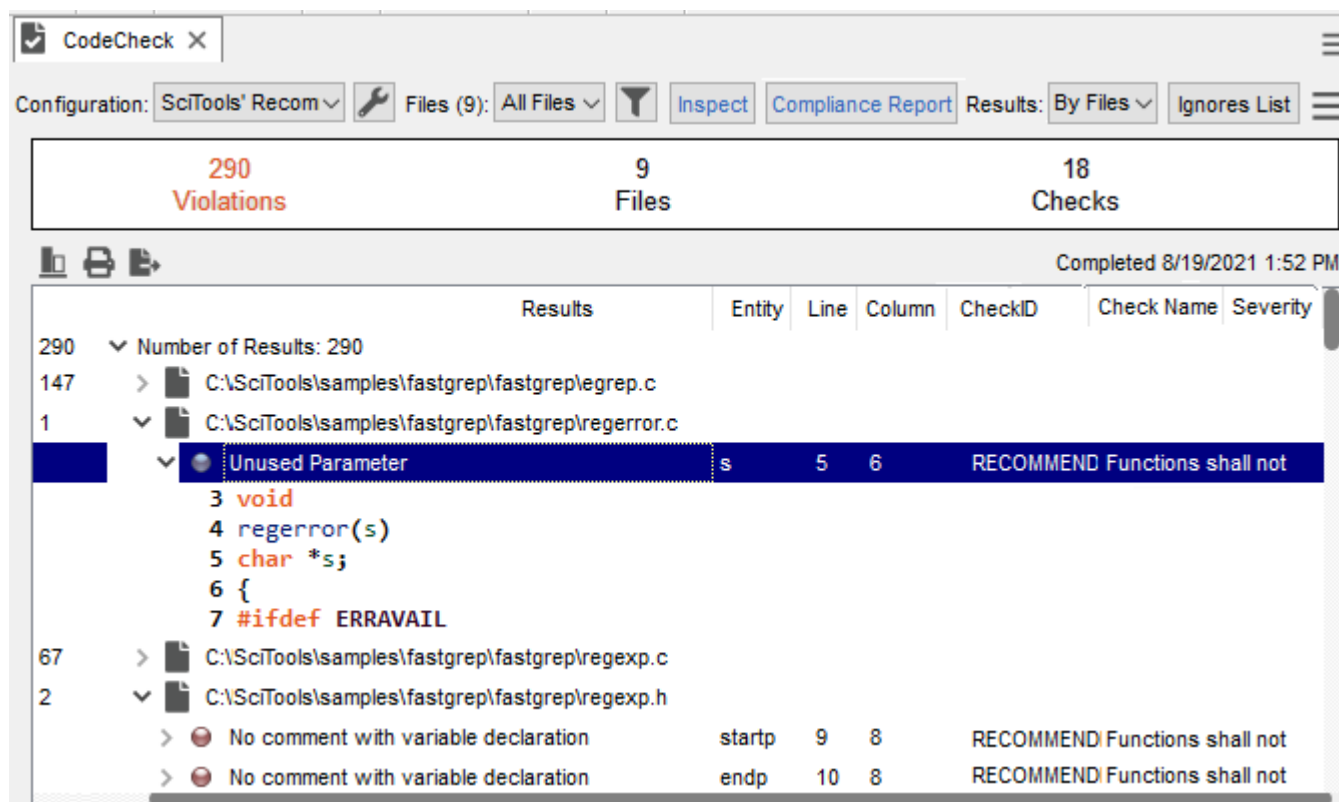
This chapter explains how to use CodeCheck to find places where your code does not conform to various standards.

This chapter contains the following sections:

Section	Page
About CodeCheck	289
Running a CodeCheck	290
Configuring Checks	291
Selecting Files to Check	295
Viewing Results	296
Ignoring or Excluding Violations	305
Exporting CodeCheck Results	310
Writing CodeCheck Scripts	312

About CodeCheck

Understand provides a tool called CodeCheck to make sure your code conforms to published coding standards or your own custom standards. These checks can be used to verify naming guidelines, metric requirements, published best practices, or any other rules or conventions that are important for your team.



Checks are available to make sure your code conforms to several published coding standards. You can select a subset of individual checks to test for from these standards. For example, you can check to make sure that all if...elseif constructs contain a final else clause.

For all languages, checks are provided to let you verify that various entity types conform to your naming conventions and to confirm that your code meets metric requirements you set for complexity, function length, and nesting depth.

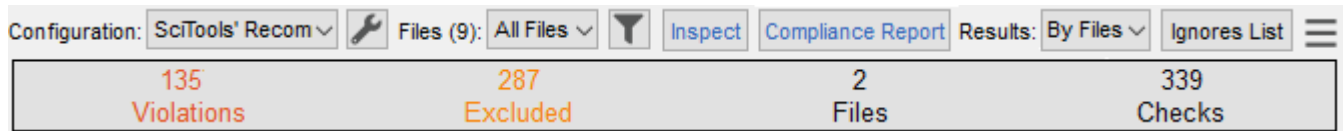
If you want to perform custom checks, you can create your own checks using Perl. For example, you can create a check to find lines longer than 80 characters or filenames that begin with a number.

CodeCheck validation suites are available in the CodeCheck category on the [support website](#).



To configure and run dependency checks on an architecture, see [page 222](#).

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 291*.
- 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, or an architecture. 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 295*. If files have been modified since the last project analysis, see *Re-Analyzing the Project on page 291*.
- 3 Click the **Inspect** button to run selected checks on the selected files. The checks are performed in the background and a progress bar is shown. Checks may take a significant time to perform for large projects. You can continue using *Understand* while checks are running. Violations are added to the results list as they are found.

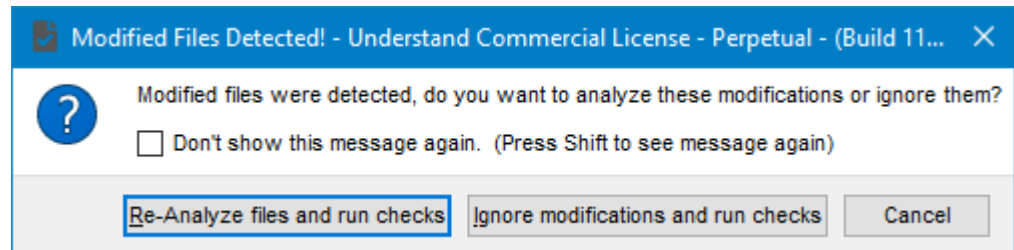
Choose **Checks > Re-Run Configuration_Name** to run the most recently run CodeCheck configuration. If you have made changes to any files, you will be asked if you want to re-analyze the changed files before running CodeCheck.

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 and the hamburger menu to change how the results are displayed. You do not need to re-run the checks to change the display. The Violation Browser also provides handy ways to search results. See *Viewing Results on page 296* 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 305*. 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 or not the CodeCheck window is contained within the *Understand* window. See *Viewing Results on page 296* and *Ignoring or Excluding Violations on page 305* for details about other hamburger menu commands.
- 7 To export results from CodeCheck, see *Exporting CodeCheck Results on page 310*.

Re-Analyzing the Project

If any source files to be inspected have been edited but not yet saved or not yet analyzed, you will be prompted to choose whether to reanalyze the project.




- **Re-Analyze files and run checks:** The analysis can be performed only with CodeCheck closed. You are asked if you want to close CodeCheck. If you click **Yes**, CodeCheck is closed, any modified files are automatically saved, the project analysis is run, then CodeCheck is reopened, and the selected checks are run on the new project analysis.
- **Ignore modifications and run checks:** The checks are run on the project as it was last analyzed. Any changes you have made to the files since the last analysis are ignored, whether or not the files have been saved.
- **Cancel:** No checks are run.

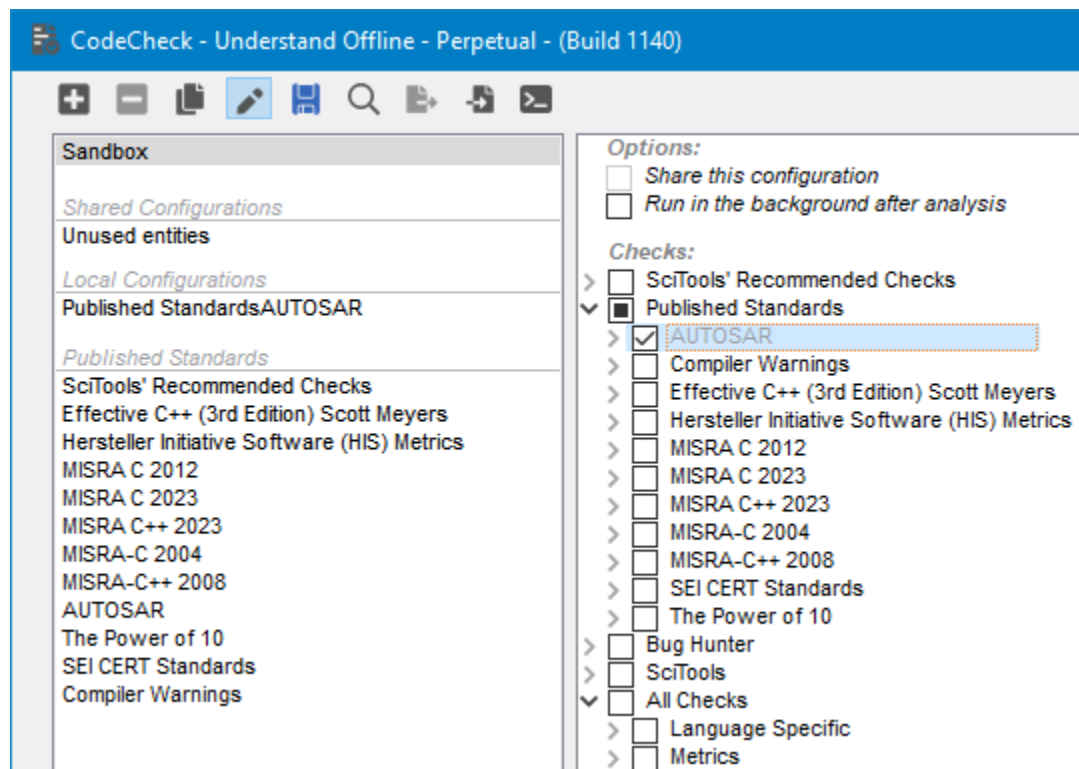
Configuring Checks

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.
- AUTOSAR (C++14): >90% of statically verifiable AUTOSAR rules are covered.
- SEI Cert Standards
- Compiler Warnings
- Effective C++ (3rd Edition) Scott Meyers
- Hersteller Initiative Software (HIS) Metrics
- MISRA-C 2004, 2012, 2023
- MISRA-C++ 2008, 2023
- The Power of 10 (developed by NASA and JPL)
- Bug Hunter checks (C/C++ only. Looks for more complex code problems such as memory leaks, access to stack memory that escapes the function, access to pointers that have been freed, and use of uninitialized values.)
- language-specific checks
- metrics for program units, classes, and packages

Note: If you select a Bug Hunter check, be aware that running Bug Hunter typically uses about 1 GB of disk space per 100,000 lines of code. This space is needed on the drive where your project is stored (*Project Storage on page 35*) in a subfolder named “ast”. You may delete this folder after you have finished using the results of the Bug Hunter checks, but this will slow down Bug Hunter the next time you run it. Running Bug Hunter on a large project takes significant time.

To configure sets of checks that meet your specific needs, click the  wrench icon or choose **Checks > Select Checks** 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. If a check has a checkmark next to it that cannot be turned off, this check is run automatically. If a check does not have any box or checkmark next to it, the check cannot be automated. Any custom checks you have installed are listed in the Checks list. See *Writing CodeCheck Scripts on page 312*.

- 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 222](#) for details.
- 5 For most checks, you can set the severity of the violation to a Informational, Low, Medium, High, or Urgent. Severity is shown in the CodeCheck results and can be used as a filter in the Violation Browser.
- 6 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 control how checks are performed. For example, the AC_00 check lets you choose which 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.

Program Unit Max Length

Severity

Urgent
High
Medium
Low
Informational

Maximum Lines 200

☐ Count only Lines of Code(ignore comment lines, blank lines, etc)

Rationale

Long functions are usually complex and difficult to read meaning they are also difficult to comprehend and to test. Functions, Methods, Packages, Procedures, Subroutines etc. should not be longer than the specified number of lines.


CheckID


METRIC_04

- 7 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.

unused

- 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.

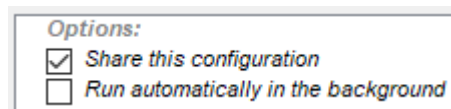
To resume editing a saved configuration, click the  edit icon. You can also right-click on the name of the configuration and choose **Edit** or **Rename** or double-click the configuration name.

To delete a configuration, click the  delete icon. You can also right-click on the name of the configuration and choose **Delete**.

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".

Running Configurations in the Background

When you edit a CodeCheck configuration, you can check the **Run automatically in the background** box.




If this box is checked, the configuration runs automatically each time the project is analyzed, and the results are shown in the Violation Browser and in the editor scrollbar. Unchecking the box does not clear existing results from these locations; results remain until the next project analysis or the next time CodeCheck inspection is run.

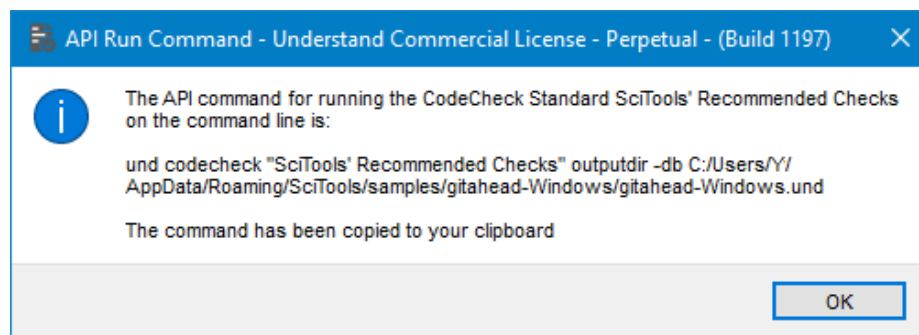
While CodeCheck is running an inspection in the background, you cannot modify the project configuration or analyze the project.

When you open an existing project, you may be prompted to allow recommended CodeCheck checks to run in the background each time you analyze the project. These checks help ensure code quality on an ongoing basis.

See *Analyzing the Code* on [page 102](#) for information about monitoring background processing.

Running Configurations on the Command Line

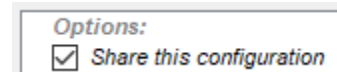
To get the `und` command to run the currently selected checks, click the  command line icon. A dialog opens that shows the command and the command shown is copied to your clipboard. See *Using the und Command Line* on [page 347](#) for more information.





Sharing Configurations

You can share CodeCheck configurations either by making them sharable or by exporting a configuration so that others can import the configuration from a file.

If you want other users of this project to be able to use this CodeCheck configuration, begin editing the configuration and check the **Share this configuration** option in the right pane 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.



To export a CodeCheck configuration to a file that can be imported, click the  **Export** icon and browse to the location where you want to save the file. Configurations are saved as JSON files. You can only export configurations that you have created (or imported); the Sandbox configuration and the published standards cannot be exported.

To import a CodeCheck configuration, click the  **Import** icon and browse to find the file that contains the configuration you want to import. If a configuration with the same name already exists in your list of configurations, a suffix, such as “-1” will be added.


Selecting Files to Check

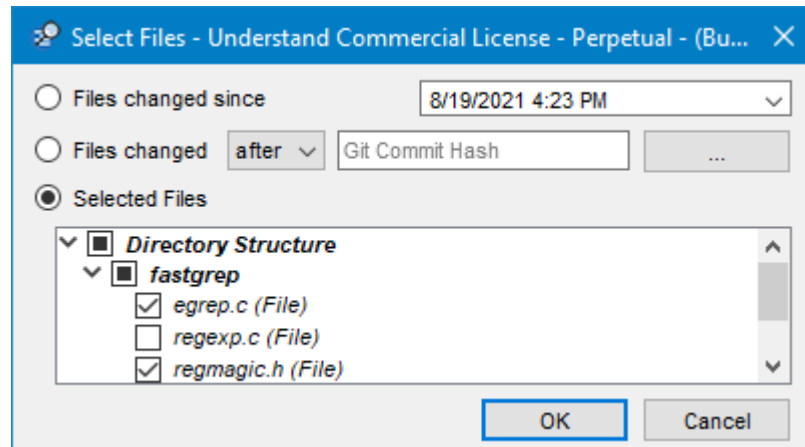
Select which files to check using the **Files** drop-down list.

If any source files you choose to inspect have been edited but not yet analyzed (including unsaved files), you will be prompted to choose whether to reanalyze the project when you run CodeCheck. See *Re-Analyzing the Project* on [page 291](#).

You can choose any of the following:

- **All Files:** All files in the project are checked.
- **Files changed since last run:** Files are checked if they have been changed since CodeCheck was run last.
- **Architecture list:** Files are checked if they are part of the selected architecture.
- **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 61](#).

- **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 in/after Git Commit:** Choose this option if you want to check files changed after a certain commit or in a certain commit. Click the ... button and select the commit you want to check (or paste the hash string for a Git commit). If you have not yet specified a Git repository, see *History Options on page 61*.
- **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 the following ways without needing to re-run the checks:

- By Files ([page 297](#))
- By Checks ([page 298](#))
- Locator ([page 299](#))
- Treemap ([page 299](#))
- Log ([page 301](#))
- Violation Browser ([page 302](#))

You can hide certain results by ignoring particular checks and violations or setting a baseline ([page 305](#)). You can also print or export results ([page 310](#)).

In general, the Results area is good for getting a high-level overview of all of the violations in the project.

The Violation Browser shows errors and warnings that are found during project analysis and CodeCheck checks that run in the background. The Violation Browser provides filtering of issues to locate, for example, errors in code that has been modified but not committed or in code that has been changed in the last few days.


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	<div> <div>C:\Users\Yvonne\AppData\Roaming\SciTools\samples\fastgrep\fastgrep\regerror.c</div> <div> <div>Unused Parameter s in Non Virtual Functions</div> <div> <div>3 void</div> <div>4 regerror(s)</div> <div>5 char *s;</div> <div>6 {</div> <div>7 #ifdef ERRAVAIL</div> </div> </div> </div>	s	5	6	CPP_
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 290](#) and *Ignoring or Excluding Violations* on [page 305](#) 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 297). However, all violations of a particular type are listed together.

Results

Entity

Line

Column

Seyerity

64 ▾ Number of Results: 645

64 ▾ SciTools' Recommended Checks

6 > Commented Out Code - RECOMMENDED_00

11 ▾ Comments Indicating Future Fixes - RECOMMENDED_19

3 ▾ E:\samplecode\zlib\deflate.c

▾ ⚠ Future work needed: key phrase 'to do' found

deflate.c

212

4

Low

210 return deflateInit2(strm, level, Z_DEFLATED, MAX_WBITS, DEF_MEM_LEVEL,

211 Z_DEFAULT_STRATEGY, version, stream_size);

212 /* To do: ignore strm->next_in if we use it as window */

213 }

> ⚠ Future work needed: key phrase 'to do' found

deflate.c

760

12

Low

> ⚠ Future work needed: key phrase 'to do' found

deflate.c

770

4

Low


1 > E:\samplecode\zlib\deflate.h

Rationale

Identifying areas of code that still need work is crucial for ensuring that software is functional, performant, reliable, maintainable, secure user-friendly. It allows developers to prioritize their efforts effectively and deliver high-quality software products.

Developer's Note

This check automatically looks for the phrases 'Fix me', 'To Do', 'TBD', as well as different variations of those phrases. Additional phrase: user is interested in flagging can be added in the above Options textbar.

