scitools

# Understand

# User Guide and Reference Manual

**Version 4.0**
**October 2015**

Scientific Toolworks, Inc.
53 N Main
St. George, UT 84770



Part Number: USTAND4.0-GEN-UG-802 (10/15)

# Contents

## Chapter 5 Searching Your Source



## Chapter 6 Editing Your Source

# Chapter 1 Introduction

This chapter introduces the *Understand* software.

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

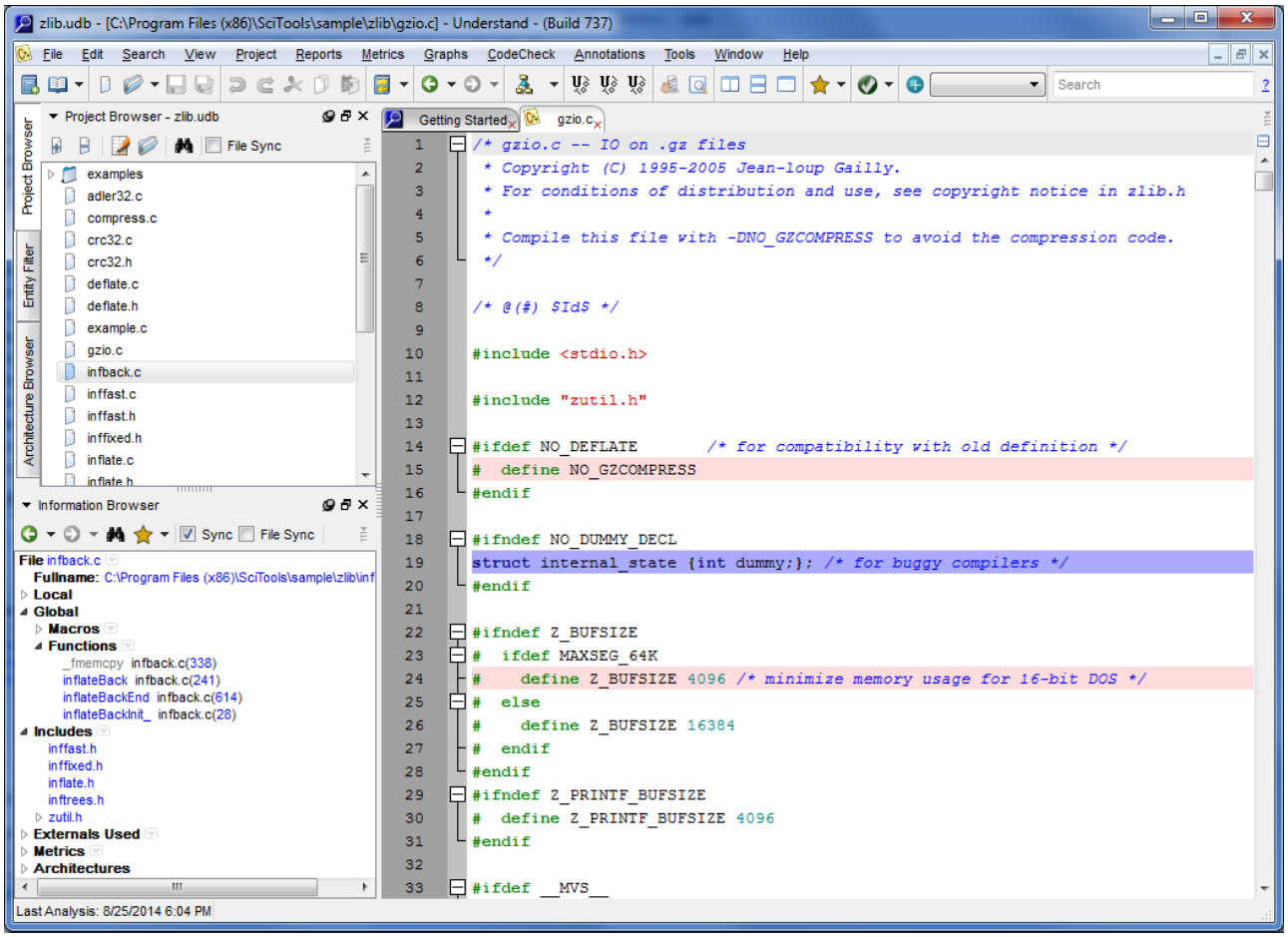
This chapter contains the following sections:

# 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, Java, Pascal/Delphi, COBOL, JOVIAL, VHDL, Fortran, PL/M, Python, PHP, HTML, CSS, JavaScript, and XML.

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




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

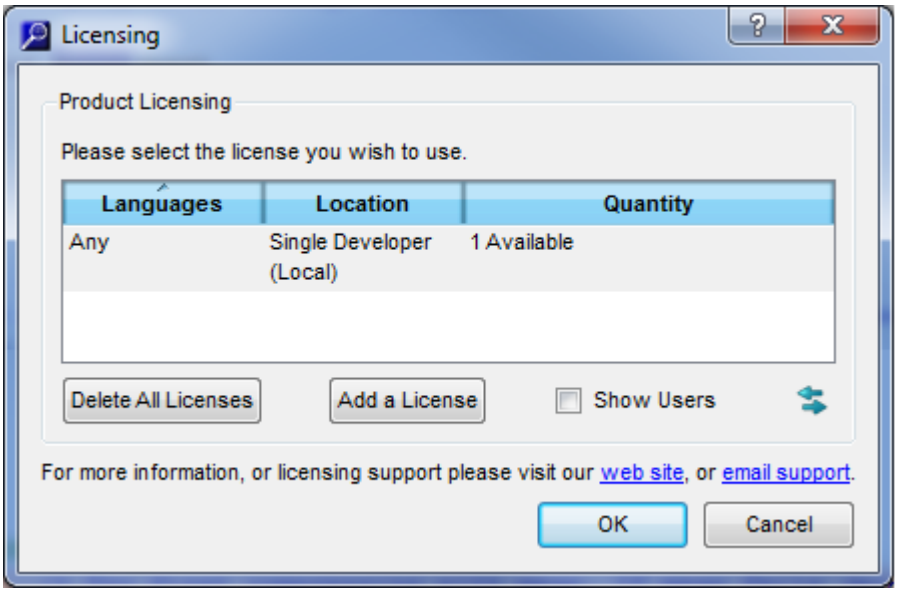
*Understand* has analysis features that help you quickly answer questions such as:

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

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

## Licensing Issues

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




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

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

See the Scientific Toolworks website for additional information about installation and licensing.

# Languages Supported

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

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

For information about support for a specific language syntax, search the Build Log on the Scientific Toolworks website (www.scitools.com/support/buildLogs.php).

# For Those Who Don't Like to Read Manuals

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

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

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

For more advanced users, try these information sources:

- Choose **Help > About Understand** to see which build you are currently running.
- See www.scitools.com/support/buildLogs.php to search through the build logs. Use the link on that page to "Sign up to receive via Email" new build notes and build announcements.
- Choose **Help > Key Bindings** for keystroke help.
- See scitools.com/support/metrics_list/ for details about specific metrics.
- 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.

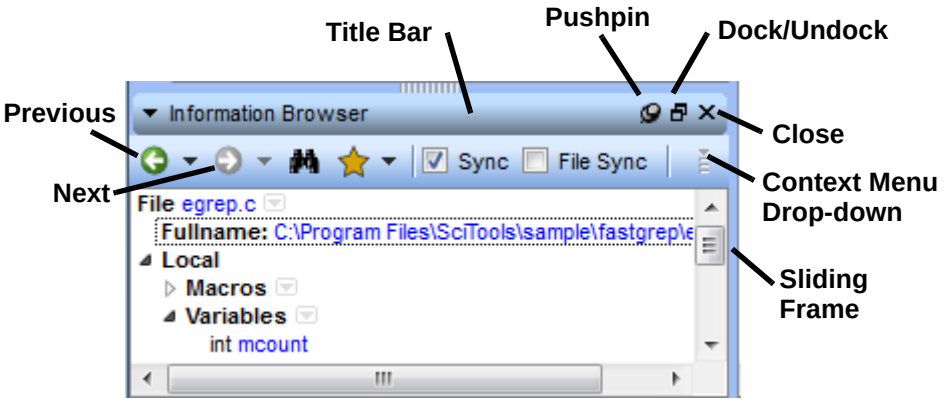
# Chapter 2 Parts and Terminology

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

This chapter contains the following sections:

# Using Understand Windows

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




- **Title Bar:** You can drag the title bar of an area around the main window. If you move to the edge of the main window, a docking area expands. If you drop the area there, it "docks" to the edge of the main window.
- **Pushpins and Drawers:** Click the  icon to move an area to a tab along the same edge of the main window to which this area was docked. This is a "drawer" that opens automatically if you point your mouse at the tab title. The drawer closes if you move your mouse away from the area without clicking on it or if you click the title tab of the currently open drawer.

  Click the  icon to "pin" a drawer open. Pinned drawers have a title bar and title bar icons like the ones shown above.
- **Dock/Undock:** Click the  icon to change the area to an undocked window. Click the icon again in an undocked window to return to a docked area.
- **Close:** Click the "X" icon to close the area or undocked window.
- **Drop-down:** Click the  icon to see the context menu for this area. Right-clicking an item within an area usually displays a context menu specific to that item.
- **Sliding Frame:** You can drag the frames between window areas to change their sizes.
- **Previous and Next:** Each area type has different icons below the title bar. For the Information Browser area shown, you can browse through the history of entities viewed. For other areas, you will see other icons.