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, check ID, check name, entity, and severity.

File	Violation	Line	Column	CheckID	Check Name	Entity	Severity
deflate.c	No comment wi...	1270	19	RECOMMEN...	Variables shou...	p	High
deflate.c	No comment wi...	535	13	RECOMMEN...	Variables shou...	len	High
deflate.c	No comment wi...	961	13	RECOMMEN...	Variables shou...	len	High
deflate.c	Future work ne...	212	4	RECOMMEN...	Comments Indi...	deflate.c	Low
deflate.c	Future work ne...	760	12	RECOMMEN...	Comments Indi...	deflate.c	Low
deflate.c	Future work ne...	770	4	RECOMMEN...	Comments Indi...	deflate.c	Low
zip.h	Fixed Value(8) ...	78	23	RECOMMEN...	Magic Numbers	MAX_MEM_L...	Medium

Rationale

You can type values to match any column. 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 170](#).

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.

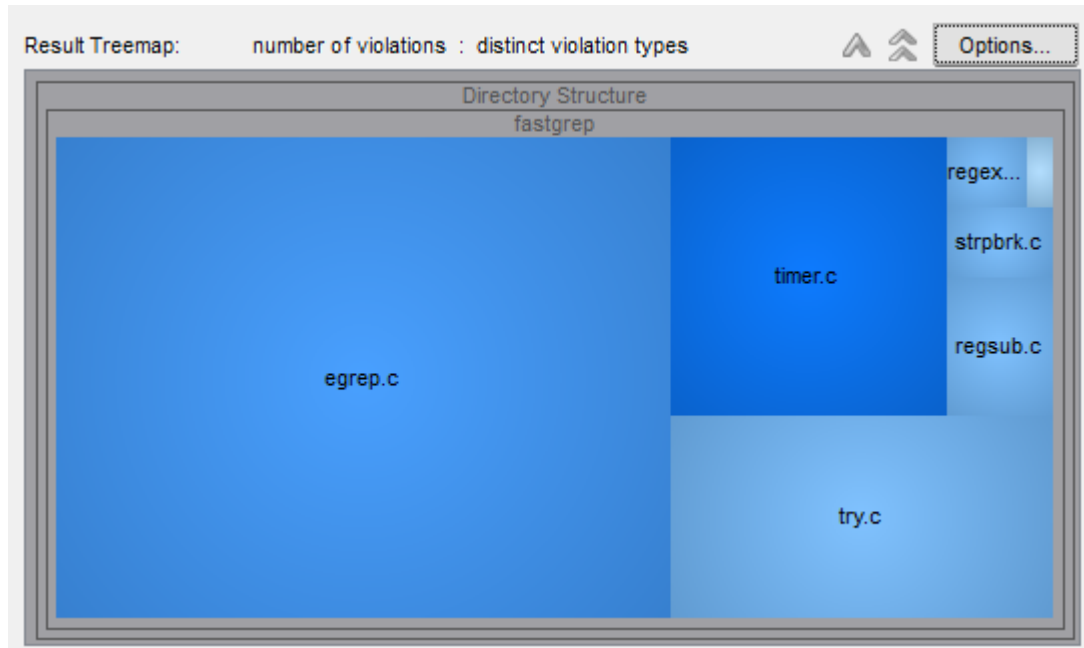
Viewing Results by Treemap

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 and compare any two of the following metrics:



- total number of violations per file
- number of distinct violation types per file
- density of violations per number of code lines per file
- maximum severity of violations per file

In this treemap, for example, 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 253](#).

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.



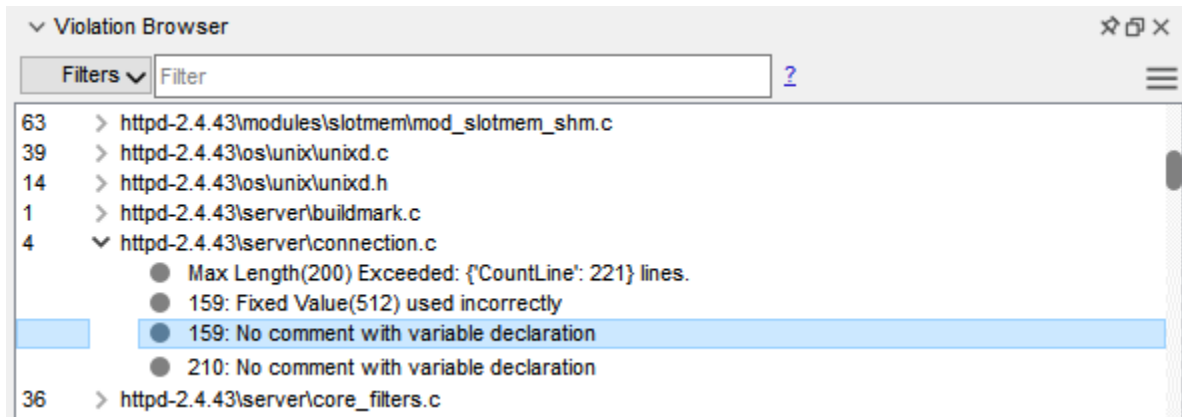
If you enabled the **Use Verbose Logging** in the  hamburger menu, the Log also includes a separate line for each violation found. This option must be selected before you click the **Inspect** button to take effect.

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.

Viewing Results in Violation Browser

The Violation Browser shows issues that were identified during the project analysis and issues found by CodeCheck checks that run in the background. For these types of issues, the Violation Browser provides fast navigation and filtering of issues to locate, for example, code that has been modified but not committed or code that has been changed in the last few days.

To open the Violation Browser, choose **Checks > Browse Violations** or **View > Violation Browser** from the menus.



Violations have a colored circle next to them to indicate the type of issue. Red circles are for errors found during project analysis. Yellow circles are for warnings found during project analysis. Gray circles are for issues identified by CodeCheck.


Click on any violation in the list to open the source file in the Source Editor and scroll to the location of the violation. In the Source Editor, lines with a violation have a caution icon in the right margin. Hover the mouse over the icon to see the violation.

```
#else
# define MOD(a) a %= BASE
# define MOD4(a) a %= BASE
#endif
```

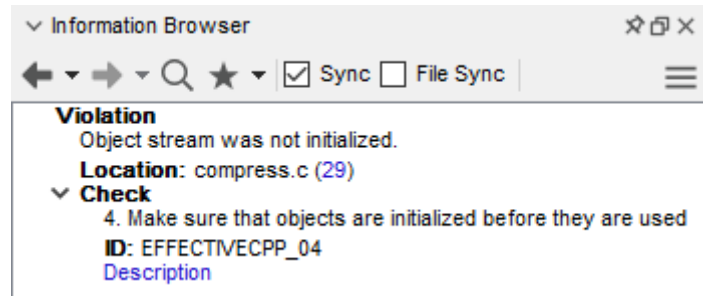
2. Do not use #define to specify types
#define used


To permanently ignore a specific violation, click the eye icon in the right margin. This opens the Ignore Issue refactor tool (see [page 307](#)). To temporarily remove a specific violation, click the “X” icon in the right margin.

- 56: Violation: tr defined but not called.
- 56: Violation: trUtf8 defined but not called.
- 70: "qt_getEnumMetaObject" defined in header file.

In the Source Editor, you can click the  **Caution** icon in the toolbar to open the Violation Browser. The up and down arrows next to the Caution icon move to previous/next issue in that source file and a pop-up describing the selected violation is shown.

If the **Sync** box is checked in the Information Browser, information about the violation is shown there. Expand the Check node to see the violation text and ID. The Severity of the violation is also shown. The **Description** link opens a page that explains the violation and gives an example in many cases.

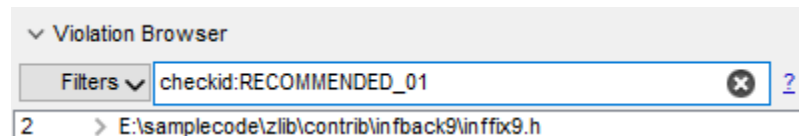


The  hamburger menu in the Violation Browser allows you to display short filenames, relative paths, or full file paths. You can also clear all contents of the Violation Browser. The **Export Analysis Errors** command allows you to send errors encountered when analyzing the project to a CSV file that contains the file path, line, column, and error message.

The **Filters** field lets you quickly narrow the list of violations to those you are interested in fixing. Type text to match anything in the text string about the violations. Use the **Filters** drop-down if you want your search to match:

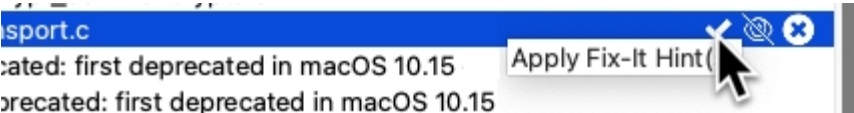
- uncommitted violations
- violations in files modified today or this week (see *History Options* on [page 61](#))
- errors encountered during project analysis
- warnings encountered during project analysis
- violations with a particular severity level

You can also filter the searches by the check ID field, the check description, line numbers, architecture nodes, and date modified. To see details about such search syntax, click the small “?” to the right of the Filters field.



Here are some examples of the capabilities of this search syntax:

Filter Example	Meaning
hello world	Matches violation text containing both “hello” and “world”.
checkid:RECOMMENDED_01	Matches an exact violation by ID.
12 in:checkid	Matches part of the violation ID string.
defined in:check	Matches short violation descriptions having to do with definitions.

Filter Example	Meaning
operator -in:violation	Excludes violations related to “operator”.
hello NOT world	Matches violation text containing “hello” but not “world”
checkid:JAVA_11,JAVA_12	Matches violations of either type JAVA_11 or JAVA_12.
arch:Team/Blue	Matches violations in code belonging to the Blue node of a Team architecture.
arch:“Directory Structure/src/parser”	Matches violations in the /src/parser node of the Directory Structure architecture.
line:20..80	Matches violations on lines between 20 and 80 of their respective files.
hint:>0	Matches violations for which a hint on how to fix the code is provided in the Source Editor issues sidebar. For such violations, you can apply the Fix-It Hint change by clicking the checkmark in the highlighted file in the Violation Browser.
	
related:>0	Matches violations with a related location, such as a beginning bracket that is not ended.
srcmodified:2022-02-01..2022-02-28	Matches violations in files modified between the specified dates (according to your operating system)
srcmodified:>v2.6.3	Matches violations in lines changed after the specified tag in Git.
srcmodified:5cd66cd8	Matches violations in lines changed in a Git commit with a hash beginning with the specified string.

Viewing Violation Metrics

On a per file or architecture node basis, the Metrics Browser shows the number of CodeCheck violations, number of violation types, and violation density metrics. The values for these metrics reflect violations of the checks selected in the most recent CodeCheck inspection. See *Metrics Browser* on [page 248](#) for more information.

Viewing SARIF Files

In addition to browsing violations identified by CodeCheck and the project analysis, you can import SARIF files into *Understand* and browse those violations. SARIF is an industry-standard format for exchanging results with other Static Analysis tools. SARIF files have a file extension of .sarif and use JSON syntax.

To import a SARIF file, drag and drop the file into *Understand*. The loaded violations can be viewed in the Violation Browser, Information Browser, and Issues sidebar.

Note that SARIF files typically contain full pathnames to files, so the files referenced in violations listed in an imported file may not be found by your project. If you want to change the file paths to match those on your drive, you can manually edit the SARIF file before importing it.

Ignoring or Excluding Violations

Violations may be excluded from the results either because they are ignored or because they are excluded:

- *Ignored violations* are added by a user and can apply to a specific violation or violations for an entity or within a file or directory. You can ignore violations using CodeCheck, a Source Editor tab, code comments, or code annotations.
- *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.

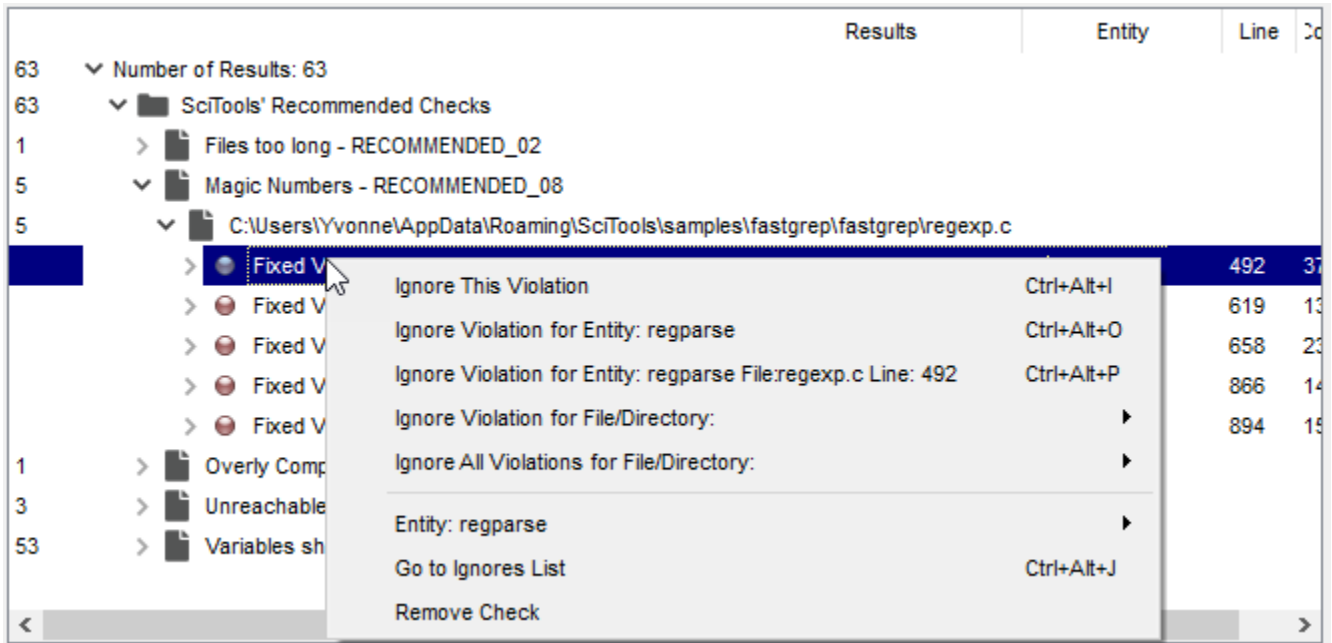
If many violations are detected, you might want to look at the checks in your configuration to see if you can set options to control the sensitivity of the checks. For example, for the “Magic Numbers” check, you can specify that bitfields can be set to 0 or 1 but other values are violations. For checks such as “Program Unit Cyclomatic Complexity”, you can set the maximum complexity that does not count as a violation.

Ignoring Violations in CodeCheck

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.

You can ignore violations of CodeCheck tests, but cannot ignore errors encountered during project analysis.



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. These notes are shown in the Note column of the Ignores List.

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 a violation is both excluded and ignored, the setting for whether to show ignored violations takes precedence over the setting for whether to show excluded violations.

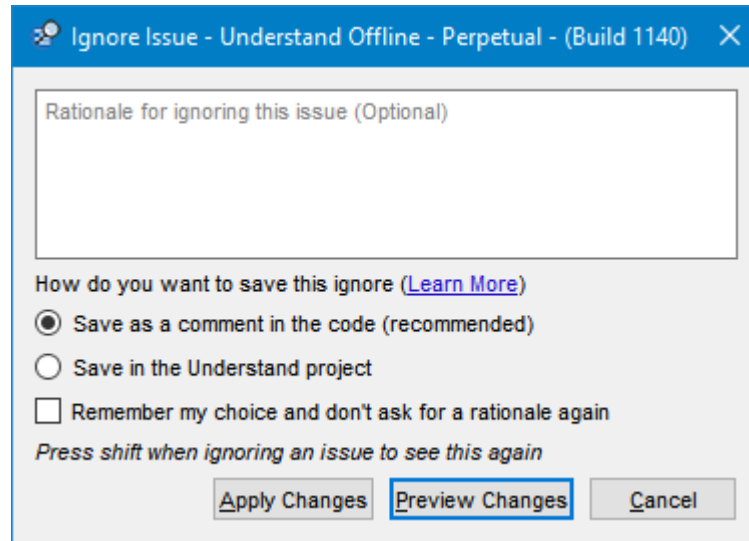
If you have ignored any violations, click the **Ignores List** button to see a list of the currently ignored violations. You can search this list as you would the Result Locator list. See *Filtering the List* on page 170 for details.

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.

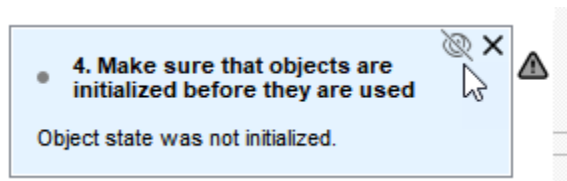
See *Running a CodeCheck* on page 290 and *Viewing Results* on page 296 for details about other hamburger menu commands.

Ignoring Violations in the Source Editor

Another way to ignore violations is to select code that contains one or more violations in the Source Editor and choose **Refactor > Ignore Violations** from the context menu.



Another way to open this dialog is to click the warning icon in the right margin of the Source Editor or the Violation Browser (see [page 302](#)) and then click the eye icon.





Type a note to explain why this violation can be ignored. The note will be shown in the Ignores List in CodeCheck.

If you choose **Save as a comment in the code**, a comment to specify an automatic ignore will be added to each line with a violation in the selected code. For example:
`/* UndCC_Line(EFFECTIVECPP_04) Jo and I discussed this, and it is OK */`
 See *Adding Automatic Ignores to Code* on [page 309](#) for syntax details. You can preview the code changes that will be made before applying them.

If you choose **Save in the Understand project**, an annotation is added to each line with a violation in the selected code. The annotation contains the same automatic ignore string as the comment option, but is only visible to other users if they share the same *Understand* project. See *Annotations* on [page 203](#) for more information.

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 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 309](#)) 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...	regex	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 or annotations 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 and annotations add a rule to the Ignores List to ignore all CodeCheck violations on this line and add the note. Use the comment syntax supported by the language 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.



The same keywords that cause automatic ignores in comments can be used in annotations. Annotations are saved as part of the project instead of as part of the code. So, they are only visible to other users who use the same copy of the project. See [Annotations on page 203](#).