# Understand Terminology

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

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

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

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

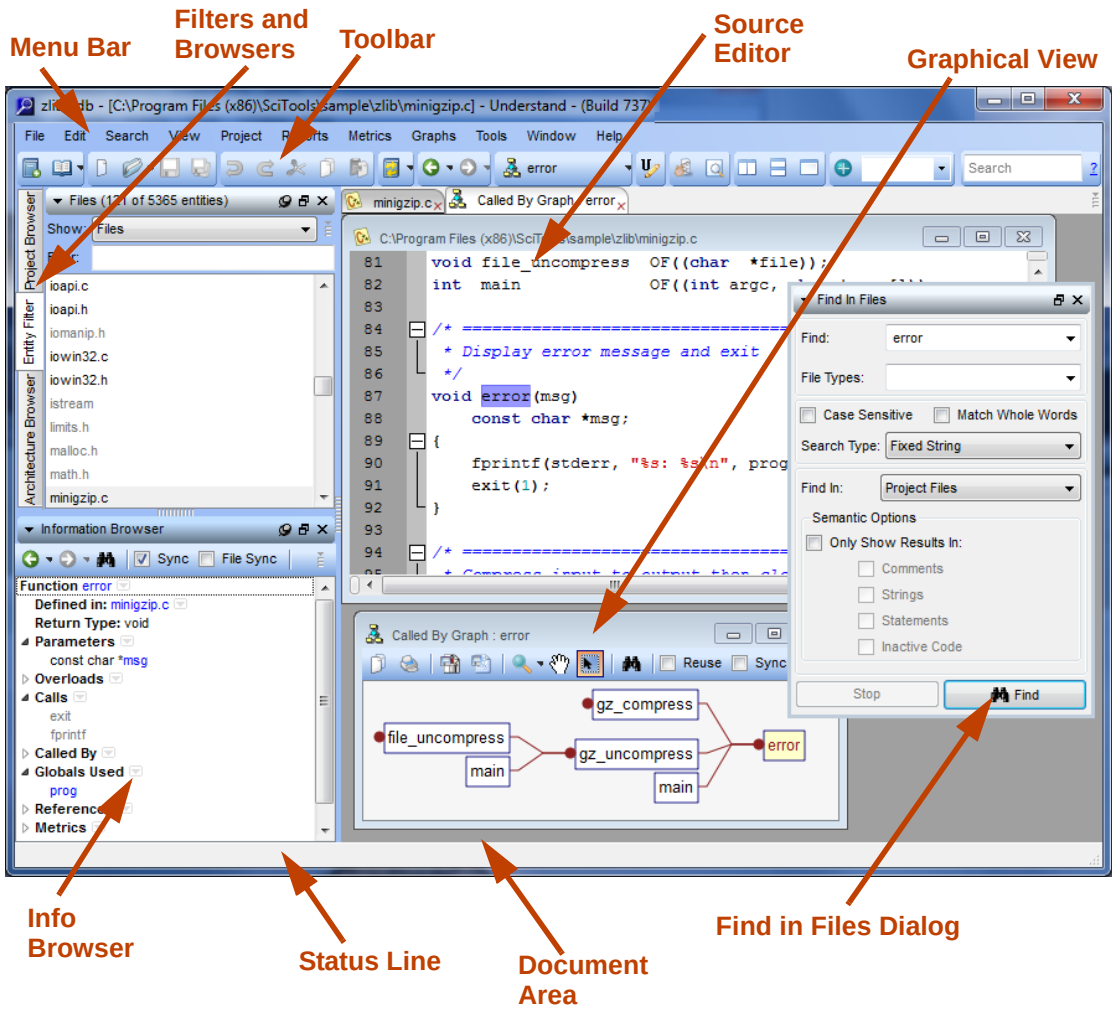
**Project:** The set of source code you have analyzed and the settings and parameters chosen. A "project file" contains the list of source files and the project settings.