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 286](#).
- **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
```

- **Compliance Report:** This set of reports helps you see how well your code complies with a standard. For example If your code needs to comply with MISRA C++2008, these reports show where your project is compliant and where it is non-compliant.

You can generate various types of reports by clicking the **Compliance Report** button in the CodeCheck. This button is active after a CodeCheck configuration has been run and violations were found. After the reports are generated, your operating systems file management tool opens to the directory containing the generated files.

The following reports are available:

- **Summary Report:** Lists all checks performed and how many violations were found and ignored. The Category column indicates whether each check is Advisory, Required, or Other according to its standard. The Compliance column indicates whether or not the code is compliant with that check.
- **Coverage Report:** Lists checks performed, including a description of each check. It indicates whether each check can be automated and whether check compliance

is required or advised by the standard. The report also indicates the percentage of checks in the standard that are supported by *Understand*.

- **Ignores Report:** For each violation ignored, this report lists the file and entity in which the violation was ignored. It also lists the Check ID, violation description, and any note provided about why the violation was ignored.
- **Violations Report:** For each violation found, this report lists the file and entity in which the violation was found. It also lists the Check ID, violation description, and severity for each violation.

The reports may be saved in CSV, HTML, and/or PDF formats. You can select a directory to contain the generated files.

Compliance Report Options - Understand Commercial License - Perpet... X

Compliance Report Options

Compliance Reports

- ☒ Summary Report
- ☒ Coverage Report
- ☒ Ignores Report
- ☒ Violations Report

Output Format

- ☒ CSV ☒ HTML ☒ PDF

Directory to Save Report

ne/AppData/Roaming/SciTools/samples **Browse**

Generate **Cancel**

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 343* for information about the Python API.

You can also use CodeCheck from the “und” command line. For details, see *Using CodeCheck on page 354*.

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 `plugins/CodeCheck` directory in your *Understand* installation and the CodeCheck subdirectory of the plugins Git repository at <https://github.com/stinb/plugins>.
- 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 syntax is correct. The easiest way to do this is to open a command line and run the application. For example, the Perl application that ships with Understand can be run with: `upperl -c mysample.upl`.

You can integrate such scripts into your development process, for example by retrieving CodeCheck violations, integrating those with your CI/CD pipeline and issue-tracking systems, triggering automated builds, and/or generating comprehensive reports.

To learn more, see [APIs on page 343](#). Browsing the CodeCheck scripts that are shipped with Understand can also be beneficial.

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**. Then choose **Tools > Clear Script Cache** from the menus or restart *Understand*.

When you install a custom check, you will see a message that identifies the directory where the check was installed. For example,

C:\Users\YourName\AppData\Roaming\SciTools\plugin\Codecheck. You can install future checks by copying files directly to this directory. This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category* on [page 109](#)).

This chapter explains the history and source-code comparison features provided by *Understand*.

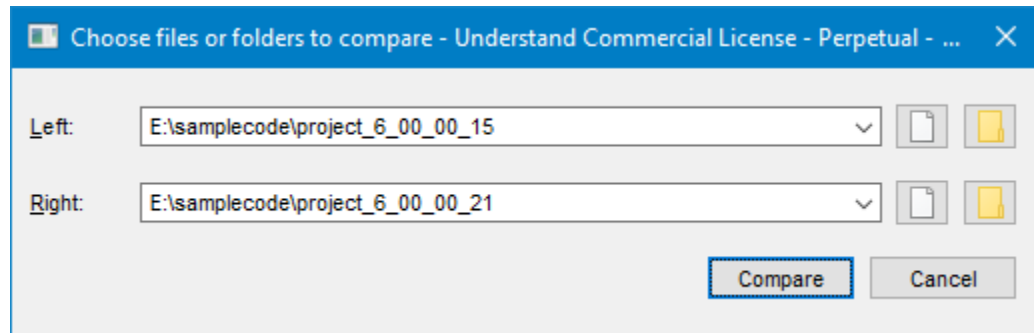
This chapter contains the following sections:

Section	Page
Comparing Files and Folders	316
Comparing Entities	319
Comparing Text	320
Comparing Projects	320
Comparison Graphs	323
Exploring Git History	326
Exploring Differences	328

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 in *Comparing Projects* on [page 320](#).

To open this tool, choose **Compare > Compare Files/Folders** from the menus.



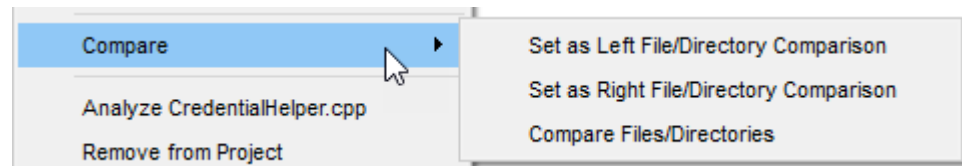
In this dialog, select a file or folder for both the left and right comparison. Both sides should be similar files or similar folders. Click the file button to browse for a file; click the folder button to browse for a directory. Then click **Compare**.




Subdirectories of the directories you choose are also compared.

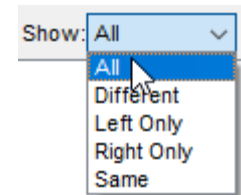
The code comparison section of the results window is described in *Exploring Differences* on [page 328](#). If you are comparing folders, there is an additional bottom pane that shows the folder-level and file-level differences and lets you filter the contents you want to compare.

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 \neq sign to indicate whether the contents have been added, deleted, or changed.

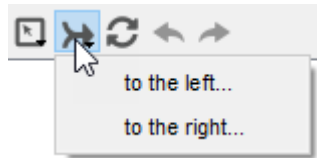
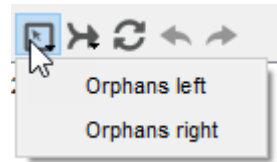
By default, all files and folders are listed. You can use the **Show** drop-down to choose whether to restrict the list to showing only:







- **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.

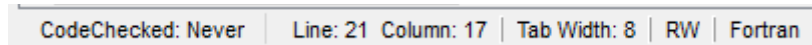
You can highlight all items that exist in only the left or right version. To do this, first right-click on the file list and choose **Expand All**. Then click the  Select icon and choose either **Orphans left** or **Orphans right**. You will see a warning that some items may have been skipped; this applies 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.

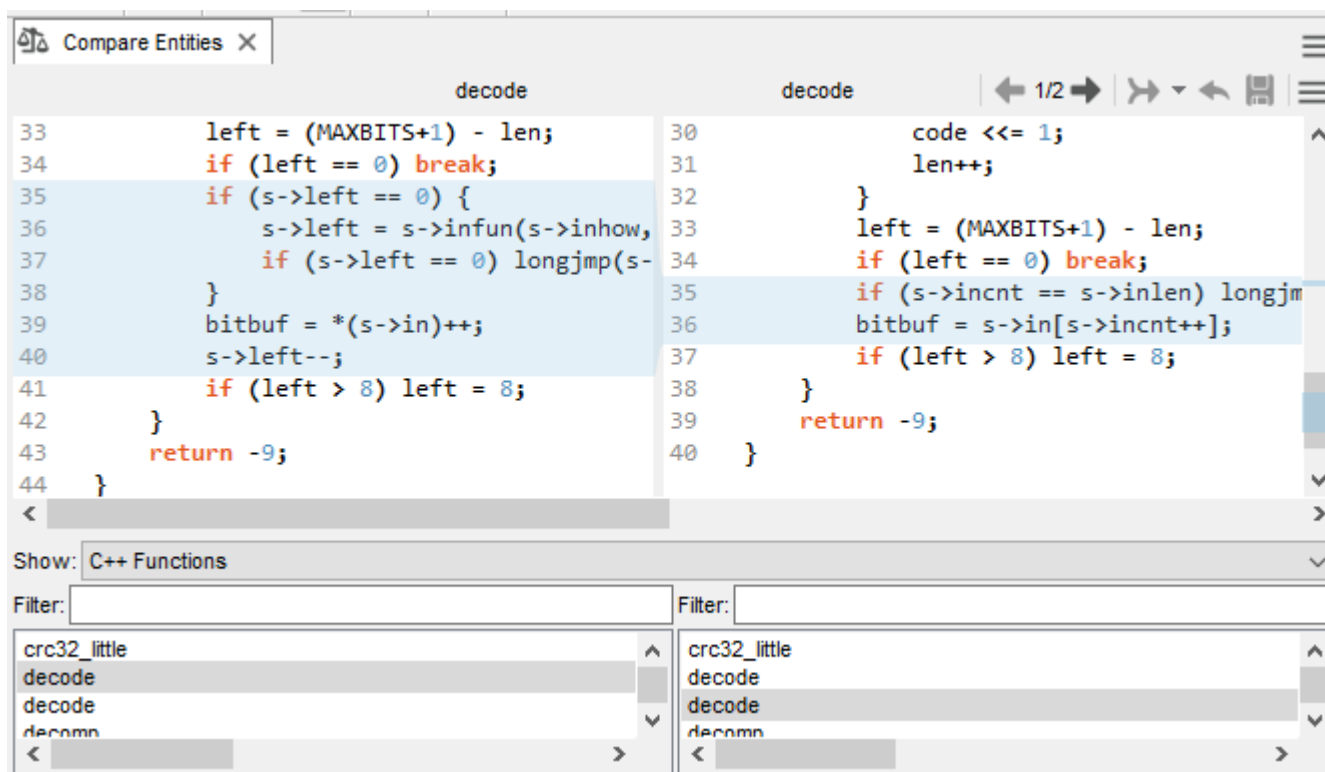


The code comparison section of the results window is described in more detail on [page 328](#).

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 top portion is similar to other types of comparisons ([page 328](#)).

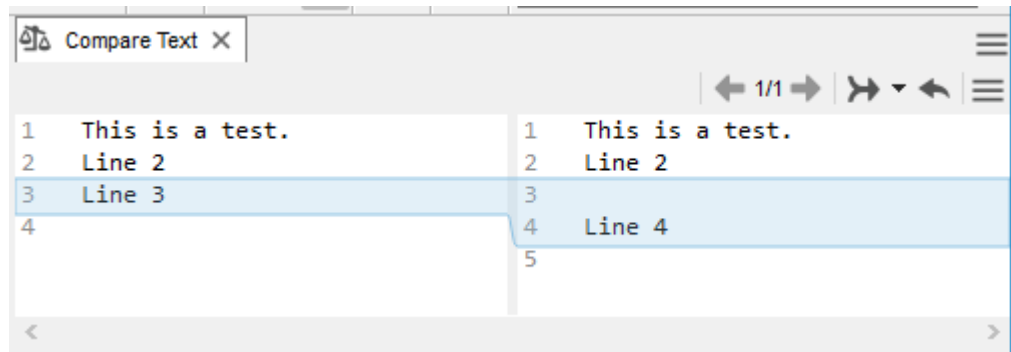
Below the comparison is an entity filter ([page 137](#)). Select a type of entity in the **Show** drop-down. Then use the lists to select two entities you want to compare. The **Filter** fields let you type a string you want to match anywhere in the entity name. Filtering occurs as you type. Wildcards 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 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

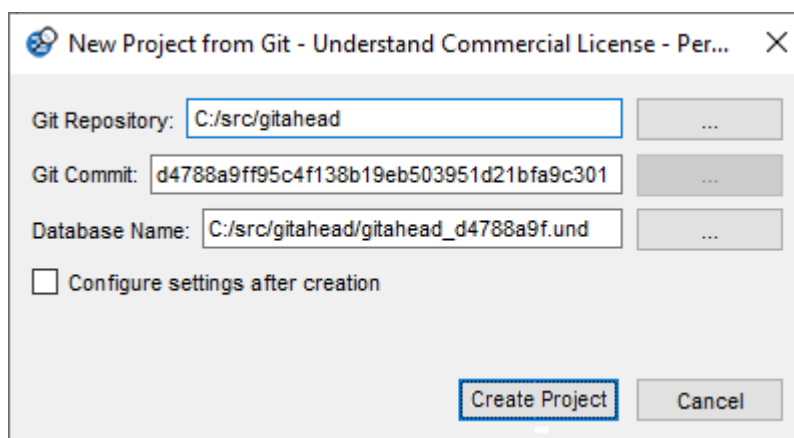
You can compare older and newer (or variant) versions of the same project if you have complete, separate copies of both versions of the project or if you are using Git to manage your source code.



To compare different versions of a project, follow these steps:

- 1 Open the newer version of the project.
- 2 Choose **Compare > Comparison 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**. In the **Git Commit** field, click the ... button and select the comment for the commit you want to compare to the current version (or paste the hash string for a Git commit). 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 46](#). 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**.
 - **Select from previously compared projects:** The Comparison Projects area lists projects you have previously compared. Hover your mouse cursor over a project filename to see the location where the project is stored. You can double-click a project to open it for comparison.
- 4 You may be prompted to analyze the new or existing database. Click **OK** and wait for the analysis to be completed. 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 Changed Entities** from the menus or click **Locate Changed Entities** at the bottom of the Comparison Projects area. (This may take some time for large projects.)

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 328*. Use of the Changed Entities Locator for filtering and sorting is similar to use of the *Entity Locator on page 168*. You can control whether changes to whitespace, line endings, and comments are counted as changes to entities in the **History** category of the Project Configuration dialog (see *History Options on page 61*).

- 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.

Changed Entities: 3418 of 3418

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%

Click the plus icon in the toolbar to add columns such as the architecture path, the date the file was modified, and differences between old and new values of metrics. If a metric you want comparison values for does not have a checkbox, select it from the list of **Additional Metrics** and click Add. You can sort and filter on any of these columns.

Select Columns - Understand Commercial License - ...

Columns

<input checked="" type="checkbox"/> Status	<input checked="" type="checkbox"/> Entity
<input checked="" type="checkbox"/> Kind	<input checked="" type="checkbox"/> Declared In
<input type="checkbox"/> Declared In Kind	<input type="checkbox"/> File
<input type="checkbox"/> Architecture	<input type="checkbox"/> Date Modified


Metrics Columns

<input checked="" type="checkbox"/> New Lines	<input checked="" type="checkbox"/> Change in Lines
<input checked="" type="checkbox"/> Removed Lines	<input type="checkbox"/> Change in Code Lines
<input checked="" type="checkbox"/> Changed Lines	<input type="checkbox"/> Change in Cyclomatic Complexity
<input checked="" type="checkbox"/> Percent Changed	<input type="checkbox"/> Change in Paths

Additional Metrics:

All Methods

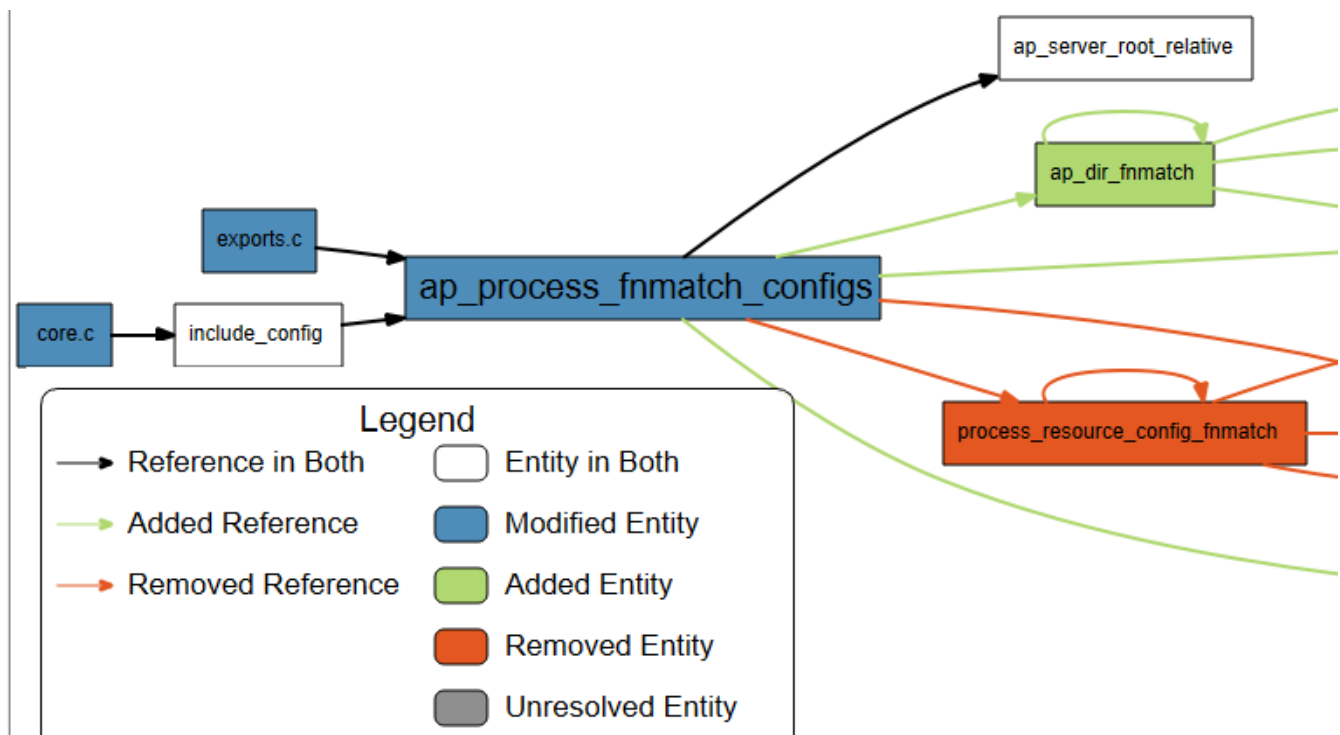
Once you set a comparison project, you can display Compare Butterfly and Compare Control Flow graphs (page 323). Comparison metrics become available in the Metrics Browser and other tools that list project metrics after you set a comparison project.

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 320 and run **Locate Change Entities**.
- 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 130).

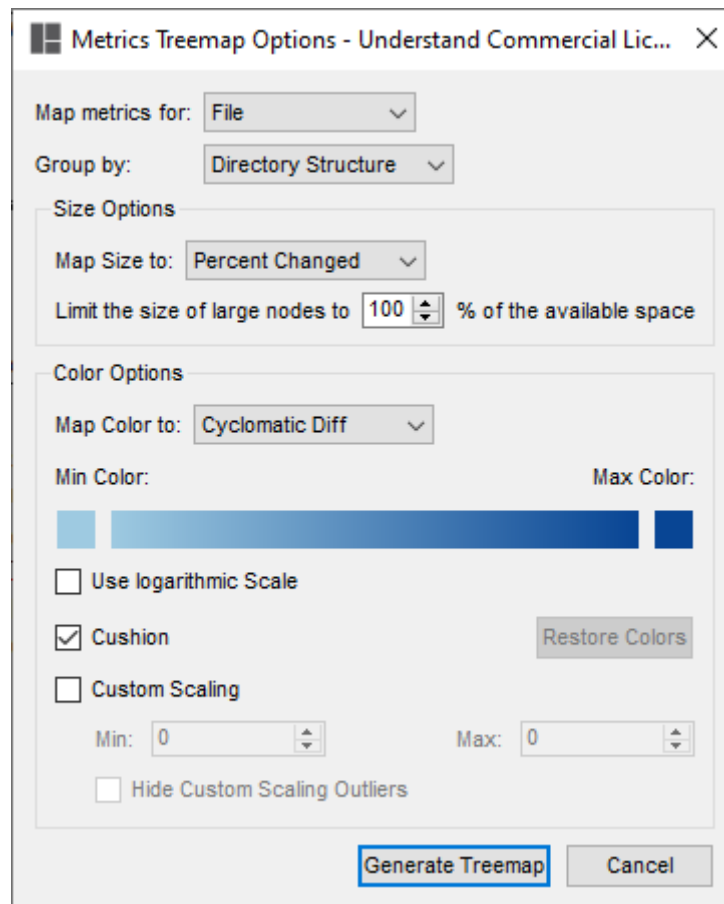


- 4 Click any node in the graph to jump to the changes made between the two versions of the project in the file differences panes below the graph.

Comparison Treemaps

You can also create a treemap that shows the relationship between two statistics about the changes to files, classes, or functions in the projects you are comparing. To generate a treemap, follow these steps:

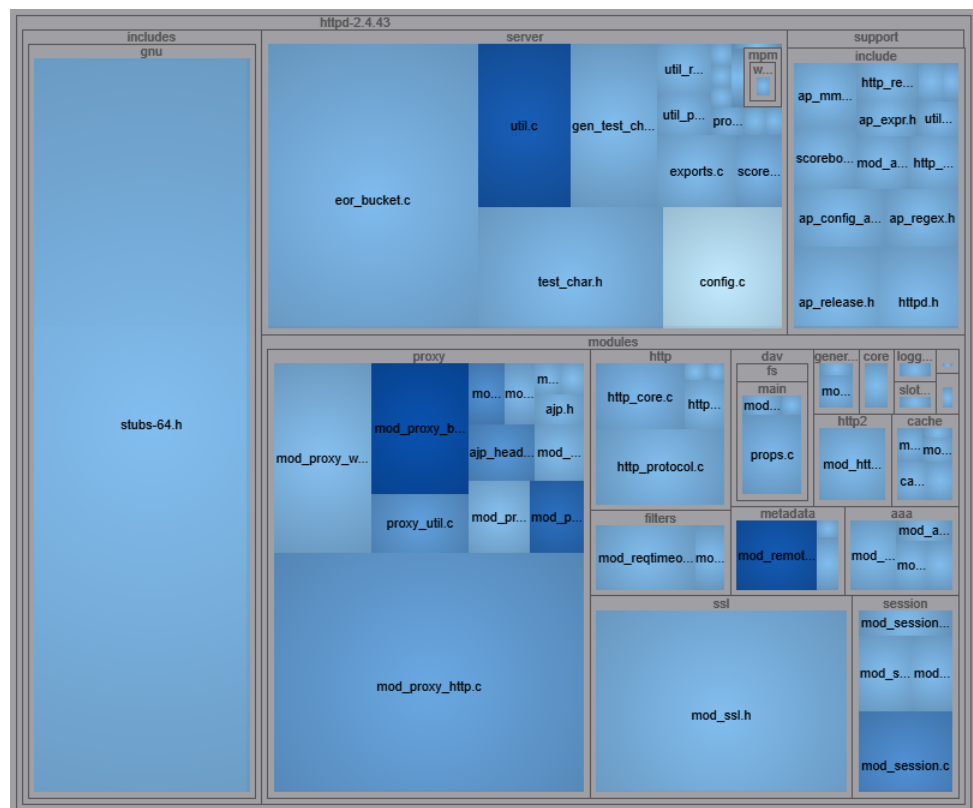
- 1 Specify a comparison project as described in *Comparing Projects* on [page 320](#) and run **Locate Change Entities**.
- 2 Choose **Compare > Changes Treemap** from the menus. This opens the Metrics Treemap Options dialog, which is described in more detail in *Metrics Treemap* on [page 253](#).



- 3 In the **Map metrics for** field, choose whether the blocks in the treemap should be for files, classes and structs, or functions.
- 4 In the **Group by** field, choose the architecture to group the blocks by. The default is the directory structure.
- 5 In the **Size Options** area, choose the comparison statistic to map to the size of each block. Options are the Percent Changed, number of new, removed, or changed lines, or the difference in the CountLine, CountLineCode, CountPath, or Cyclomatic complexity metrics. If some of your files or functions are very large compared to the rest of the project, limiting the size of large nodes makes the treemap easier to view. Your settings in the **History** category of the Project Configuration dialog (see

page 61) control whether changes to whitespace, line endings, and comments are counted as changes.

- 6 In the **Color Options** area, set **Map Color** to the comparison statistic to map to the color of each block. 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.
- 7 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.
- 8 Uncheck the **Cushion** box if you want to see solid colors in the blocks. By default, the blocks have a gradient fill.
- 9 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.
- 10 Click **Generate Treemap** to display the treemap. (This may take some time for large projects.) You can return to the Options dialog by clicking **Options** in the upper-right corner of the treemap.



- 11** Clicking on a file node in the comparison treemap opens a differences view that shows the differences in that file. See *Comparing Files and Folders* on [page 316](#).



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 61](#)).

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 **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.


 In addition, in a Source Editor window, you can use the **Toolbar Sections** category in the  hamburger menu to show or hide the **Version Control** toolbar icons, which show/hide the blame column in the Source Editor margin and open the uncommitted changes differences view.

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_leaked_mocks, 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 iffs in their explicit form ::kWarningVerbosity,
41 | "Controls how verbose Google Mock's output is."
42 | "Due to confusion arisen from "iff" standing for "if and
43 | "only if",
44 | "this commit uses the latter."
45 | "info - prints all messages.\n"
46 | "warning - prints warnings and errors.\n"
47 | "error - prints errors only.");
47 A ▾ GMOCK_DEFINE_int32_(default_mock_behavior, 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.

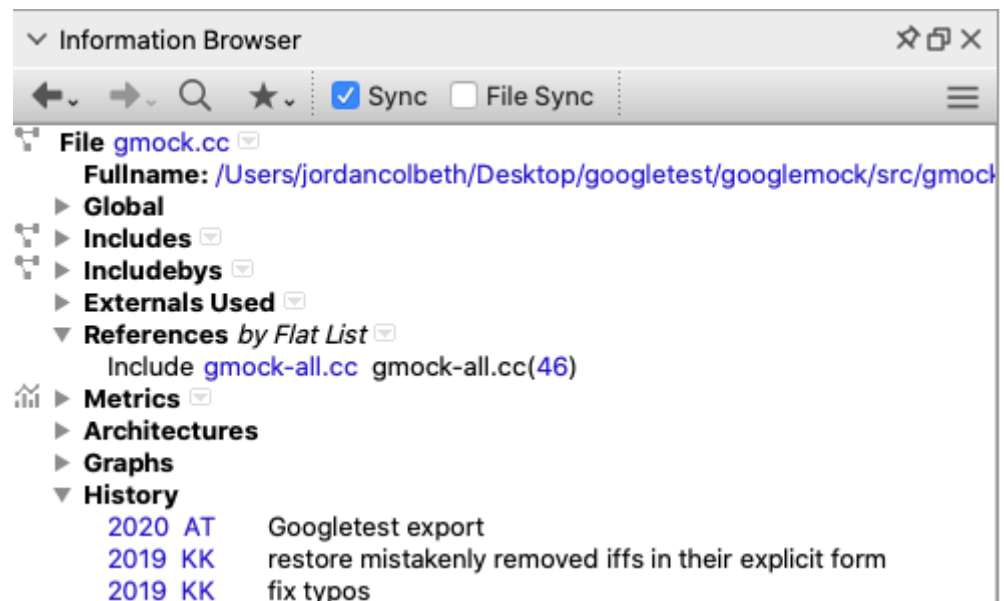
When inline blame information is enabled, commit information is shown inline to the right of code lines. Only Git commit information for the currently selected line is shown. Hover your mouse over the line to see more of the commit message if it is long.