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

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

## Parts

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

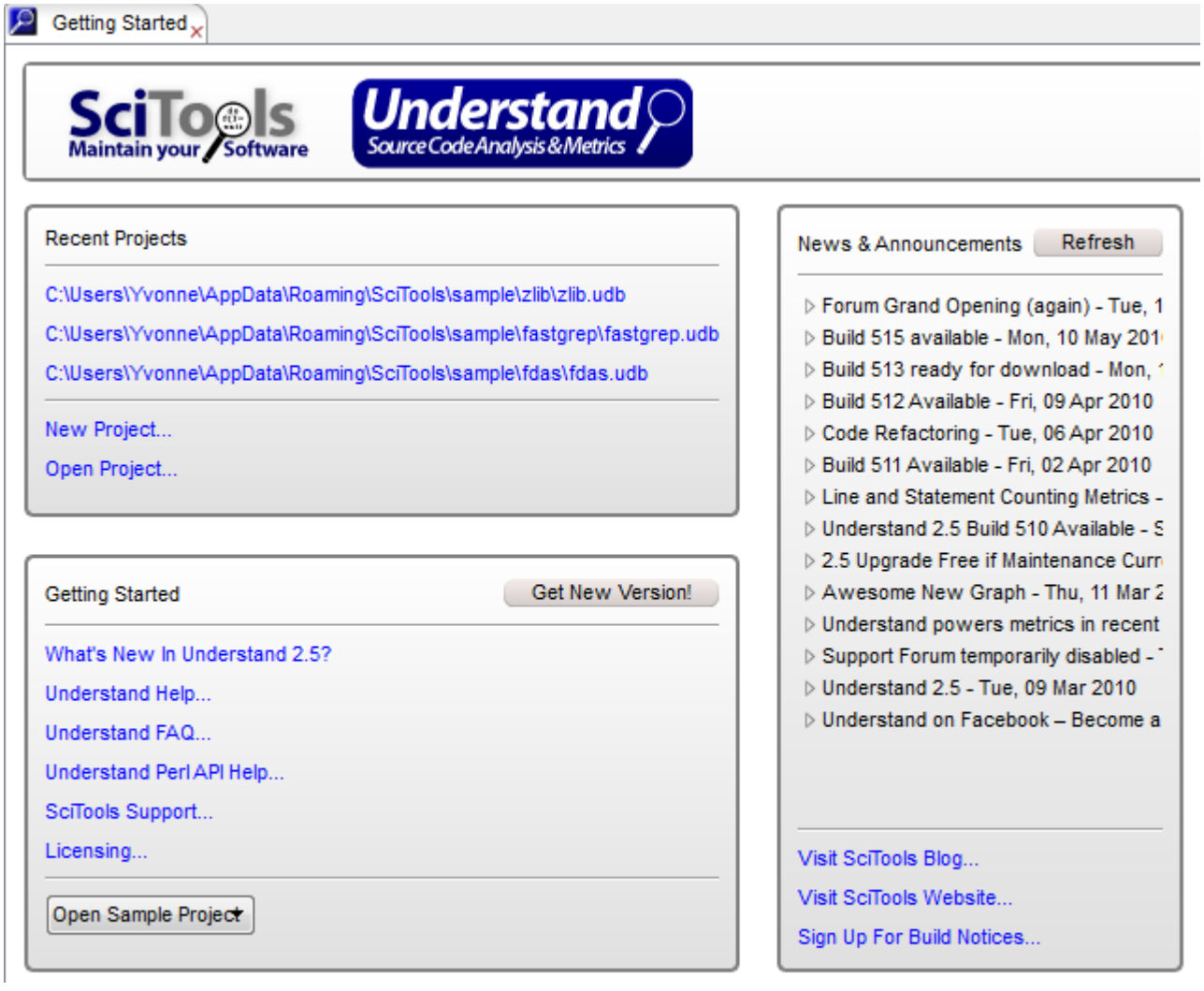


# Starting Understand

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

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

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



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

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

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

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

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

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

## Other Ways to Run Understand

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

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

- **Windows 2000/XP:** C:\Documents and Settings\USERID\Application Data\SciTools\Understand.ini
- **Windows Vista/7/8:** C:\Users\USERID\AppData\Roaming\SciTools\Understand.ini
- **Linux/Unix:** ~/.config/SciTools/Understand.conf
- **Mac OS X:** ~/Library/Preferences/com.scitools.Understand.plist