```


154 | // Do we see a Mock flag?
155 | ▾ if (MockBoolFlag(arg, "catch_leaked_mocks",
156 |      &MOCK_FLAG(catch_leaked_mocks)) ||
157 |      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 | ▾ // 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.

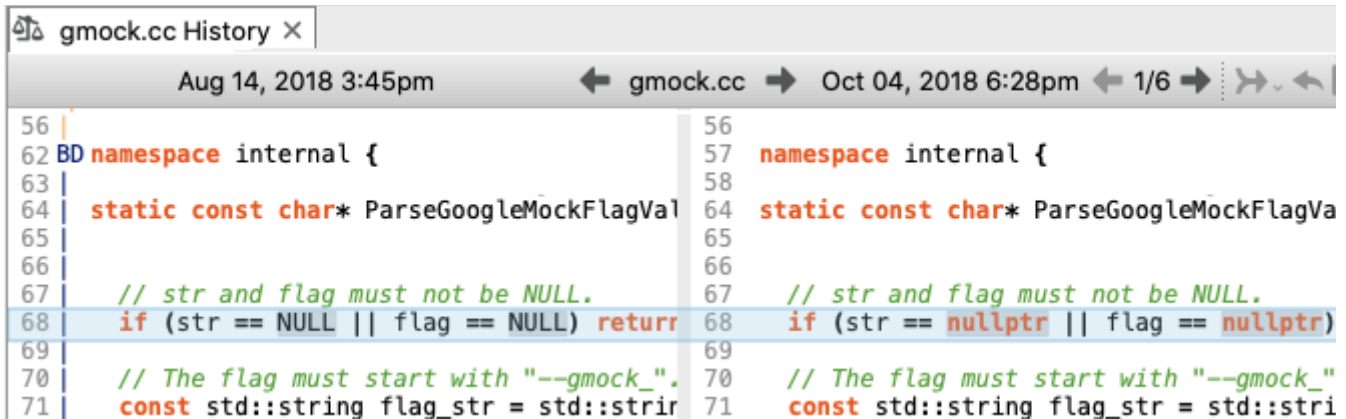


You can open a Commit History view that is similar to other comparison views by:

- 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.



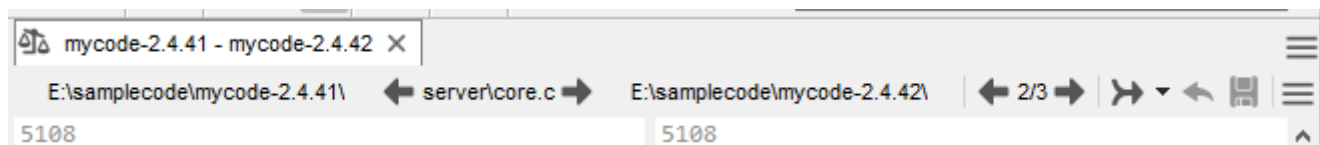
```

56 |
62 | namespace internal {
63 |
64 | static const char* ParseGoogleMockFlagVa
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::stri

```

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.







```

5108

```

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  server\core.c  arrows around the filepath.



If multiple differences are found, you can navigate to the next or previous difference using the  2/3  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 183* 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**. You can also choose **Merge Selected to the Left** or **Merge All to the Left**. 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.

Running Tools and External Commands

This chapter will show you how to configure and use source code editors and other external tools from within *Understand*.

This chapter contains the following sections:

Section	Page
Configuring Tools	331
Adding Tools to the Context Menus	338
Adding Tools to the Tools Menu	339
Adding Tools to the Toolbar	340
Importing and Exporting Tool Commands	341
Running External Commands	342
Installing and Running Plugins	343

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 333](#) 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 333](#) 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 341](#).

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.

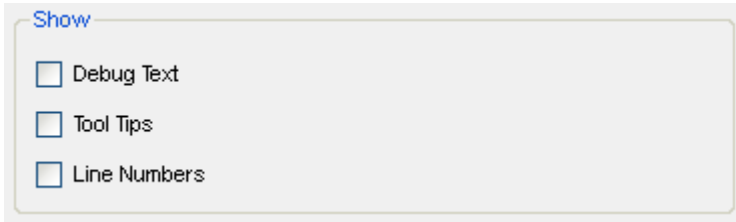
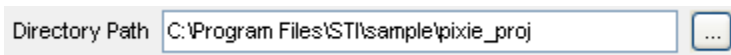

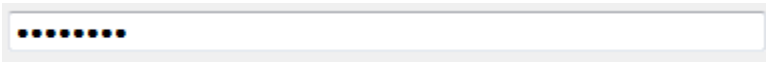

Variable	Description
\$CurScopeEntity	Scope of current entity.
\$CurSelection	Currently selected text in the current window (file windows only).
\$CurWord	The word/text at the current cursor position in the current file window.
\$DeclCol	Column in which the selected entity was declared, defaults to 1.
\$DeclFile	Full path name of the file in which the selected entity was declared.
\$DeclFileShortName	Filename only of the file in which the selected entity was declared.
\$DeclLine	Line in which the selected entity was declared, defaults to 1.
\$DeclScopeEntity	Name of the entity within which the selected entity is declared.
\$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, <code>\$PromptForCheckBox "Show Debug Text"</code> displays the following prompt:

☐ Show Debug Text

\$PromptForCheckBoxGH	Prompts user with a series of checkboxes displayed in a horizontal group. For example, <code>\$PromptForCheckBoxGH "Show=Debug Text;Tool Tips;Line Numbers"</code> displays the following prompt. The label ("Show" in this example) is optional. A semicolon must be used to separate items. The text strings for all checked items (separated by spaces) are passed to the command.
-----------------------	---

Show

☐ Debug Text ☐ Tool Tips ☐ Line Numbers

Variable	Description
\$PromptForCheckBoxGV	<p>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.</p> 
\$PromptForDir	<p>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:</p> 
\$PromptForFile	<p>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:</p> 
\$PromptForPassword	<p>Prompts user for a password. Characters typed in this field are obscured.</p> 
\$PromptForRadioBoxGH	<p>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.</p> 

Variable	Description
\$PromptForRadioBoxGV	<p>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.</p> <div><div>Format</div><div><div><input checked="" type="radio"/> PNG</div><div><input type="radio"/> BMP</div><div><input type="radio"/> GIF</div><div><input type="radio"/> JPEG</div></div></div>
\$PromptForSelect	<p>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.</p> <div><div>Build Version</div><div><div>Debug</div><div>Debug</div><div>Release</div><div>Optimized</div></div></div>
\$PromptForSelectEdit	<p>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.</p>
\$PromptForText	<p>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.</p> <div><div>Replace</div><div>foo</div></div>

In general, the multiple-selection `$Prompt` variables accept strings of the format `"label=item1;item2"`. Any number of items may be separated by semicolons. The item strings for all selected items (separated by spaces) are passed to the command.

The label is optional except in the cases of `$PromptForCheckBox`, `$PromptForDir`, `$PromptForFile`, and `$PromptForText`. The default value is optional in the cases of `$PromptForDir`, `$PromptForFile`, and `$PromptForText`.

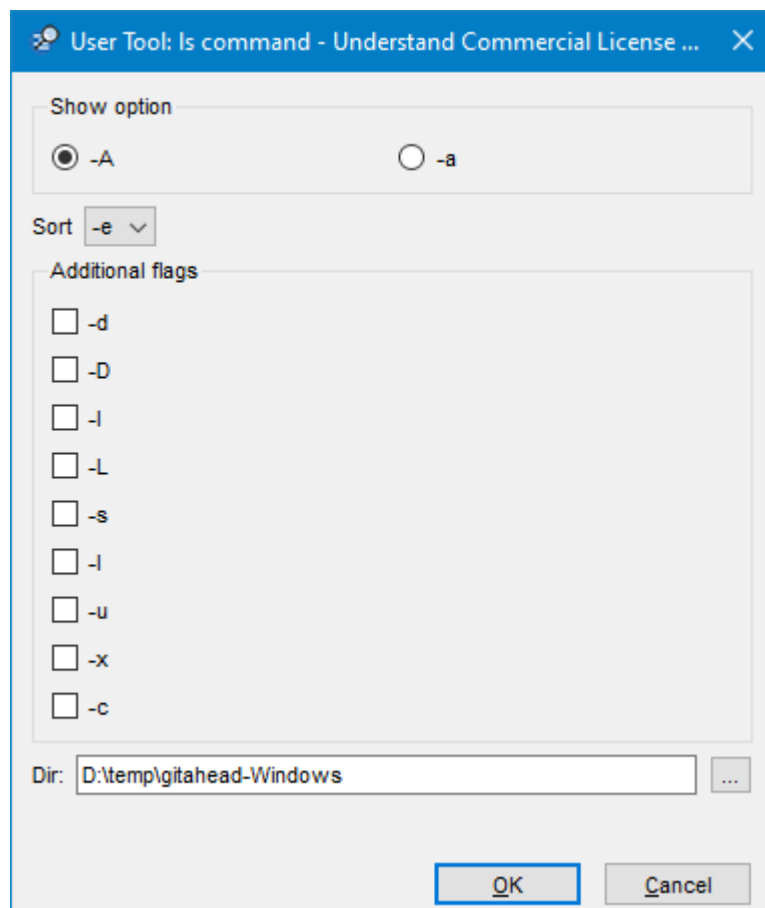
Prompts are processed after the other types of variables, so you can use other variables in the labels and values. For examples, see `$PromptForDir` and `$PromptForFile` in the previous table.

In addition, operating system environment variables can be used in prompt syntax. For example, `$PromptForSelect "Dir=$PATH"` presents a drop-down list of all the directory paths in your `$PATH` definition.

You can optionally provide the item list in a separate file. In that case, the syntax for most `$Prompt` variables is `label=@fullpath_of_listfile.txt`.

You can combine variables to pass all the parameters needed by a command. All prompts are combined into one dialog. For example, if the command is "ls", you can use the following parameters to create a dialog that lets you select command-line options for the ls command:

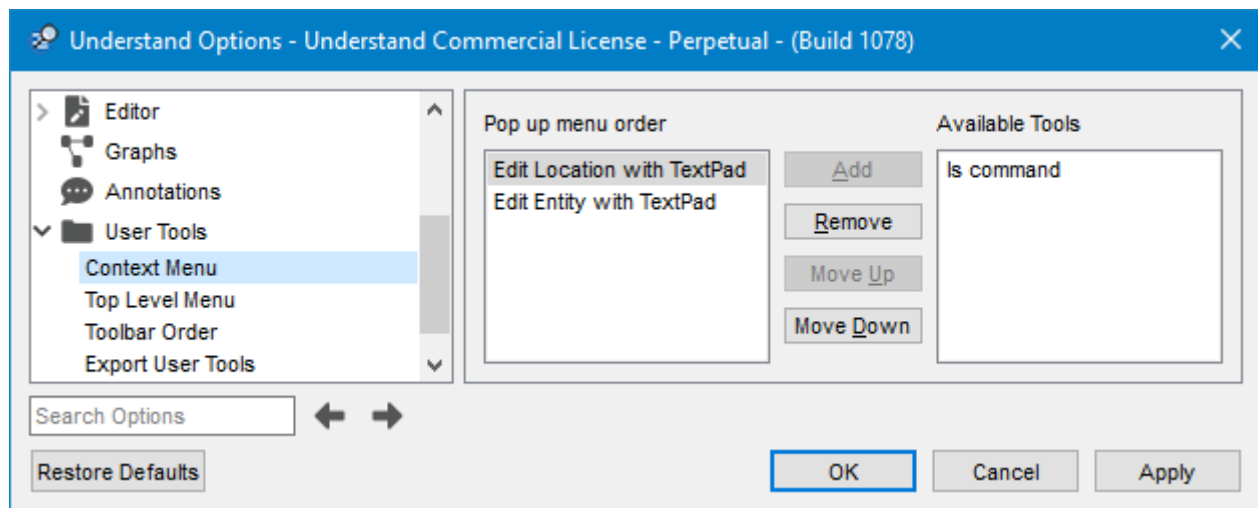
```
$PromptForRadioButtonGH "Show option=-A;-a" $PromptForSelect "Sort=-e;-t" $PromptForCheckBoxGV "Additional flags=-d;-D;-l;-L;-s;-l;-u;-x;-c"
$PromptForDir "Dir:=$CurProjectDir"
```



Adding Tools to the Context 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.)

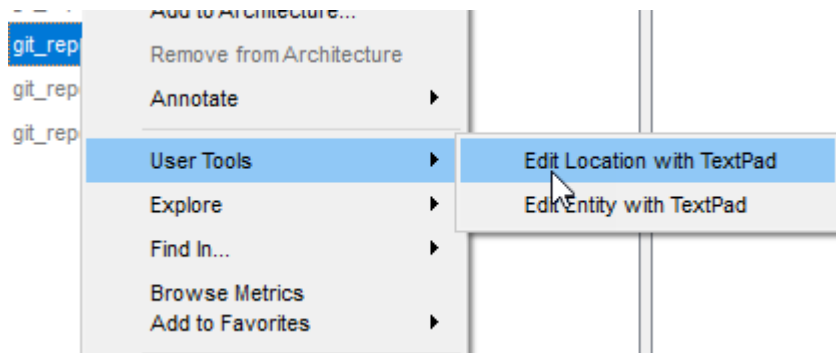
Choose **Tools > User Tools > Configure** and then select the **User Tools > Context Menu** category.



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.

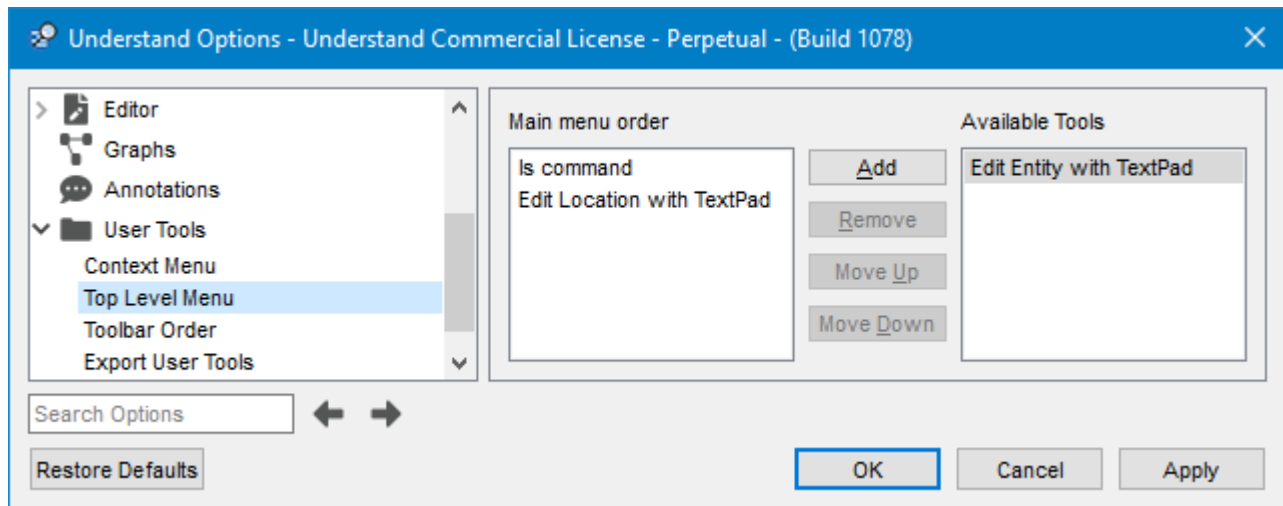


Tools are active or inactive on the context menu based on the context of the parameters provided to the tool. For example, a source editor that specifies \$DeclFile as a parameter is selectable from the context menu for any entity where the declaration is known, but will not be active for an undeclared entity or when no entity is selected.

Adding Tools to the Tools Menu

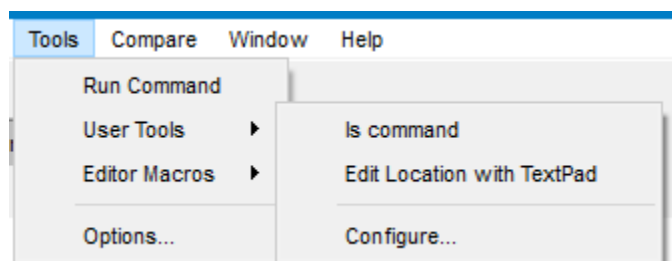
Once a command is defined in the Tools tab, the **Top Level Menu** category in the Options dialog lists user tools that are currently in the **Tools > User Tools** menu on the left and commands you can add to that menu on the right.

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**.

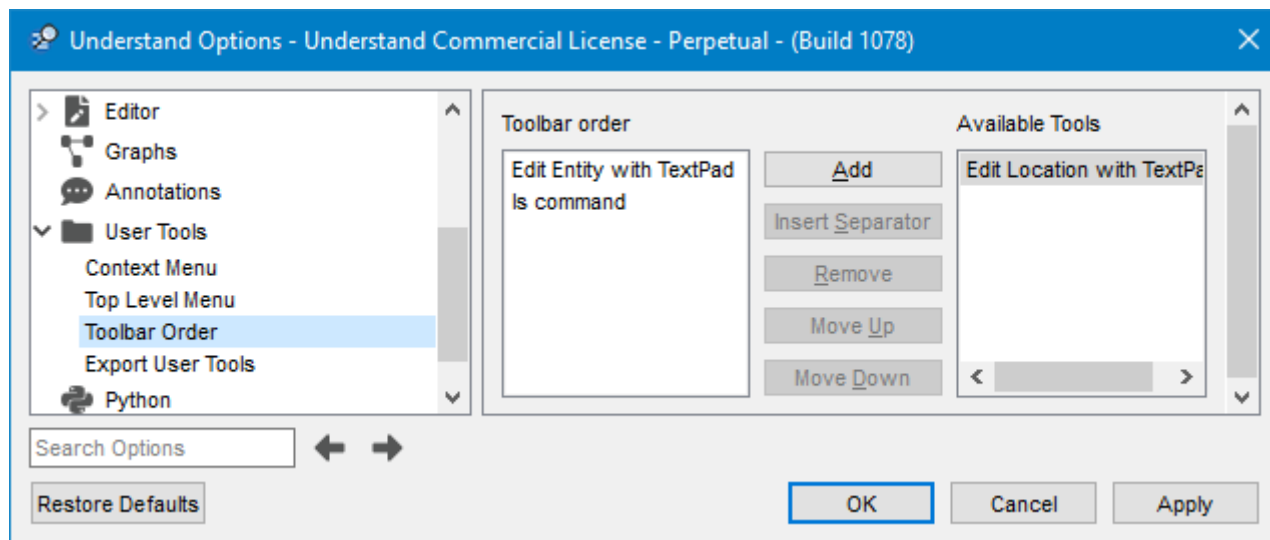
User tools appear on the **Tools** menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.



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.

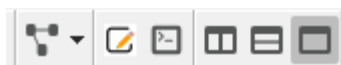


To add a tool to the toolbar, select it on the right and click **Add**. To remove it from the toolbar, select it on the left and click **Remove**.

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.

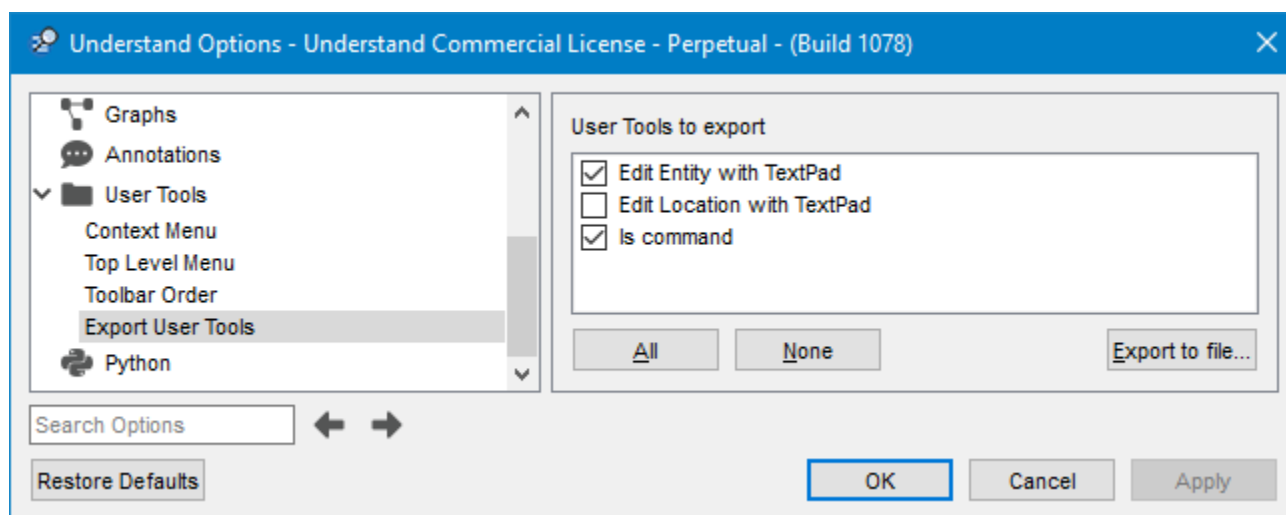


Note: You can control which icons are visible in the main toolbar by right-clicking on the background of the toolbar and checking or unchecking items for the various toolbar sections.

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.



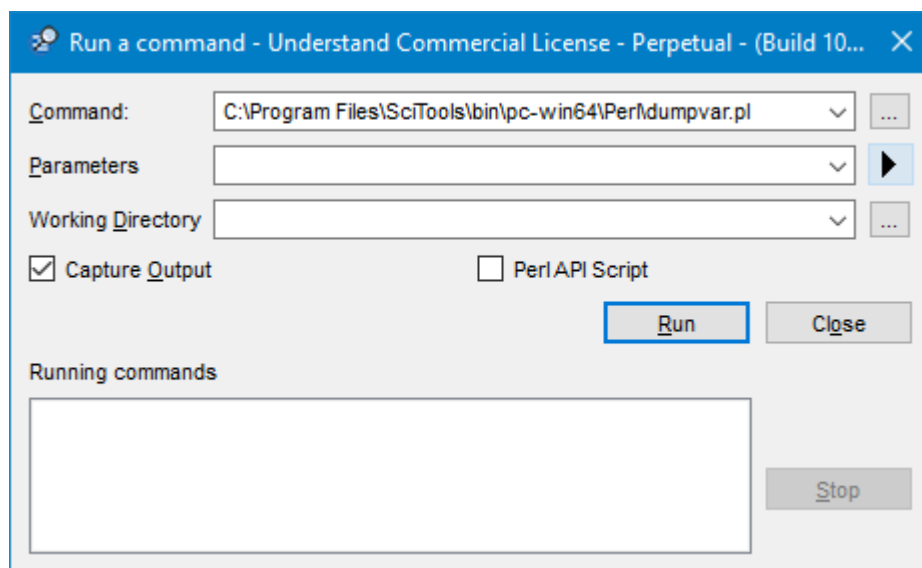
- 2 Check the boxes next to commands you want to share.
- 3 Click **Export to file**.
- 4 Choose a location and filename for an initialization file (*.ini) that contains the selected user tool information.
- 5 Click **Save**.

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 333](#).
- 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 109](#).

On Unix systems, output to both stdout and stderr are captured.

Installing and Running Plugins

Plugins are scripts that interact directly with *Understand* using an *Understand* API. Plugins come in several types, and can reference the entire project or a specific entity. (See *Writing CodeCheck Scripts* on [page 312](#) for another type of script that uses APIs.)

Examples, templates, and documentation are available in the `plugins` directory of your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>.

To install a plugin, drag the file to the *Understand* window. For example, a plugin that uses the Perl API would be a `.upl` file, and a plugin that uses the Python API would be a `.upy` file. You can also copy the plugin file to the appropriate subdirectory for the type of plugin (Arch, CodeCheck, Graph, IReport, or Metric) within the following location:

- **Windows:** `C:\Program Files\SciTools\conf\plugin\User` or `C:\Users\<user>\AppData\Roaming\SciTools\plugin`
- **Linux:** `/home/<user>/.config/SciTools/plugin`
- **MacOS:** `/Users/<user>/Library/Application Support/SciTools/plugin`

This location can be changed using the **Settings Folder** option in the General category of the Options dialog (see *General Category* on [page 109](#)).

Then choose **Tools > Clear Script Cache** from the menus or restart *Understand* to see your plugin within *Understand*.

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 347](#).

Plugin and API examples, templates, and documentation are available in the `plugins` directory in your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>.

The Python and Perl APIs are the most robust. If you are writing your own graph and interactive report plugins, using either the Perl or Python API is recommended. The following APIs are available:

- **Python API:** This API uses Python 3. It provides full access to the *Understand* database from Python scripts. Several examples ship with *Understand* in the scripts folder. Choose **Help > Python API Documentation** for documentation. *Understand* uses Python v3.12.
- **Perl API:** Many sample scripts for the Perl API ship with *Understand* in the scripts folder. Search our Support website for “Perl”. *Understand* uses Perl v5.34.0.
- **C API:** This API lets you access data *Understand* captures inside your C programs.
- **Java API:** This API is more lightweight; it does not have the full capabilities of the other APIs. See the `doc/manuals/java` subdirectory of the *Understand* installation.

- **.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

Examples, templates, and documentation are available in the `plugins/IReport` directory in your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>.

To run a project-level report, choose **Project > Interactive Reports > *plugin_name*** from the menus.

To run an entity-specific or architecture-node-specific report, right-click on the name of the entity almost anywhere in *Understand* and choose **Interactive Reports > *plugin_name*** from the context menu.

If programmed to do so, an interactive report may prompt you to enter values and can validate your responses. If you use an interactive report multiple times, the responses you provided most recently are used as the defaults.

See *Importing Report Plugins* on [page 244](#) for more information.

Plugin Graphs

Plugin graphs should be created using the Perl or Python API and should have a `.upl` or `.upy` file extension. Examples, templates, and documentation are available in the `plugins/Graph` directory in your *Understand* installation and the plugins Git repository at <https://github.com/stinb/plugins>.

To run a project-level graph, choose **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 or architecture-node-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.

See *Importing Graphical View Plugins* on [page 287](#) for more information.

CodeCheck Scripts

Custom checks appear in the CodeCheck configuration dialog grouped by name.

See *Writing CodeCheck Scripts* on [page 312](#) for more information.

Metric Plugins

Metric plugins are Python scripts that can be installed the same as any other *Understand* plugin. (The Perl API is not supported for metric plugins.) Once installed, the custom metrics are listed in any metric list where they are applicable.

For example, you can create custom coverage metrics related to coverage testing or number of commits in Git. See the `plugins/Metric` directory in your *Understand* installation or the Metric folder in the Git repository at <https://github.com/stinb/plugins> for example plugins.

A metric plugin should expect an object with information about the requested metric, a database, and a target. The target can be an entity, architecture, or database.

A metric plugin must contain a “value” function that returns a number. The other functions in a metric plugin provide information about the metric (ID, name, description) and when it is available.

If a metric plugin is slow, remember that metrics are generally calculated by background threads.

See *Metric Plugins* on [page 256](#) for more information.

Architecture Plugins

Automatic architecture plugins are Python scripts that run every time you open an *Understand* project. They allow you to easily create your own architectures that are automatically updated.

See *Using Automatic Architectures* on [page 219](#) for more information.

Chapter 15 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	347
Using the understand Command Line	355
Using Buildspy to Build Understand Projects	356

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 with 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 with 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 349
analyze	Analyzes the project files	page 353
codecheck	Runs CodeCheck	page 354
convert	Convert a legacy .udb project to a .und project	page 349
create	Creates an empty project	page 349
export	Exports settings, dependencies, or architectures	page 352
help	Gives help information for a command	page 349
import	Imports project settings and architectures	page 352
license	Provide information about the software license	page 351
list	Lists information about the project	page 351
metrics	Generates project metrics	page 353
process	Runs all the commands in a text file in batch mode	page 347
projectinfo	Provide information about the project	page 351
purge	Purges the project database	page 353
remove	Removes files, directories, roots, and architectures	page 351
report	Generates project reports	page 353

Option	Discussion	See
settings	Sets project settings and overrides	page 352
upperl	Runs Perl scripts	page 354
version	Shows the current software version	page 349

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 full version of *Understand* for the und command tool by using the following command:

```
und version -full
```

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 352](#)) 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
```

You can create a project for a specific Git commit by using the -gitcommit option to specify a githash value.

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
```

If you are adding files using multiple command lines, you can use the `-analyzelater` option to prevent the database from being analyzed after every command line.

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.lis
```

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.

For more information, use the following command:

```
und help add
```

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. These take precedence over named roots defined in the *Understand* project configuration. See [page 66](#). For more information, use the following command:

```
und help roots
```

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 settings 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, dependencies, macros, includes, or changed entities 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 examples 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 -mode link 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
```

Use the `-errors`, `-warnings`, or `-accuracy` options to customize the output messages.

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 351](#)), and changed using the `settings` command (see [page 352](#)). 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 351](#)), and changed using the `settings` command (see [page 352](#)). 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 in *Configuring Checks on page 291*.

Options are provided to specify which files or architecture 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 of 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> .
-disablewebengine	Open web pages (such as Welcome and Home) in your default web browser rather than within <i>Understand</i> . (Linux only)
-existing	Detects any instance of <i>Understand</i> and sends 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. (Buildspy is the command-line equivalent of the Build Watcher feature that is part of the New Project Wizard within *Understand*; see *Using Build Watcher to Create Projects* on [page 43](#) for details.)

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](#).

This chapter lists the menu 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	358
Edit Menu	359
Search Menu	359
View Menu	360
Project Menu	360
Architectures Menu	361
Reports Menu	361
Metrics Menu	361
Graphs Menu	361
Checks Menu	362
Annotations Menu	362
Tools Menu	362
Compare Menu	363
Window Menu	363
Help Menu	364

File Menu

The **File** menu in *Understand* contains the following commands:

Command	See
New > Project	page 37
New > File	page 199
Open > Project	page 22
Open > Legacy .udb Project	page 22
Open > File	page 199
Close <i>project_name</i>	page 22
Export	page 285
Save <i>filename</i>	page 184
Save <i>filename</i> As	page 184
Save All	page 184
Page Setup	page 208
Print <i>filename</i>	page 208
Print Entity Graph	page 286
Recent Files	page 112
Recent Files > Clear Menu	page 112
Recent Projects	page 112
Recent Projects > Clear Menu	page 112
Exit	page 22

Edit Menu

The **Edit** menu in *Understand* contains the following commands:

Command	See
Undo	page 183
Redo	page 183
Cut	page 182
Copy	page 182
Copy Image to Clipboard	page 263
Paste	page 182
Select All	page 182
Comment Selection	page 196
Uncomment Selection	page 196
Change Case	page 196
Toggle Overtyping	page 198
Zoom > Zoom In	page 179
Zoom > Zoom Out	page 179
Zoom > Reset Zoom	page 179
Fold All	page 196
Soft Wrap	page 198
Hide Inactive Lines	page 196

Search Menu

The **Search** menu in *Understand* contains the following commands:

Command	See
Find	page 176
Find Previous	page 176
Find & Replace	page 176
Go to Line	page 181
Go to Matching Brace	page 195
Instant Search	page 161
Find in Files	page 163
Replace in Files	page 166
Show Search Results	page 165
Find Entity	page 168

View Menu

The **View** menu in *Understand* contains the following commands:

Command	See
Toolbars	page 175
Browse Mode	page 182
Analysis Log	page 102
Bookmarks	page 199
Contextual Information	page 177
Dependency Browser	page 148
Entity Filter	page 137
Entity Locator	page 168
Favorites	page 156
Information Browser	page 139
Metrics Browser	page 248
Previewer	page 195
Project Browser	page 145
Scope List	page 180
Window Selector	page 172
Violation Browser	page 302

Project Menu

The **Project** menu in *Understand* contains the following commands:

Command	See
Overview	page 247
Configure Project	page 46
Analyze Changed Files	page 102
Analyze All Files	page 102
Improve Project Accuracy > Undefined Macros	page 105
Improve Project Accuracy > Missing Includes	page 106
Improve Project Accuracy > More Information	page 104
Export Dependencies > Architecture Dependencies	page 152
Export Dependencies > File Dependencies	page 152
Export Dependencies > Class Dependencies	page 152
Interactive Reports	page 343

Architectures Menu

The **Architectures** menu in *Understand* contains the following commands:

Command	See
Browse Architectures	page 211
Design Architecture	page 217
Manage Orphans	page 207
Add <entity> to Architecture	page 215
Remove <entity> from Architecture	page 215

Reports Menu

The **Reports** menu in *Understand* contains the following commands:

Command	See
Configure Reports	page 225
Generate Reports	page 226
View Reports > HTML	page 227
View Reports > Text	page 227

Metrics Menu

The **Metrics** menu in *Understand* contains the following commands:

Command	See
Browse Metrics	page 248
Export Metrics	page 250
Metrics Treemap	page 253

Graphs Menu

The **Graphs** menu in *Understand* contains the following commands:

Command	See
Dependency Graphs > By <i>architecture</i>	page 220
Dependency Graphs > Load Saved Dependency Graph	page 263
UML Class Diagram > By <i>architecture</i>	page 261
Graphs for <i>selected entity</i>	page 263

Checks Menu

The **Checks** menu in *Understand* contains the following commands:

Command	See
Browse Violations	page 302
Open CodeCheck	page 290
Select Checks	page 291
Re-Run <checks>	page 290
Analyze Changes and Re-Run <checks>	page 290
Configure Dependency Checks	page 222
Run Dependency Checks	page 222

Annotations Menu

The **Annotations** menu in *Understand* contains the following commands:

Command	See
Browse All Annotations	page 204
Annotation Viewer	page 203
View <file> Annotations	page 203
Configure Annotations	page 132
Manage Orphans	page 207
Annotate	page 205