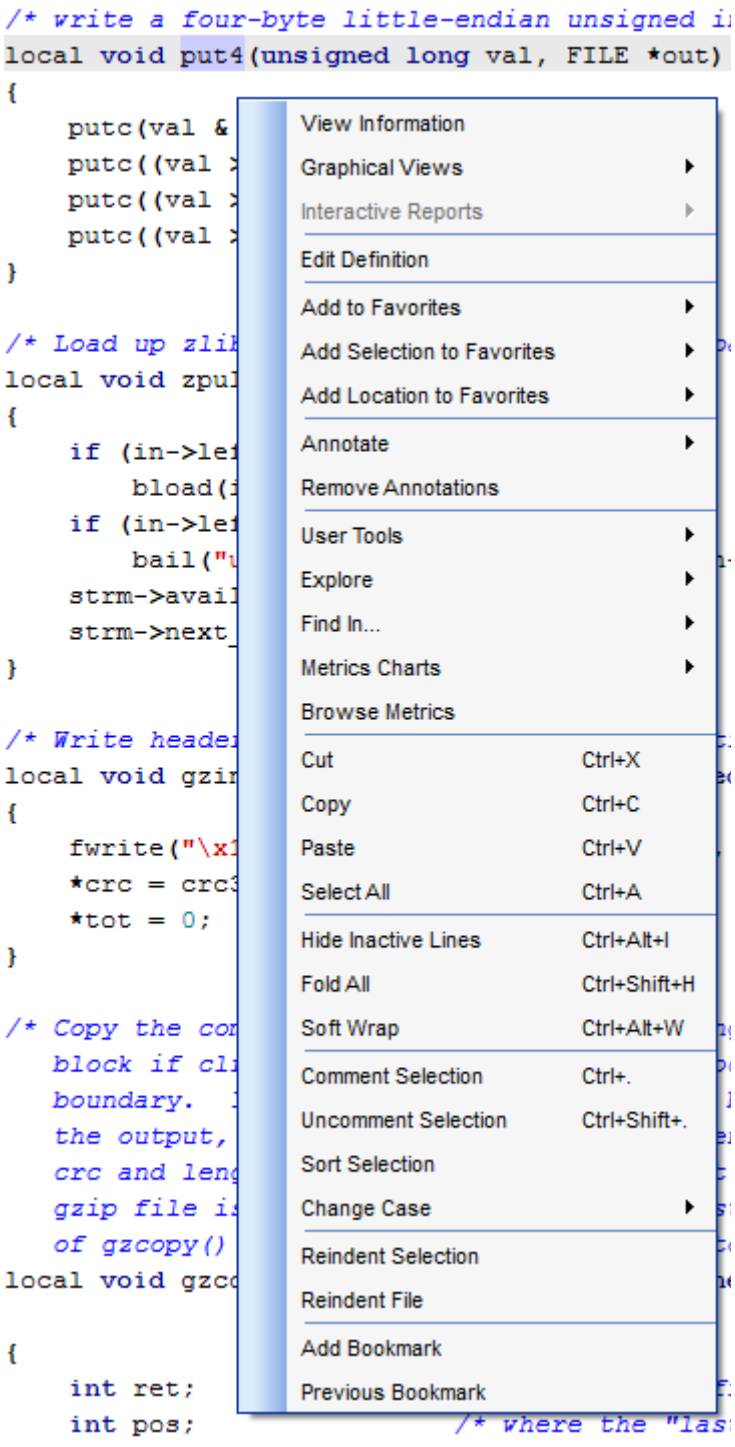
# Context Menus Are Everywhere

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

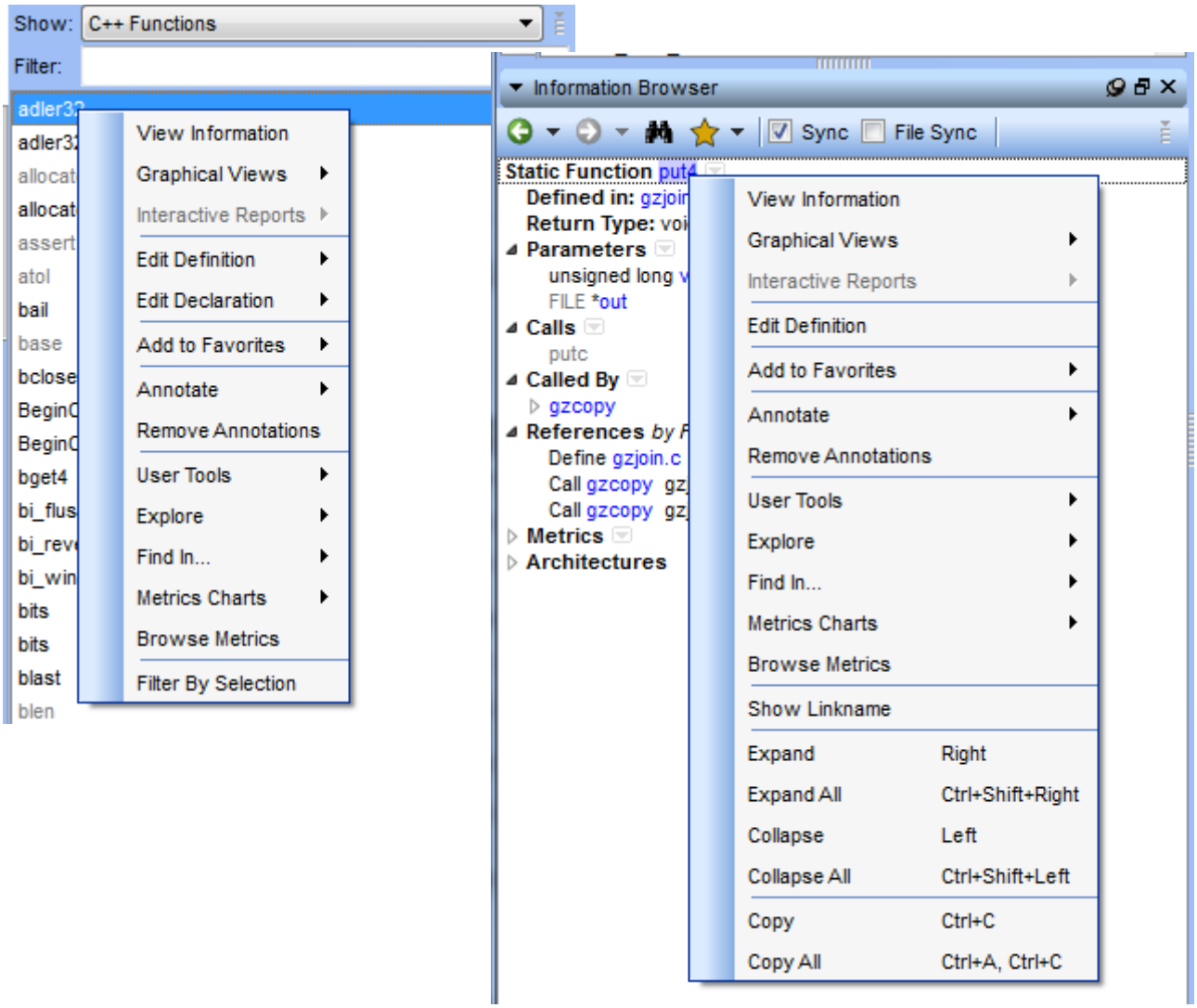
For example, if you right-click on an entity in the Source Editor, you see the list of commands shown here.