Tools Menu

The **Tools** menu in *Understand* contains the following commands:

Command	See
Run Command	page 342
User Tools > <i>tool_name</i>	page 339
User Tools > Configure	page 331
Editor Macros > Record Macro	page 198
Editor Macros > Replay Macro	page 198
Editor Macros > Save Macro	page 198
Editor Macros > Configure Macros	page 127
Clear Script Cache	page 343
Options	page 108

Compare Menu

The **Compare** menu in *Understand* contains the following commands:

Command	See
Comparison Projects	page 320
Locate Changed Entities	page 320
Changes Treemap	page 324
Compare Files/Folders	page 316
Compare Entities	page 319
Compare Arbitrary Text	page 320

Window Menu

The **Window** menu in *Understand* contains the following commands:

Command	See
Close <i>current_file</i>	page 184
Close All Document Windows	page 172
Session Browser	page 175
Release Window	page 172
Split Workspace Vertically	page 172
Split Workspace Horizontally	page 172
Unsplit Workspace	page 172
Reset Windows to Default Layout	page 175
< <i>open source file list</i> >	page 172
Windows Navigator	page 172

Help Menu

The **Help** menu in *Understand* contains the following commands:

Command	See
Help Content	page 17
Key Bindings	page 116
Example Projects	page 17
Perl API Documentation	page 33
Python API Documentation	page 33
Frequently Asked Questions	page 17
Show Welcome Page	page 22
Check for Updates	page 22
Licensing	page 17
Privacy	page 17
Reset All Hints	page 17
Show Suggestions	page 104
About Understand	page 17

Index

Symbols

- : in F77 identifiers, Fortran 90
- ? in regular expressions 171
- ? wild card 171
- . in regular expressions 171
- .NET framework 87
- " prefixing octal constants or string literals, Fortran 90
- "" surrounding normal includes 83
- [-] in regular expressions 172
- [] in regular expressions 172
- [^] in regular expressions 172
- * comments field 90
- * in regular expressions 171
- * wild card 171
- /* ... */ C-style comments 79, 90
- \ in regular expressions 172
- \< in regular expressions 171
- \> in regular expressions 171
- # comment in command line file 354
- ^ in regular expressions 171
- + expanding tree in Information Browser 140
- + in regular expressions 171
- <> surrounding system includes 83
- = macro definition in command line 354
- | in regular expressions 172
- \$ in regular expressions 171
- \$ prefixing external tool parameters 332, 333

A

- a 22
- About Understand option, Help menu 17
- Activate when Control key is pressed field, Browse Mode Editor options 128
- actual parameters, relationship to formal parameters, Ada 69
- Ada
 - configuration for 68
 - macro definitions 70
 - versions supported 15
- Ada category, Project Configuration dialog 68
- Add found include files to source list field, C++ Includes 83
- Add found system include files to source list field, C++

- Includes 83
- addDir option, und command 349
- addFiles option, und command 350
- Adobe Acrobat, saving graphical views to 287
- Advanced category, Editor options 123
- Aggregate by Architecture menu 277
- aggregate metrics 65
- Aggregate Nodes by menu 277
- Alerts category, User Interface options 113
- Allow 90
- Allow Call Expansion menu 277
- Allow Colons in Names field, Fortran 90
- Allow C-style comments field, Fortran 90
- Allow embedded SQL field, Pascal 96
- Allow function declaration without parentheses field, Fortran 90
- Allow interactivity during intensive processing field, General options 110
- Allow Nested Comments field, C++ 79
- Allow parameter declaration without parentheses field, Fortran 90
- Allow quote in octal constants field, Fortran 90
- Analysis Log option, View menu 103
- Analysis Log window 103
 - reopening previous analysis Log 174
 - saving to text file 103
- Analysis Options area 332
- Analyze All Files option, Project menu 102
- Analyze category, Understand Options dialog 119
- Analyze Changed Files option, Project menu 102
- analyze option, und command 353
- Analyze unincluded headers in isolation field 83
- analyzeFiles option, und command 353
- analyzing projects 102
 - after changing configuration of 47
 - beep on completion of 114
 - on command line 353
 - improving 104
- and operators, including in strict complexity, Ada 69
- angle brackets (<>) surrounding system includes 83
- Animate windows/drawers field, User Interface options 111
- annotations 203
 - attaching files 206
 - automatic ignores 309
 - creating 205
 - deleting 207

- editing 205
 - ignore CodeCheck violations 207
 - ignore violations 309
 - orphans 207
 - searching 204
 - sharing 204
 - tagging with hashtag 206
 - viewing 203, 204
 - Annotations category, Project Configuration dialog 132
 - Annotations menu 277, 362
 - Annotations Viewer 203
 - anti-virus software, turning off while generating reports 226
 - Apache Lucene syntax 161
 - APIs 33, 343
 - Append the names of externally linkable entities with field
 - C++ 80
 - Fortran 91
 - Application font field, General options 109
 - Architecture Browser 31, 211
 - duplicating architectures 213
 - graphical views in 212, 220
 - metrics in 213, 223, 252
 - opening Architecture Builder from 213
 - renaming architectures 213
 - right-clicking in 214
 - Architecture Builder 213
 - Architecture Name menu 277
 - architectures 20, 210
 - auto-architectures 212
 - duplicating 213
 - editing. *See* Architecture Builder
 - exporting dependency list 152
 - graphical views of 212, 220
 - hierarchies of, exploring 211
 - list of 211
 - listing 31
 - metrics 64
 - metrics about 213, 223, 252
 - navigating 31
 - orphan nodes 216
 - plugins 219
 - renaming 213, 215
 - Architectures menu 361
 - arrow styles 272
 - arrows, for Information Browser history 144
 - ASP style tags 101
 - assembly
 - configuration for 72
 - embedded in C code 81, 85
 - include files, directories for 72
 - support 15
 - Assembly category, Project Configuration dialog 72
 - Assign References menu 282
 - Assignments graph 279
 - AST cache 79
 - asterisk (*)
 - in regular expressions 171
 - wild card 171
 - Auto Category, C++ Includes 83
 - auto-architectures 31, 212
 - Auto-complete fields, Advanced Editor options 125
 - Auto-indent fields, Advanced Editor options 125, 126
 - Automatic compool file field, JOVIAL 95
 - Automatically analyze changed files on save field 119
 - Automatically analyze changed files periodically field 119
 - AUTOSAR checks 291
- ## B
- background processing
 - CodeCheck 290, 294
 - external commands 342
 - interactivity during 110
 - metrics 256
 - view status 102
 - backslash (\) in regular expressions 172
 - base classes
 - count of 242
 - Base Classes menu 277
 - baseline ignore 308
 - Basic
 - configuration for 74
 - support 15
 - beep on analysis completion 114
 - beep, preferences 114
 - Bidirectional Edges menu 277
 - bitmaps, saving graphical views as 285
 - Blank metric 244
 - blocks
 - expanding and collapsing 179, 196
 - in Filter Area 26
 - blue background for text 128

blue text 128
bookmarks
 creating 199
 list of 174, 200
 navigating 199
Bookmarks option, View menu 174, 200
Boolean searches 162
braces, matching. *See* brackets
brackets
 in regular expressions 172
 matching 195
Browse Architectures option, Project menu 211
Browse Metric Charts option, Metrics menu 254
Browse Metrics option, Metrics menu 246, 248
Browse Mode option, View menu 182
Browse Mode, Source Editor 182
Browser Area 21
Bug Hunter 291
Build Log 36
Build Project page 43
Build Watcher 43
Buildspy 356

C

C API 343
C/C++
 API for custom reports 33, 229, 246
 configuration for 76, 79, 87
 include files, auto including 83
 include files, directories for 81
 include files, ignoring 84
 macro definitions 85
 preprocessor directives 85
 strict analysis 76, 87
 versions supported 15
C#
 configuration for 87
 reference files, importing 87
 versions supported 15
C# category, Project Configuration dialog 87
C++ (Strict) category, Project Configuration dialog 76, 87
C++ category, Project Configuration dialog 79
caching include files, C++ 81
Calendar auto-architecture 31, 212
Called By Depth menu 277

Called by menu 277
Calls Depth menu 277
capitalization, of selected text 196
caret (^) in regular expressions 171
Caret Line field, Editor options 122
Cascading Style Sheet 226
case
 changing for selected text 196
 of displayed entity names, Ada 70
 of displayed entity names, Fortran 91
 of displayed entity names, JOVIAL 95
 of displayed entity names, Pascal 97
 of externally linkable entities, Ada 69, 95
 of externally linkable entities, Fortran 90
 of externally linkable entities, Pascal 96
 of identifier names, Fortran 90
Case of externally linkable entities field
 Ada 69, 95
 Fortran 90
 Pascal 96
Case sensitive identifiers field, Fortran 90
case sensitivity
 of comparison 329
 of Fortran identifiers 90
CBO (Count of Coupled Classes) metric 242
certification 16
Change Case option, Edit menu 196
Change Log 36
character strings, blue text for 128
Check for Updates option, Help menu 23
Checks menu 362
CIS. *See* Contextual Information Sidebar
Class Details menu 282
.class files 93
Class Metrics report 242
Class OO Metrics report 242
Class Paths category, Java 93
classes
 base classes for 242
 cohesion of 242
 coupled 242
 derived 242
 exporting dependency list 152
 extended by other classes 279
 extending other classes 279
 implemented 279
 implemented by 279

- listing in Filter Area 26
- metrics about 242
- providing without source code, Java 93
- reports about 235, 242
- clipboard
 - copying graphical view to 285
 - copying text from Source Editor to 182
- cloning Git repository 42
- Close All Document Windows option, Window menu 172, 184
- Close option
 - File menu 23
 - Window menu 172, 184
- Close Selected Window(s) command 174
- Cluster menu 278
- CMake import 59
- COBOL files 50
- Code metric 244
- Code Style menu 278
- CodeCheck 288
 - background checks 103, 294
 - checks 292
 - compliance reports 310
 - configuration 292
 - exporting results 310
 - plugins 312
 - result log 301
 - running 290
 - scripts 312
- cohesion for class data members 242
- Collapse menu 278
- colon (:) in F77 identifiers, Fortran 90
- Color Mode field, Advanced Editor options 123
- Color Theme field, Advanced Editor options 123
- colors
 - for Source Editor 123
 - gradient for metric value 274
 - graphs 272
 - in comparison 329
 - in HTML reports 226
 - for printing source code 123
 - of rows in User Interface, alternating 115
 - in Source Editor, customizing 179, 127
 - in Source Editor, default 128
- Column Chooser 169
- column truncation
 - Fortran 90
 - JOVIAL 94
- command line. *See* und command; understand command
- command renames, reports about 236
- Command window 342
- Command Window Font 112
- commands, external. *See* external tools
- Comment metric 244
- Comment Selection menu option 196
- comments
 - adding or removing 196
 - associating with entities, Ada 70
 - associating with entities, C 78
 - associating with entities, C++ 81
 - associating with entities, Fortran 91
 - associating with entities, Java 93
 - associating with entities, Python 99
 - associating with entities, Web 101
 - checking spelling 202
 - C-style 79
 - green text for 128
 - ignoring violations 307, 309
 - nested, C++ 79
- Comments menu 278
- companion file 199
- Compaq Pascal 96
- Compare Arbitrary Text option 320
- Compare Arbitrary Text option, Tools menu 320
- Compare Entities option, Compare menu 319
- Compare files by content field 83
- Compare Files/Folders option, Compare menu 316
- Compare menu 363
 - Compare Entities option 319
 - Compare Files/Folders option 316
- compare projects 320
 - from Git commits 43
- compilation environment, Ada 69
- compilation unit
 - With By relationships of 284
- Compile time dependencies 62
- compiler
 - compared to Understand 35
- Compiler field
 - C++ 79
- Compiler Include Paths field, C++ 79
- Compiler warnings 291
- complexity

- exception handlers included in, Ada 69
- FOR-loops included in, Ada 69
- metrics 63
- strict, and/or operators in, Ada 69
- Compliance Report 310
- Component field, Key Bindings options 116
- compool file, JOVIAL 95
- Configure option, Project menu 46
- Configure Reports option, Reports menu 225
- constants
 - displayed in graphical views 278
 - reports about 239
- Constants menu 278
- contact information 15
- Context Browser, Contextual Information Sidebar 177
- contextmenu option, understand command 355
- Contextual Information option, View menu 177
- Contextual Information Sidebar (CIS) 177
- !COPY directives, directories to search, JOVIAL 95
- Copy category, JOVIAL 95
- Copy Filename command 146
- Copy option, Edit menu 182
- copying text
 - rectangular area 182
- Count and/or operators in strict complexity field, Ada 69
- Count exception handlers in complexity field, Ada 69
- Count for-loops in complexity field, Ada 69
- Count of All Methods metric 242
- Count of Base Classes metric 242
- Count of Coupled Classes metric 242
- Count of Derived Classes metric 242
- Count of Instance Methods metric 242
- Count of Instance Variables metric 242
- Count of Methods metric 242
- coupled classes 242
- Create and cross-reference record object components
 - field, Ada 69
- Create implicit special member functions field 80
- Create references in inactive code field, C++ 80
- Create references in inline assembly, C++ 81
- Create references to local objects field, C++ 80
- Create references to macros during macro expansion
 - field, C++ 80
- Create references to parameters field, C++ 80
- Create relations between formal and actual parameters
 - field, Ada 69
- Crossing layout option 280

- cross-reference reports 230
- cross-referencing record object components, Ada 69
- CSS files 15
- C-style comments
 - in Fortran 90
 - nesting of, allowing 79
- CSV file
 - exporting dependencies to 152
 - exporting metrics to 246, 250
- Ctrl+Alt+F keystroke, find and replace text 176
- Ctrl+F keystroke, find text 176
- Ctrl+j keystroke, jump to matching bracket 195
- Ctrl+right-click keystroke
 - creating new windows 24, 135
- Ctrl+Shift+j keystroke, select text in brackets 195
- CUDA files 15, 75, 78
- \$Cur variables 332, 333
- Cut option, Edit menu 182
- cwd option
 - understand command 355
- Cyclomatic complexity 237, 244
- cyclomatic complexity metrics 64
- Cytoscape
 - installation location 120
 - XML output 152
- Cytoscape XML 152

D

- dark mode 110, 127
- Data Dictionary report 230
- Data Flow In Depth menu 278
- Data Flow Out Depth menu 278
- database 20
 - changes to format of 36
 - file extension for 20, 35
 - multi-user read/write access for 36
 - See also* project
- Date Format field 124
- db option
 - understand command 355
- Debug menu 278
- DEC Pascal 96
- \$Decl variables 332, 333, 334
- Declaration Tree report 234
- declaration views
 - calling methods displayed in 277
 - constants displayed in 278

- default members displayed in 278
- extended by classes displayed in 279
- extended classes displayed in 279
- external functions displayed in 279
- file dependencies displayed in 278
- globals displayed in 279
- header files include by's displayed 280
- implemented by classes displayed in 279
- implemented classes displayed in 279
- imported entities displayed in 280
- include files displayed in 280
- inherited entities displayed in 280
- invocations displayed in 280
- local items displayed in 281
- members displayed in 281
- objects displayed in 281
- operators displayed in 281
- private members displayed in 281
- protected members displayed in 281
- public members displayed in 281
- rename declarations in 281
- static functions displayed in 283
- types displayed in 284
- used-by items shown in 284
- variables displayed in 284
- With By relationships displayed in 284
- With relationships displayed in 284
- See also graphical views
- declarations
 - implicit, report about 238
 - local, including in project, C++ 80
- declarations vs. definitions 62
- Default Members menu 278
- Default style field, Editor options 121
- default working directory 109
- Define/Declare menu 278
- definitions vs. declarations 62
- Delphi
 - versions supports 15
- Delphi Pascal 96
- Depended On By Depth menu 278
- dependencies
 - browsing 148
 - compile time vs. link time 62
 - exporting 149, 152
 - file 278
- Dependencies category, Project Configuration dialog
 - 62
- Dependency Browser 148
 - preferences 62
- Dependency category, Understand Options dialog 120
- Dependency Checks
 - configuring 222
 - exporting 223
 - running 223
- Dependency Graph 212
- Dependency Graphs option, Graphs menu 220
- Dependent Of menu 278
- Dependents menu 278
- Depends On Depth menu 278
- derived classes 242
- Derived Classes menu 278
- DFM converter exe field 96
- dictionary, spelling 202
- diff option, understand command 355
- Different Word Color in comparison 329
- directives, C++ 85
- directories
 - adding to project 50
 - comparing 316
 - copying 318
 - deleting from project 51
 - overriding settings for 52
 - watched, setting 51, 52, 53
- Directory Structure auto-architecture 31, 212
- disk space requirement 16
- Display entity names as field
 - Ada 70
 - Fortran 91
 - JOVIAL 95
 - Pascal 97
- DIT (Max Inheritance Tree) metric 242
- docking windows 19
- Document Area 21
- documentation 17
- dollar sign (\$)
 - in regular expressions 171
 - prefixing external tool parameters 332, 333
- DOS line termination style
 - for reports 226
 - for saving source files 122
- DOT format, saving graphical views as 285
- double quotes. See quotes
- double-clicking

- in Bookmarks area 200
- column header dividers 168
- entities in Entity Filter 142
- entities in Information Browser 142
- in Exploring View 147
- messages in Analysis Log window 103
- in Project Browser 248
- in Search Results window 165
- in Source Editor 30
- drawers 19, 111
- Drill down command 255
- drop-down menu *See* right-clicking
- Duplicate Architecture menu option 213
- duplicate references, C++ 81
- Duplicate Subtrees menu 279

E

- Edge Labels menu 279
- edges
 - in graphs 267
 - styles 273
- Edit Architecture menu option 213
- Edit Graphic Filters menu option 268
- Edit menu 359
 - Change Case option 196
 - Copy option 182
 - Cut option 182
 - Select All option 182
 - Toggle Overtyping option 198
- Editor category, Understand Options dialog 121
- Editor Macros option, Tools menu 198
- editor windows, saving automatically 109
- editor, external 129
- editor, source. *See* Source Editor
- Effective C++ checks 291
- Emacs editor 129, 130
- embedded SQL, in Pascal 96
- EN 50128 2, 16
- Enable Editor Tooltips field 129
- Enable permissions checking for NTFS filesystems field 110
- encoding formats 55, 122
- entities 20
 - comments associated with, Ada 70
 - comments associated with, C 78
 - comments associated with, Fortran 91

- comments associated with, Java 93
- comments associated with, Python 99
- comments associated with, Web 101
- comparing two entities 319
- current, information about 177
- displaying source for. *See* Source Editor
- favorites of, displaying 156
- favorites of, marking 155
- full name of, displaying 138
- graphical views of. *See* graphical views
- hierarchy of, exploring 147, 177
- information about. *See* Information Browser
- list of, alphabetic 230
- listed in Entity Filter 137
- listed in Entity Locator 27, 168
- listed in Filter Area 26
- metrics for. *See* metrics
- names of, formatting for reports 57
- references for 143
- relationships between 20
- sorting of 137
- unknown 284
- unresolved 284
- See also specific entities*
- Entity Filter 137
 - displaying information about entities in 139
 - displaying source of selected entity 142
 - entities not listed in 27, 168
 - jumping to entities in 26
 - location of 21
 - right-clicking in 25
- entity index, for HTML reports 227
- Entity Locator 27, 168
 - column headers in, customizing 169
 - columns in, hiding and reordering 169
 - columns in, resizing 168
 - filtering by selection 170
 - filtering manually 171
 - filtering with regular expressions 171
 - filtering with wildcards 171
 - length of names in 168
 - opening 168
 - right-clicking in 27, 168
 - right-clicking on column header 169, 171
 - right-clicking on entities 170
 - sorting entities in 169
- Entity Locator option, View menu 27, 168