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

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



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

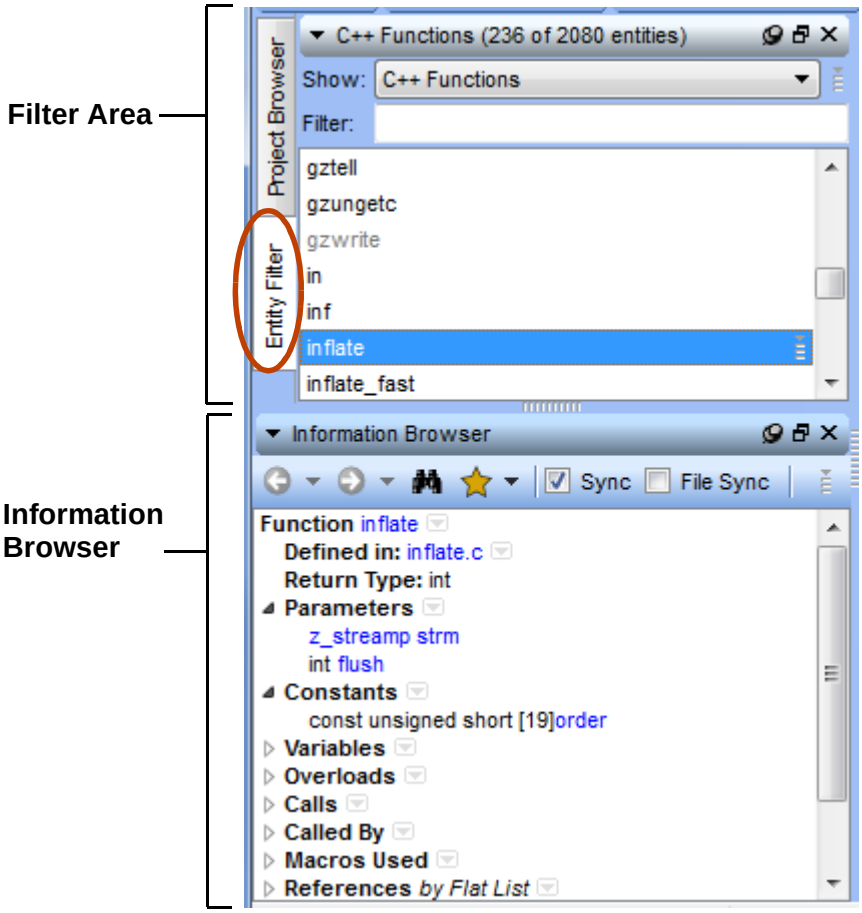
*Understand* provides several ways to quickly locate items of interest in your source code. These features include the Filter Area, the Entity Locator, and the Find in Files dialog.

## Entity Filter

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

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

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

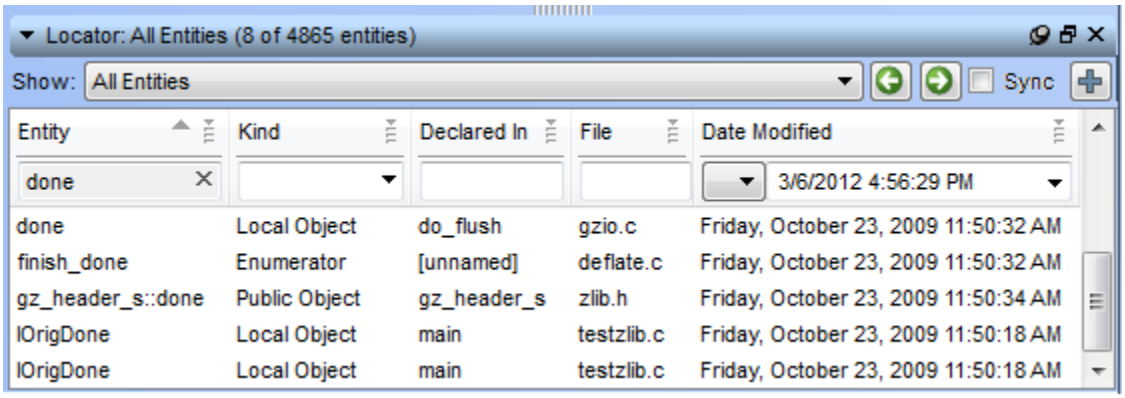



For details, see *Entity Filter* on page 129 and *Information Browser* on page 131.

## 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 database for your project, use the *Entity Locator.*

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




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

For details, see *Entity Locator* on page 155.

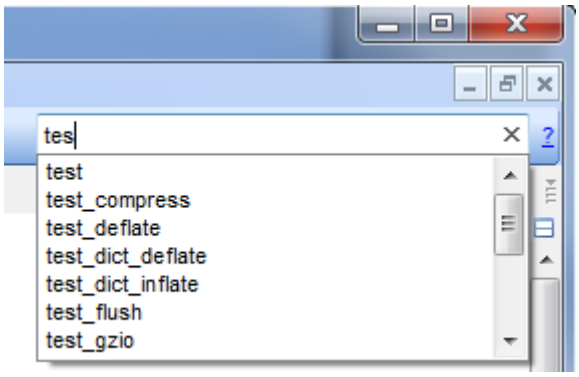
As in any other window, the context menu is also active.

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

## Instant Search

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

A number of powerful search options are supported with Instant Search. See *Instant Search* on page 148.



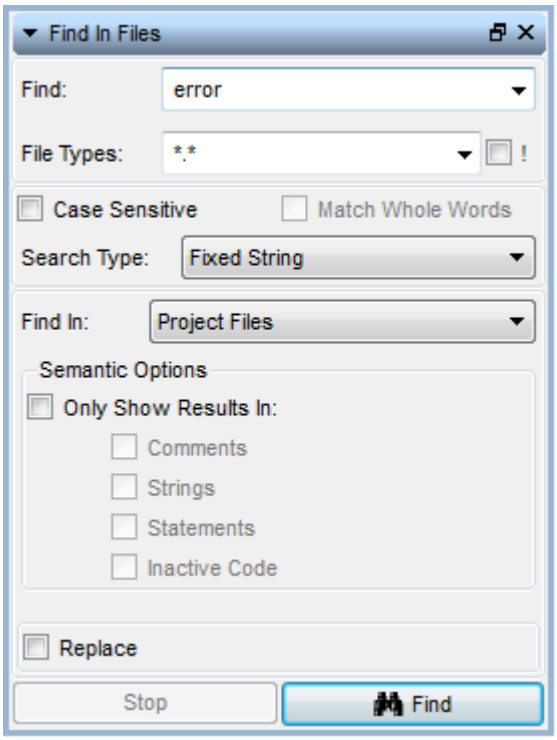
## Find in Files

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

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

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

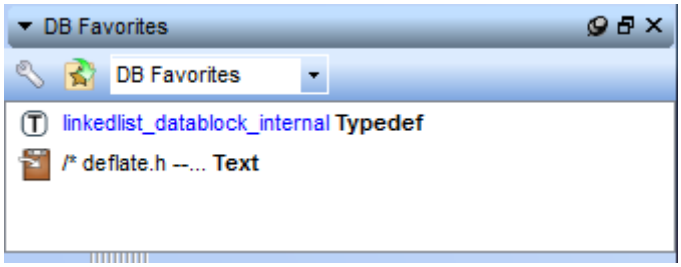
See *Find in Files* on page 150 for more information.




## Favorites

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

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




See *Favorites* on page 142 for more information.

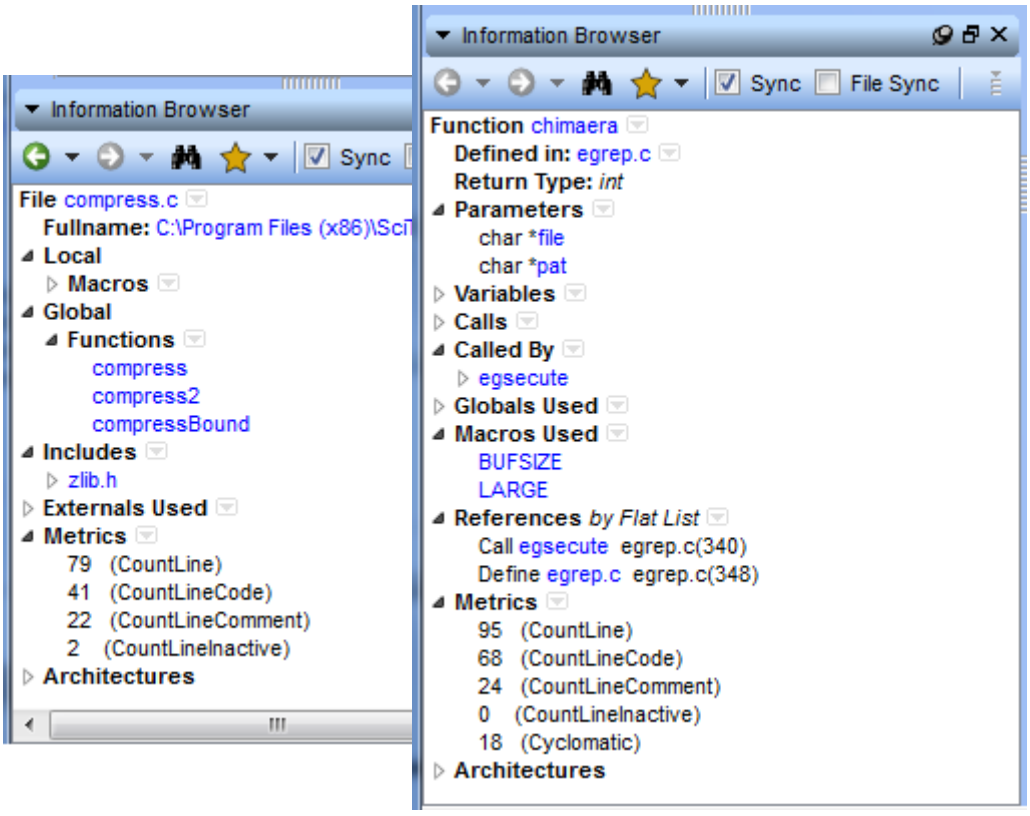
# Information Browser

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

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

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

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

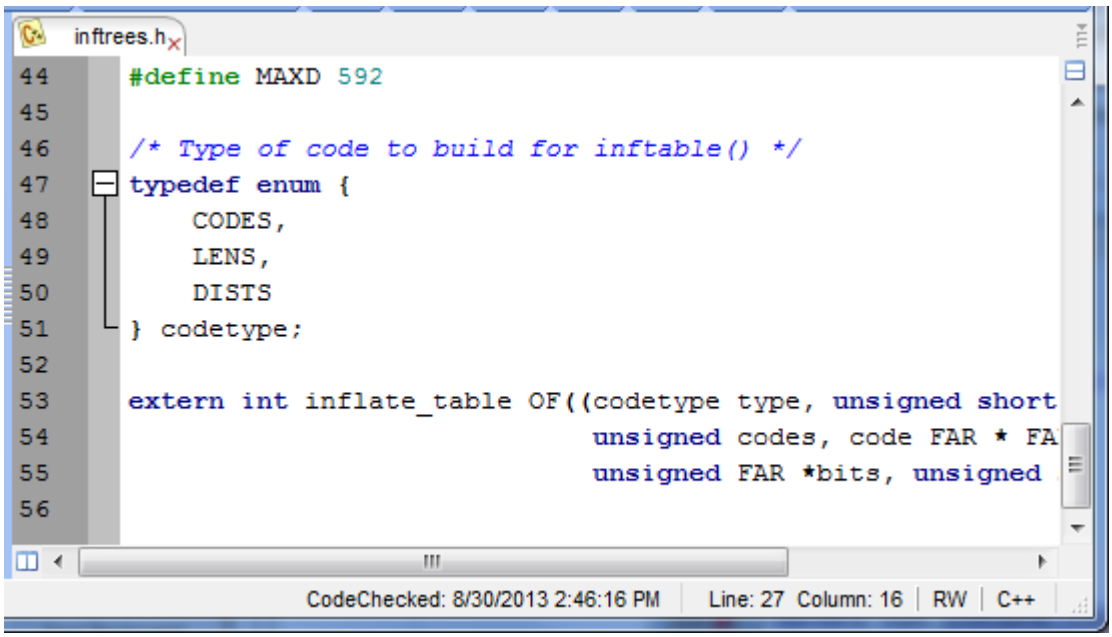



For details, see *Information Browser* on page 131.