Entity Name Format As menu 279
 entity_index.html file 227
 enum, refactoring 192
 environment variables
 using in include paths, assembly 73
 using in include paths, C++ 82
 equal sign (=) in macro definition in command line 354
 errors
 displaying from Analysis Log window 103
 parse errors, prompting for, Ada 70
 parse errors, prompting for, Fortran 91
 Essential metric 244
 Example Projects option, Help menu 17
 Exception Cross-Reference report 233
 exceptions
 handlers for, including in Ada complexity 69
 reports about 233
 excluded violations 305
 -existing option, understand command 355
 Expand Macros menu 279
 Exploring view 147
 Export Analysis Errors command 303
 Export Dependency CSV 152
 Export Metrics option, Metrics menu 246, 250
 exporting
 CodeCheck results 310
 graph themes 276
 tool commands 341
 Extend Tree report 235
 Extended By menu 279
 Extends menu 279
 extensions. *See* file extensions
 external editor 129
 External Editor category, Editor options 129
 External Functions menu 279
 external tools
 adding to Right-click Menu 333, 338
 adding to toolbar 340
 adding to User Tools menu 339
 commands for, importing and exporting 341
 commands for, running 342
 configuring 331
 editor 129
 externally linkable entities
 case of, Ada 69, 95
 case of, Fortran 90
 case of, Pascal 96

 prefix for 80, 91, 97
 suffix for, C++ 80
 suffix for, Fortran 91
 Extract Function refactoring 190
 Extract Temp refactoring 193

F

F5 key, Find in Files option 163
 favorites 28
 displaying 156
 marking entities as 155
 Favorites option, View menu 28, 156
 File Average Metrics report 244
 File Contents report 231
 File Explorer, open in 146
 file extensions
 configuring for project 54
 for database 20, 35
 File Information, Contextual Information Sidebar 177
 File menu 358
 Close option 23
 New > File option 199
 New > Project option 37
 Open > File option 199
 Open > Legacy .udb Project option 22
 Open > Project option 22
 Page Setup option 208
 Print Entity Graph option 286
 Print option 208
 Recent Projects option 22
 Save All option 184
 Save option 184
 File Metrics report 243
 File Mode field, Editor options 122
 File Options category, Project Configuration dialog 55
 file permission checking 110
 File Sync box 142
 File Types category, Project Configuration dialog 54
 Filename menu, graphical views 279
 filenames
 in graphical views 279
 for reports 227
 in title areas 112
 files
 comparing 316
 copying 318
 exporting dependency list 152

- Information Browser for 177
- location included in listing of 138
- renaming 187
- searching 28, 163
- searching and replacing in 166
- toolbar for 175
- Files category, Project Configuration dialog 49
- fill color 272
- Filter Area 26
 - See also Entity Filter; Project Browser
- Filter By Selection menu option 170
- Filter field, Entity Filter 138
- Filter menu 279
- filters
 - for graphical views 268
 - See also Entity Filter; Entity Locator
- Find dialog 176
- Find Entity option, Search menu 168
- Find in Files dialog 21, 28
- Find in Files menu option 28, 163
- Find option, Search menu 176
- Find Results window 165
- Finder, open in 146
- fixed file format, Fortran 90
- Fix-It Hint 304
- folders. See directories
- folding code 196
- font
 - Command Window 112
 - for printing source code 123
 - for Source Editor windows 121
 - in Source Editor windows 196
 - in Understand, changing 109
- Font Size field, Advanced Editor options 123
- FOR-loops, including in complexity metrics, Ada 69
- formal parameters, relationship to actual parameters,
 - Ada 69
- Format field, Fortran 90
- Fortran
 - configuration for 89
 - extensions supported 15
 - include files 91
 - macro definitions 91
 - reports showing non-standard extension usage 238
 - versions supported 15
- Fortran category, Project Configuration dialog 89
- Fortran Extension Usage report 238

- frames in windows, sliding 19
- Frameworks Category, C++ Includes 83, 84
- free file format, Fortran 90
- freeze project at specific commit 42
- function declarations without parentheses, Fortran 90
- Function Name menu 279
- Function Pointer menu 279
- functions
 - external 279
 - in graphical views 282
 - listing in Filter Area 26
 - metrics about 244
 - reports about 231, 244
 - static 283
 - See also invocations; procedures; program units
- fuzzy C/C++ analyzer 79
- fuzzy search 162

G

- gcc build 356
- General category, Understand Options dialog 109
- Generate Reports option, Reports menu 226
- Generic Instantiation report 236
- Getting Started dialog
 - displaying at startup 109
 - opening 23
- Git
 - blame 123, 126, 181
 - check uncommitted changes 290, 295
 - cloning 42
 - commit hash 42, 296
 - comparing project commits 320
 - create project for commit 349
 - create project for repository 38, 42
 - differences 181
 - error messages 42
 - highlighting changes 124
 - history 326
 - repository specification 43, 61
 - uncommitted lines 124
- Global Objects menu 279
- Globals menu 279
- Go To Line dialog 181
- Go to Matching Brace option, Search menu 195
- gradient color for metric 274
- Graph Architecture 212, 220
- Graphic Filter dialog 268

- graphical user interface (GUI), parts of 21
 - graphical views 32
 - of architecture 212, 220
 - browsing 266
 - calling methods displayed in 277
 - constants displayed in 278
 - copying to clipboard 285
 - customizing 271
 - default members displayed in 278
 - diagram of 21
 - entity name truncation for 283
 - expanding or contracting nodes in 268
 - extended by classes displayed in 279
 - extended classes displayed in 279
 - external functions displayed in 279
 - file dependencies displayed in 278
 - filename display options 279
 - filtering 268
 - fullnames displayed in 281
 - function pointers displayed in 279
 - globals displayed in 279
 - header files include by's displayed 280
 - hierarchy levels displayed in 280
 - implemented by classes displayed in 279
 - implemented classes displayed in 279
 - imported entities displayed in 280
 - include files displayed in 280
 - inherited entities displayed in 280
 - intrinsic functions displayed in 280
 - invocations displayed in 280
 - layout configuration for 280
 - legend 265, 271
 - local items displayed in 281
 - members displayed in 281
 - multiple subtrees displayed in 279
 - objects displayed in 281
 - operators displayed in 281
 - parameters, displaying in 281
 - path highlighting in 270
 - printing 286
 - private members displayed in 281
 - procedures displayed in 282
 - protected members displayed in 281
 - public members displayed in 281
 - rename declarations in 281
 - reusing 265
 - saving 285
 - saving as PDF 287
 - shading 274
 - spacing entities in 283
 - SQL entities shown in 283
 - static functions displayed in 283
 - synchronizing with other windows 265
 - themes 275
 - types displayed in 284
 - unknown entities, displaying 284
 - unresolved entities, displaying 284
 - used-by items shown in 284
 - uses items shown by 284
 - variables displayed in 284
 - With By relationships displayed in 284
 - With relationships displayed in 284
 - See also structure views; hierarchy views
 - Graphical Views menu option 212, 220
 - Graphs category, Understand Options dialog 130
 - Graphs for option, Graphs menu 220
 - Graphs menu 361
 - Dependency Graphs option 220
 - Graphs for option 220
 - Project Graphs option 229
 - gray background for text 128
 - Green Hills project 38
 - green text 128
 - GUI (graphical user interface), parts of 21
 - GVIM editor 130
- ## H
- hashtag in annotations 203, 206
 - header files 280
 - dependencies 62
 - help 17
 - Help menu 364
 - About Understand 17
 - Check for Updates option 23
 - Example Projects option 17
 - Help Content option 17
 - Key Bindings option 17, 198
 - Perl API Documentation option 17, 246
 - Python API Documentation option 17, 33, 246, 343
 - Show Suggestions option 17, 104
 - Show Welcome page option 23
 - help option, und command 349
 - Hersteller Initiative Software (HIS) metrics 291
 - Hide Inactive Lines option, View menu 196

- hierarchy views 32
 - layout of 280
 - multiple subtrees displayed in 279
 - number of levels in 280
 - parameters, displaying in 281
 - See *also* graphical views
- Highlight Color
 - in comparison 329
- Highlight Different Words
 - color 329
 - in comparison 329
- highlighting full line at cursor 122
- history
 - Git 326
 - Information Browser 140, 144
 - Source Editor 173
 - toolbar for 175
- History category, Project Configuration dialog 61
- Home tab 246
- Horizontal Non-Crossing layout option 280
- HTML
 - colors in reports 226
 - entity index in reports 227
 - exporting metrics to 246
 - file extensions 100
 - files 15
 - generating reports as 57, 225, 226
 - viewing reports as 227
- Hyper Grep. See Find in Files dialog
- hyphen (-) collapsing tree in Information Browser 140

I

- IB. See Information Browser
- IEC 61508 2, 16
- IFANIN (Count of Base Classes) metric 242
- Ignore category, C++ Includes 84
- Ignore directories in include names field 83
- Ignore Parent Overrides field 53
- ignore violations 305
 - automatic ignores 309
 - baseline ignore settings 308
 - comments 307
 - in CodeCheck 305
 - notes 308
 - notes about 308
 - with Refactor tool 307

- Implementation fields field, JOVIAL 95
- Implemented By menu 279
- Implements menu 279
- implicit special member functions, C++ 80
- Implicitly Declared Objects report 238
- import files
 - adding to project, Python 99
 - excluding from dependencies 62
- Import report 236
- imported entities, displaying in graphical views 280
- importing
 - graph themes 276
 - tool commands 341
- Imports category
 - Python 99
- Imports menu 280
- importusertools option, understand command 355
- inactive code, cross-reference information for, C++ 80
- inactive lines, hiding 196
- Include Depth menu 280
- Include File Cross-Reference report 233
- include files
 - adding as source files, assembly 73
 - adding as source files, C++ 82
 - adding as source files, Fortran 91
 - adding before each project file, C++ 83
 - adding in bulk, assembly 73
 - adding in bulk, C++ 82
 - adding in bulk, Fortran 91
 - adding to project, assembly 72
 - adding to project, C++ 81
 - adding to project, Fortran 91
 - compiler path for, C++ 79
 - displayed in graphical views 280
 - environment variables in paths for, assembly 73
 - environment variables in paths for, C++ 82
 - excluding from dependencies 62
 - ignoring during analysis, C++ 84
 - Pascal search path 97
 - replacement text for, C++ 84
 - replacement text for, Fortran 91
 - reports about 233
 - system include files, C++ 83
- Include Global Objects menu 280
- Include line numbers in rich text field 125
- Include Standard Libraries menu 280
- Include Unresolved menu 280

- Include Virtual Edges menu 280
- Includeby Depth menu 280
- Included By menu 280
- Includes category
 - C++ 81
 - Fortran 91
- Includes menu 280
- Indent field, Editor options 122
- indentation 125, 126
 - fixing 197
- Information Browser (IB) 26, 29, 139
 - architecture information in 211
 - copying information in 140, 144
 - displaying from entity in Entity Filter 139
 - displaying source of selected entity 142
 - entity information displayed by, choosing 141
 - expanding and collapsing the tree 140
 - for current entity 177
 - for current file 177
 - history for 140, 144
 - location of 21
 - metrics in 143, 246
 - multiple occurrences open 142
 - References in 143
 - right-clicking in 25
 - saving information in 140, 144
 - searching 141
 - synchronizing 142
- Information Browser option, View menu 139
- inheritance metrics 64
- Inherited By menu 280
- Inherits menu 280
- initialization files 23
- Inline Function refactoring 189
- Inline Scope refactoring 192
- Inline Temp refactoring 192
- Insert Spaces Instead of Tabs field, Editor options 122
- installation 16
- instance methods 242
- instance variables 242
- Instant Search 161
- Instant Search menu option 161
- instantiation
 - reports about 236
- interactivity during background processing 110
- interfaces
 - listing in Filter Area 26

- interrupt handlers, listed as unused program units 240
- intrinsic functions, parsing, Fortran 90
- Intrinsic menu 280
- Intrinsics file field, Fortran 90
- Invocation Tree report 235
- invocations
 - reports about 235
- Invocations menu 280
- ISO 26262 2, 16

J

- .jar files 93
- Java
 - configuration for 92
 - versions supported 15
- Java API 33, 343
- Java category, Project Configuration dialog 92
- Javascript
 - file extensions 100
- JavaScript files 15
- JavaScript in reports 57
- JDK, versions supported 15
- JNI external entities, Java 92
- JOVIAL
 - configuration for 94
 - directories for !COPY directives 95
 - versions supported 15
- JOVIAL category, Project Configuration dialog 94
- JPEG format, saving graphical views as 285
- jQuery analysis 100
- Jump to Matching Brace menu option 195
- Jump to Matching Directive menu option 195

K

- Keil Uvision project 38
- Key Bindings category, Understand Options dialog 116, 198
- Key Bindings option, Help menu 17, 198
- keyboard mappings 116
- Keyboard Scheme field, Key Bindings options 116
- keywords, orange text for 128
- KNI external entities, Java 92
- knot metrics 64

L

- Language auto-architecture 31, 212

Languages category, Project Configuration dialog 48
-lastproject option, understand command 355
-lastproject_cwd option, understand command 355
Layout menu 280
layout, for graphical views 280
LCOM (Percent Lack of Cohesion) metric 242
legacy project
 conversion 349
legacy projects, converting 22
Legend
 controlling graph 265, 271
 customizing graphs 271
 edge styles 273
 node styles 272
 saving styles 275
 themes 275
 viewing 271
Legend menu 280
Less memory usage versus speed field, Ada 69
Level menu 280
libraries, standard
 Ada, directory for 69
 excluding 62
 including in Analysis Log 119
 Pascal, paths for 97
Library directories field, Ada 70
Library menu 281
licensing 16
line count metrics 63
line endings for source files 122
line numbers, displaying in Source Editor 179
line styles 272
line termination style
 for reports 226
line termination style, saving source files 122
Lines metric 244
Link time dependencies 62
Linux OS support 16
@lisfile.txt file 354
local object declarations, including in project, C++ 80
local parameters, listed in Entity Locator 27
Locals menu 281
Lucene, Apache syntax 161

M

Macintosh line termination style 122

MacOS support 16
Macro Cross-Reference report 233
macros
 adding in bulk, C++ 86
 changing, Ada 71
 compiler-specific, C++ 79
 defining in command line 354
 defining on command line, Ada 71
 defining, Ada 70
 defining, C++ 85
 of editing changes, recording and replaying 198
 expansion text for, C++ 81
 importing, Ada 71
 listing in Filter Area 26
 recording references when expanding, C++ 80
 reports about 232, 233
 undefined, C++ 86
 See also objects
Macros category
 Ada 70
 C++ 85
 Fortran 91
 Pascal 97
Main subprograms field, Ada 70
makefile 356
Margins field, Editor options 123
"Mastering Regular Expressions" (O'Reilly) 172
Max Inheritance Tree metric 242
McCabe (Cyclomatic) complexity 237, 244
members
 cohesion of 242
 default 278
 displayed in graphical views 281
 private 281
 protected 281
 public 281
Members menu 281
memory
 caching include files, C++ 81
 optimizing analysis to use less, Ada 69
menu bar 21
menus. *See specific menus*
methods
 cohesion of 242
 instance 242
 listing in Filter Area 26
 metrics about 242

- metrics 213, 223, 252
 - categories 63
 - color scale 274
 - definitions 249
 - displayed in Information Browser 143
 - exporting to CSV file 246, 250
 - exporting to HTML 246
 - for project 247, 248
 - in Information Browser 246
 - list of, online 237, 241
 - plugins 256, 344
 - reports about 241, 242, 243, 244
 - shading graph nodes 274
 - violation 304
- Metrics Browser 246
- Metrics category, Project Configuration dialog 63
- Metrics menu 361
 - Browse Metric Charts option 254
 - Browse Metrics option 246, 248
 - Export Metrics option 246, 250
- Metrics Summary 213, 223, 252
- Microsoft Visio files, saving graphical views as 285
- Microsoft Visual C++, as editor 129
- minus sign (-) collapsing tree in Information Browser 140
- MISRA checks 291
- misspellings 202
- modifications
 - highlight in scroll bar 124
 - saving file 180
- Modified Entities menu 281
- Modified metric 244
- modules, listing in Filter Area 26
- MSDos Batch files 50
- multiple users, initialization files for 23

N

- Name menu, graphical views 281
- named roots 67
 - portability mode 67
- Navigation category, Editor options 128
- Navigation Mode, Source Editor 128
- nested comments, C++ 79
- nesting levels 64
- .NET API 344
- New File option, File menu 199

- new option, understand command 355
- New Project option, File menu 37
- New Project Wizard 37, 112
- next button 19
- Next icon 30
- NIM (Count of Instance Methods) metric 242
- NIV (Count of Instance Variables) metric 242
- No Truncation text option 283
- No Wrap text option 283
- NOC (Count of Derived Classes) metric 242
- node styles
 - in graphs 272
 - shading 274
- Node.js analysis 101
- noproject option, understand command 355
- notification preferences 113
- NTFS filesystem 110

O

- Object Cross-Reference report 232
- object oriented metrics 64
- Objective C/C++ 15, 78
 - enabling 75
- object-oriented metrics 242
- objects
 - displayed in graphical views 281
 - listing in Project Window 26
 - reports about 232, 239
- Objects menu 281
- offline mode 16
- online help 17
- Open Externally command 146
- Open File option, File menu 199
- Open last project at startup field, General options 109
- Open Project option, File menu 22
- Operators menu 281
- operators, displayed in graphical views 281
- Options option, Tools menu 108
- Options subcategory, Reports 56
- or operators, including in strict complexity, Ada 69
- orange text 128
- orphans
 - annotations 207
 - architecture nodes 216
- Output subcategory, Reports 225
- Overrides menu 281
- Overview option, Project menu 247

P

packages

- reports about 236

packages, listing in Filter Area 26

page guide, showing 122

Page Setup option, File menu 208

Parameter Calls menu 281

parameter declarations without parentheses, Fortran 90

parameters

- cross-reference information for, C++ 80

- included in listing of 138

- relationships between formal and actual, Ada 69

- reports about 232, 238, 239

- See also* objects

Parameters menu 281

parentheses, matching. *See* brackets

parse.udb file 20, 35

parsing. *See* analyzing projects

Pascal

- configuration for 96

- macro definitions 97

- search paths 97

- standard libraries, paths for 97

- versions supported 15

Pascal category, Project Configuration dialog 96

Passive menu 281

PATH definition 347

path highlighting, in graphical views 270

pattern matching. *See* regular expressions

PC line termination style

- for reports 226

PC line termination style for saving 122

.pcn file 231

PDF, saving graphical views to 287

Percent Lack of Cohesion metric 242

period (.) in regular expressions 171

Perl

- API 343

- API for custom reports 33

- CodeCheck scripts 312

- files 50, 100

- interface for custom reports 229, 246

- scripts 20

Perl API documentation 33, 343

Perl API Documentation option, Help menu 17, 246

permission checking 110

PHP files 15, 101

PHP version field 101

pin icons. *See* pushpin icons

Plm files 50

plugin

- installing 287, 343

plugins

- APIs 33, 343

- architecture 219

- CodeCheck 312, 314

- examples 17, 229, 244, 256, 344

- graphs 260, 262

- installing 314

- metrics 256, 344

- running 343

plus sign (+)

- expanding tree in Information Browser 140

- in regular expressions 171

PNG format, saving graphical views as 285

Popup Menu category, Tool Configurations dialog 338

Portability category, Project Configuration dialog 66, 67

portability mode 66

portability of project 66

pound sign (#) comment in command line file 354

Power of 10 checks 291

pragma statements, defining macros referenced in, Ada 70

Predeclared entities file field, Pascal 96

Prepend the names of externally linkable entities with field

- C++ 80

- Fortran 91

- Pascal 97

Prepend the names of JNI/KNI external entities with field, Java 92

preprocessor directives, C++ 85

Preprocessor field, Ada 69

preprocessor macros. *See* macros

preprocessor support, enabling, Fortran 90

previous button 19

Previous icon 30

Print Entity Graph option, File menu 286

Print option, File menu 208

printing

- CodeCheck results 310

- graphical views 286

- highlighted source code 124
- source files 208, 286
- privacy 16
- Private Members menu 281
- procedures
 - in graphical views 282
 - reports about 231
 - See also invocation; functions; program units
- Program Unit Complexity report 237
- Program Unit Cross-Reference report 231
- Program Unit Metrics report 243
- program units
 - metrics about 243
 - reports about 234, 235, 236, 237, 240, 243
 - With relationships for 236, 240
- programming languages
 - selecting for project 48
 - setting in Source Editor 181
- project 20
 - adding source files on command line 349
 - analyzing (parsing). See analyzing projects
 - closing 23
 - closing automatically when opening new project 113
 - comparing 320
 - configuring 46
 - creating 37
 - creating on command line 347
 - creating, with New Project Wizard 112
 - directory hierarchy for, displaying 145
 - existing, opening 22
 - metrics 64
 - metrics for 241, 247, 248
 - opening most-recent project at startup 109
 - portability of 66
 - saving configuration of 47
 - storage 35
 - toolbar for 175
- Project Browser 145
- Project Browser option, View menu 145
- Project Configuration dialog 46, 225
 - Ada category 68
 - Assembly category 72
 - C# category 87
 - C++ (Strict) category 76, 87
 - C++ category 79
 - Dependencies category 62
 - File Options category 55
 - File Types category 54
 - Files category 49
 - Fortran category 89
 - History category 61
 - Java category 92
 - JOVIAL category 94
 - Languages category 48
 - Metrics category 63
 - Pascal category 96
 - Portability category 66
 - Python category 98
 - Reports category 56
 - saving configuration 47
 - Search category 62
 - Visual Basic category 74
 - Web category 100
- Project Graphs option, Graphs menu 229
- Project Interactive Reports option, Reports menu 229
- Project menu 360
 - Analyze All Files option 102
 - Analyze Changed Files option 102
 - Browse Architectures option 211
 - Configure option 46
 - Overview option 247
- Project Metrics report 241
- Project Overview 247
- Project overview 246
- Prompt before closing the current project field, User Interface Alerts options 113
- Prompt for missing include files field 83
- Prompt for missing includes field, C++ Includes 83
- Prompt on parse errors field
 - Ada 70
 - Fortran 91
- \$Prompt variables 332, 333, 334
- Protected Members menu 281
- Public Members menu 281
- pushpin icons 19
- Python
 - API 343
 - API for custom reports 33
 - configuration for 98
 - versions supported 15
- Python API Documentation option, Help menu 17, 33, 246, 343
- Python category, Project Configuration dialog 98

Q

- quality reports 237
- question mark (?)
 - in regular expressions 171
 - wild card 171
- quiet_startup option, understand command 355
- quote (") prefixing octal constants or string literals 90
- quotes ("") surrounding normal includes 83

R

- Radar Scroll Bar field 124
- RAM recommended 16
- Random Edge Color menu 281
- read-only access 55
- Recent files most recently use list field, User Interface Lists options 112
- Recent projects most recently use list field, User Interface Lists options 112
- Recent Projects option, File menu 22
- Record Macro option, Tools menu 198
- rectangular area, copy and paste 182
- rectangular text selection 182
- red project file icon 180
- refactoring 185
 - Extract Function 190
 - Extract Temp 193
 - Ignore Violations 307
 - Inline Function 189
 - Inline Scope 192
 - Inline Temp 192
 - Rename 186, 187
- reference files, C# 87
- References category, C# 87
- References menu 281
- References, in Information Browser 143
- regular expressions
 - book about 172
 - in filters for Entity Locator 171
 - in Find dialog 176
- Reindent Selection command 197
- Related Classes menu 282
- relationship 20
- relative portability mode 66
- Rename Architecture menu option 213
- rename declarations
 - displayed in graphical views 281
 - reports about 236
- Rename entity 186, 187
- Rename files 187
- Renames menu 281
- Renames report 236
- Renesas project 38
- Replace in Files area 166
- Replace in Files option, Search menu 166
 - vs. Refactor > Rename 186
- Replacement Text category
 - C++ Includes 84
 - Fortran Includes 91
- Replay Macro option, Tools menu 198
- reports 225
 - anti-virus software, turning off while generating reports 226
 - canceling generation of 226
 - categories of 228
 - CodeCheck compliance 310
 - colors in 226
 - configuring 225
 - cross-reference reports 230
 - customizing with APIs 33
 - customizing with Perl or C 229, 246
 - entity name format in 57
 - filenames for 227
 - generating 225, 226
 - generation time of 226
 - HTML output 57, 226
 - HTML output, entity index for 227
 - list of, choosing from 58, 225
 - metrics reports 241
 - quality reports 237
 - structure reports 234
 - text output 57, 226
 - viewing 227
- Reports category, Project Configuration dialog 56
 - Options subcategory 56
 - Output subcategory 225
 - Selected subcategory 58, 225
- Reports menu 361
 - Configure Reports option 225
 - Generate Reports option 226
 - Project Interactive Reports option 229
 - View Reports option 227
- resetlicenseconfig option, understand command 355
- Returns menu 282

RFC (Count of All Methods) metric 242

Right-click Menu

- external tools available in 333, 338
- Find in Files option 163

right-clicking 19, 135

- + or - sign in Information Browser tree 140
- on Analyze icon 102
- anywhere in Understand 24
- in Architecture Browser 214
- bold heading in Information Browser 141
- creating windows by 24
- on entities to display source 142
- on entities in Entity Locator 27
- on entities in Entity Locator entities 170
- on entities in Information Browser 25
- on entities in Source Editor 24, 30
- in Entity Filter 25
- in Entity Locator 168
- on Entity Locator column headers 169, 171
- in Information Browser 144
- on Selector window 174
- reusing windows by 24, 135
- on selected text 182, 196
- in Source Editor 183
- on Source Editor tabs 201
- See also* Ctrl+right-click

RO (read-only) indicator, in Source Editor 181

Root filters, Entity Filter 139

Routines menu

- functions displayed in 282

routines. *See* functions; procedures

row colors, alternating 115

Run Command option, Tools menu 342

RW (read-write) indicator, in Source Editor 181

S

SARIF files 304

Save all modified editor windows when application loses focus field, General options 109

Save All option, File menu 184

Save comments associated with entities field

- Ada 70
- C 78
- C++ 81
- Fortran 91
- Java 93

- Python 99
- Web 101

Save duplicate references field, C++ 81

Save macro expansion text field

- C++ 81

Save on command field, User Interface Alerts options 113

Save on parse field, User Interface Alerts options 113

Save option, File menu 184

saving edits automatically 109

Scope Information, Contextual Information Sidebar 177

Scope List option, View menu 180

Scope List, Source Editor 180

- See also* Structure Browser, Source Editor 177
- toolbar for 175

scripts

- CodeCheck 312
- Perl 20

scroll bar highlights 124, 176, 179

Search category, Project Configuration dialog 62

Search for include files among project files field, C++

- Includes 83

Search Level menu 282

Search menu 160, 359

- Find Entity option 168
- Find in Files option 28, 163
- Find option 176
- Go to Matching Brace option 195
- Instant Search option 161
- Replace in Files option 166

Search Paths Category, Pascal 97

Search Results window 28

searching

- annotations 204
- files 160, 163
- graphical views 264

SEI Cert standard 291

Select All option, Edit menu 182

Select Block menu option 195

Selected subcategory, Reports 58, 225

Selector area 174

sharing

- annotations 204
- architectures 219
- CodeCheck configurations 295
- projects 35, 66

sharp sign (#) comment in command line file 354

-
- shortcut commands 116
 - Show Page Guide field, Editor options 122
 - Show selection field 124
 - Show standard library files field, Analyze options 119
 - Show tabs field, User Interface options 111
 - Show the Getting Started dialog on startup field,
 - General options 109
 - Show Welcome Page option, Help menu 23
 - Simple Invocation Tree report 235
 - Simple With Tree report 236
 - SlickEdit editor 130
 - sliding frames 19
 - SlowConnect option, understand command 355
 - Soft Wrap option, View menu 198
 - Sort menu
 - sorting entities in 283
 - Sort Selection command 198
 - sorting entities 137
 - Sound beep field 114
 - Sound beep on analysis completion field, Analyze options 114
 - Source Editor 30, 179
 - auto-complete options 125
 - auto-indent options 125, 126
 - bookmarks, creating 199
 - bookmarks, navigating 199
 - bracket matching in 195
 - Browse Mode in 182
 - case, changing in 196
 - closing files 184
 - colors in, customizing 127, 179
 - colors in, default 128
 - commenting and uncommenting code 196
 - configuring 121
 - Contextual Information Sidebar (CIS) in 177
 - copying to clipboard from 182
 - creating files 199
 - displaying by right-clicking on entities 142
 - displaying from Find Results window 28
 - displaying from Search Results window 165
 - external editor replacing 129
 - folding (hiding blocks in) code 196
 - hiding inactive lines in 196
 - history of locations visited in, moving through 173
 - jumping to specific line number 181
 - keystrokes in, list of 198
 - language, setting 181
 - line numbers displayed in 179
 - list of specific structures in 177, 180
 - location of 21
 - macros of editing changes, recording and replaying 198
 - moving between windows of 30
 - opening files 199
 - printing from 123, 208
 - read-write (RW) and read-only (RO) indicators 181
 - right-clicking in 24, 30, 183
 - right-clicking tabs in 201
 - saving files 184
 - Scope List in 180
 - searching and replacing text in source files 176
 - searching in source files 160
 - status bar in 181
 - status icon in 180
 - tabs in, controlling behavior of 201
 - toolbar for 175
 - source files
 - adding to project 49, 146
 - adding to project on command line 349
 - analyzing using projects. *See* project
 - closing 172, 184
 - creating 199
 - deleting from project 51
 - directories for, adding to project 50
 - directories for, deleting from project 51
 - displaying by double-clicking entity 142
 - displaying by right-clicking on entities 142
 - displaying from Find Results window 28
 - displaying from Search Results window 165
 - editing. *See* Source Editor
 - encoding for 55, 122
 - excluding from source list 51
 - external editor for 129
 - imported, report about 236
 - include files specified as, assembly 73
 - include files specified as, C++ 82
 - include files specified as, Fortran 91
 - line endings for 122
 - list of, generating from command line 354
 - listing in Filter Area 26
 - metrics about 243
 - moving between windows of 30
 - opening 199
 - overriding settings for 53

- portability of 66
 - printing 208, 286
 - printing, configuration for 123
 - read-only access for 55
 - removing from project 146
 - saving 184
 - searching 28, 160, 163
 - Sources tab, Project Configuration dialog 49
 - sourcestyles.css file 226
 - spaces, converting tabs to 122
 - Spacing menu 283
 - special member functions, C++ 80
 - spell checking
 - options 126
 - using 202
 - split
 - Source Editor 197
 - workspace 173
 - workspace, toolbar for 175
 - Spring Framework 92
 - SQL
 - embedded in Pascal 96
 - files 50
 - Sql menu 283
 - square brackets. *See* brackets
 - src.jar file 93
 - src.zip file 93
 - Standard field, Ada 69
 - standard libraries
 - Ada, directory for 69
 - displaying in Analysis Log 119
 - excluding 62
 - Pascal, paths for 97
 - Standard Library Paths category, Pascal 97
 - standards, CodeCheck 292
 - Start menu, Understand commands in 22
 - startup
 - Getting Started dialog displayed on 109
 - statement count metrics 63
 - static functions, displayed in graphical views 283
 - Static menu 283
 - statistical usage reporting 16
 - status bar, Source Editor 181
 - status icon, in Source Editor 180
 - status line 21
 - Sticky box 173
 - strict C/C++ analyzer 76, 87
 - strict complexity, count and/or operators in, Ada 69
 - Strict metric 244
 - strings
 - checking spelling 202
 - regular expressions 171
 - searching 160
 - spell checking 202
 - wildcards 171
 - Structure Browser, Contextual Information Sidebar 177
 - structure reports 234
 - structure views 32
 - See also* declaration views; graphical views
 - Styled Labels menu 283
 - Styles category, Editor options 127
 - subprograms, listing in Filter Area 26
 - subtraction sign (-) collapsing tree in Information Browser 140
 - Suggestions view 104
 - support contact information 15
 - SVG format, saving graphical views as 285
 - switches, command line
 - understand command 355
 - synchronizing Information Browser 142
 - Sysroot field, C++ Includes 82
 - system include files, C++ 83
- ## T
- tab, automatic 125, 126
 - tabs for windows, displaying 111
 - tabs, converting to spaces 122
 - Tcl files 50
 - technical support contact information 15
 - temporary bookmark 200
 - text
 - checking spelling 202
 - comparing 320
 - copying to clipboard 182
 - favorite 158
 - files 50
 - generating reports as 57, 225, 226
 - selecting 182
 - viewing reports as 227
 - Text Comparison window 320
 - Text menu, graphical views 283
 - TextPad editor 130
 - themes for graphs 275
 - title areas, filenames in 112

- title bar 19
- Title Formats field, User Interface options 112
- Toggle Overtyping option, Edit menu 198
- tool commands, importing and exporting 341
- Tool Configurations dialog
 - Popup Menu category 338
 - Toolbar category 340
- toolbar
 - external tools available from 340
 - hiding and displaying icons 175
 - location of 21
 - visibility of icons in, controlling 340
- Toolbar category, Tool Configurations dialog 340
- Tools menu 362
 - Editor Macros option 198
 - Options option 108
 - Run Command option 342
 - User Tools option 331, 333, 339, 341
- tools, external. *See* external tools
- tooltips 129
- Treat system includes as user includes field
 - C++ Includes 83
- Tree row indentation field, User Interface options 115
- treemap
 - CodeCheck 299
 - metrics 253
- Truncate column field
 - Fortran 90
 - JOVIAL 94
- Truncate Long text option 283
- Truncate Medium text option 283
- Truncate Short text option 283
- truncation at column
 - Fortran 90
 - JOVIAL 94
- Turbo Pascal 96
- Type Cross-Reference report 232
- types
 - displayed in graphical views 284
 - listing in Filter Area 26
 - reports about 232, 239
- Types menu, graphical views 284
- TypeScript
 - file extensions 100
- TypeScript files 15
- Typetext menu 284
- Typetext menu, graphical views 284

U

- .udb file extension 20, 35
- UML Sequence Diagram 262
- Uncomment Selection menu option 196
- uncommitted lines 124
- und command 346, 347
 - adding files to project 349
 - analyzing a project 353
 - options in latest version, listing 349
- UndCC codes 309
- undefined macros, C++ 86
- Undefines category, C++ Macros 86
- Understand
 - compared to compiler 35
 - contact information 15
 - features of 14
 - multiple users for 23
 - online help for 17
 - starting 22
 - starting from command line 355
 - windows in 19, 21
- understand command 355
- Understand Options dialog 108
 - Analyze category 119
 - Dependency category 120
 - Editor category 121
 - General category 109
 - Graphs category 130
 - Key Bindings category 116, 198
 - User Interface category 111
- undocking windows 19
- Unicode file handling, in comparison 329
- Uninitialized Items report 238
- Unix
 - line termination style, for reports 226
 - line termination style, for saving source files 122
- UNIX support 16
- unknown entities, displaying in graphical views 284
- Unknown menu, graphical views 284
- unresolved entities
 - displaying in graphical views 284
 - listed in Entity Locator 27
- Unresolved menu, graphical views 284
- unsupported languages 100
- Unused Object report 239
- Unused Objects and Functions report 238

Unused Program Unit report 240
 Unused Type report 239
 Update Information Browser field, Browse Mode Editor options 129
 uperl command 20
 .upl file 287, 343
 .upy file 287, 343
 upython command 20
 usage reporting 16
 Use alternating row colors field, User Interface options 115
 Use case-insensitive lookup for includes field
 C++ Includes 83
 Use Class Grouping menu 284
 Use default working directory field, General options 109
 Use include cache field, C++ 81
 Use preprocessor field, Fortran 90
 use statements, reports about 240
 Use the New Project Wizard when creating new projects field, Configure options 112
 Usedby Menu, graphical views 284
 User Interface category, Understand Options dialog 111
 user interface, parts of 21
 User Tools option, Tools menu 331, 333, 339, 341
 users, multiple
 CodeCheck configurations 295
 initialization files for 23
 sharing annotations 204
 sharing architectures 219
 sharing projects 35, 66
 Uses Menu, graphical views 284
 Uses Not Needed report 240

V

variables
 displayed in graphical views 284
 instance 242
 listed in Entity Locator 27
 metrics about 242
 reports about 232, 238, 239
 uninitialized, report about 238
 unresolved 27
 See *also* objects
 Variables menu, graphical views 284
 VDX format, saving graphical views as 285
 Verilog files 50
 Version field
 Ada 68
 Fortran 89
 Java 92
 JOVIAL 94
 Pascal 96
 vertical bar (|) in regular expressions 172
 Vertical Non-Crossing layout option 280
 VHDL
 terminology 99
 versions supported 15
 vi editor 130
 View menu 360
 Analysis Log option 103
 Bookmarks option 174, 200
 Browse Mode option 182
 Contextual Information option 177
 Entity Locator option 27, 168
 Favorites option 28, 156
 Hide Inactive Lines option 196
 Information Browser 139
 Project Browser option 145
 Scope List option 180
 Soft Wrap option 198
 Window Selector option 174
 View Reports option, Reports menu 227
 views. See declaration views; graphical views; hierarchical views
 Violation Browser 302
 violations
 checking for 291
 highlight in scroll bar 124
 ignoring 305
 metrics 304
 Virtual Calls menu 284
 Visio files, saving graphical views as 285
 -visit option, understand command 355
 Visit Source field, Browse Mode Editor options 129
 Visual Basic
 configuration for 74
 support 15
 Visual Basic category, Project Configuration dialog 74
 Visual C++, as editor 129
 Visual Studio
 as external editor 130
 auto-architecture 212

VSDX format, saving graphical views as 285

W

-wait option, understand command 355

warnings, displaying from Analysis Log window 103

watched directories

setting 51, 52, 53

Web category, Project Configuration dialog 100

web languages 15

metrics separate for 63

Web support 15

websites

external editor information 130

metrics, list of 237, 241

O'Reilly and Associates 172

Welcome page 22

white file icon 180

whitespace

in comparison 329

indicators for 123

Whitespace field, Editor options 123

wild cards, in filters for Entity Locator 171

Window menu 172, 363

Close All Document Windows option 172, 184

Close option 172, 184

Windows Navigator option 173

Window Selector option, View menu 174

windows 19, 21

animated opening and closing of 111

closing 19

creating with Ctrl+right-click 24, 135

docking and undocking 19

filenames in title area of 112

frames in, sliding 19

list of 136

list of open windows 173, 174

organizing 172

reusing for graphical views 265

reusing with right-click 24, 135

tabs for, displaying 111

Windows category, Application Styles options 115

Windows category, User Interface options 114

Windows line termination style

for reports 226

Windows line termination style for source files 122

Windows Navigator option, Window menu 173

Windows OS support 16

With Bys menu, graphical views 284

With Tree report 236

Withs menu, graphical views 284

Withs Not Needed report 240

wizard, new project 37

WMC (Count of Methods) metric 242

working directory, default 109

Wrap Long text option 284

Wrap Medium text option 283

Wrap Mode field, Advanced Editor options 124

Wrap Short text option 283

wrapping lines

display 198

printing 124

X

XML output 152

XML source files 15

Y

yellow project file icon 180