# 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.



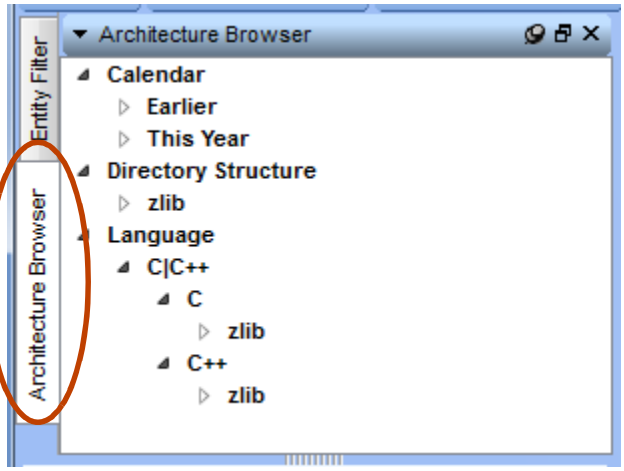

As with any other place in *Understand*, a context menu is available throughout the editor. To learn about something just right-click on it to see what information is available.

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

# Architecture Browser

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

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




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

For details, see *About Architectures* on page 192.

# Graphical Views

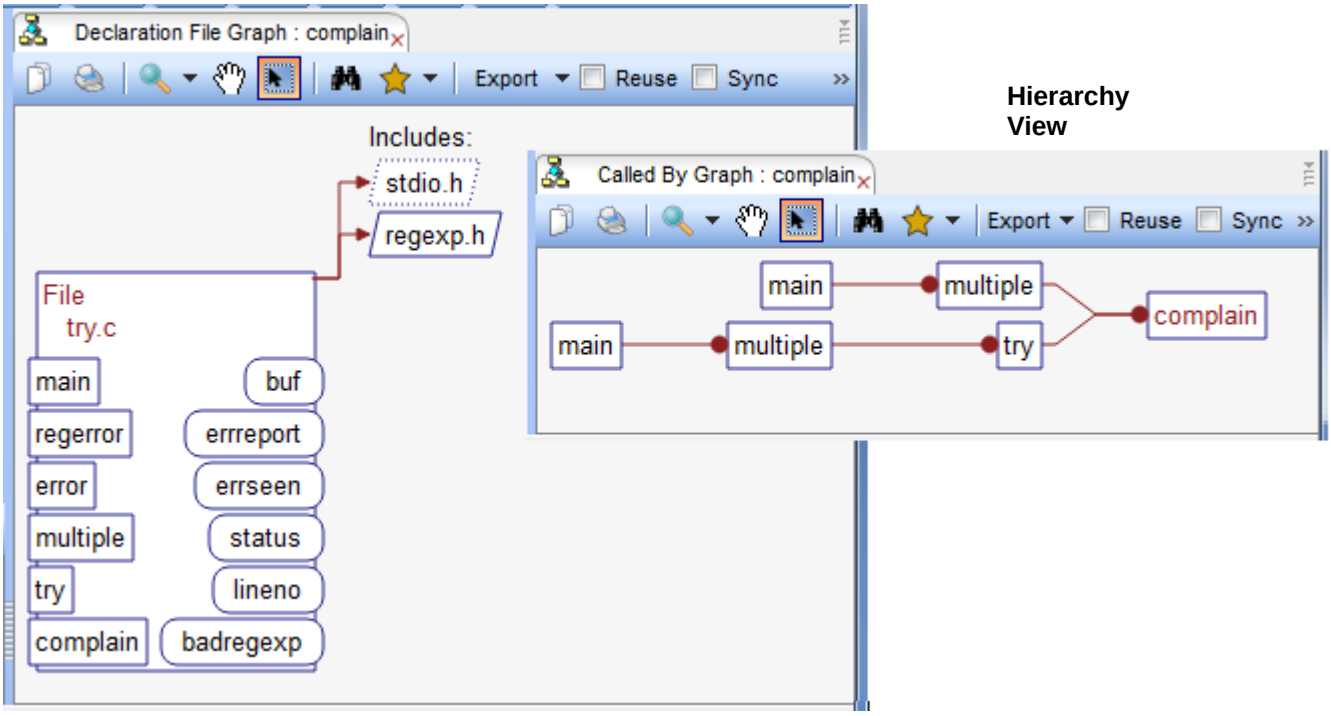
*Understand* analyzes your software code and creates a database containing information about the entities and the relations between entities. The database can then be browsed using various “graphical view” windows. The graphical views are divided into these kinds:

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

Examples of each type are shown in the following figure:

**Structure View**

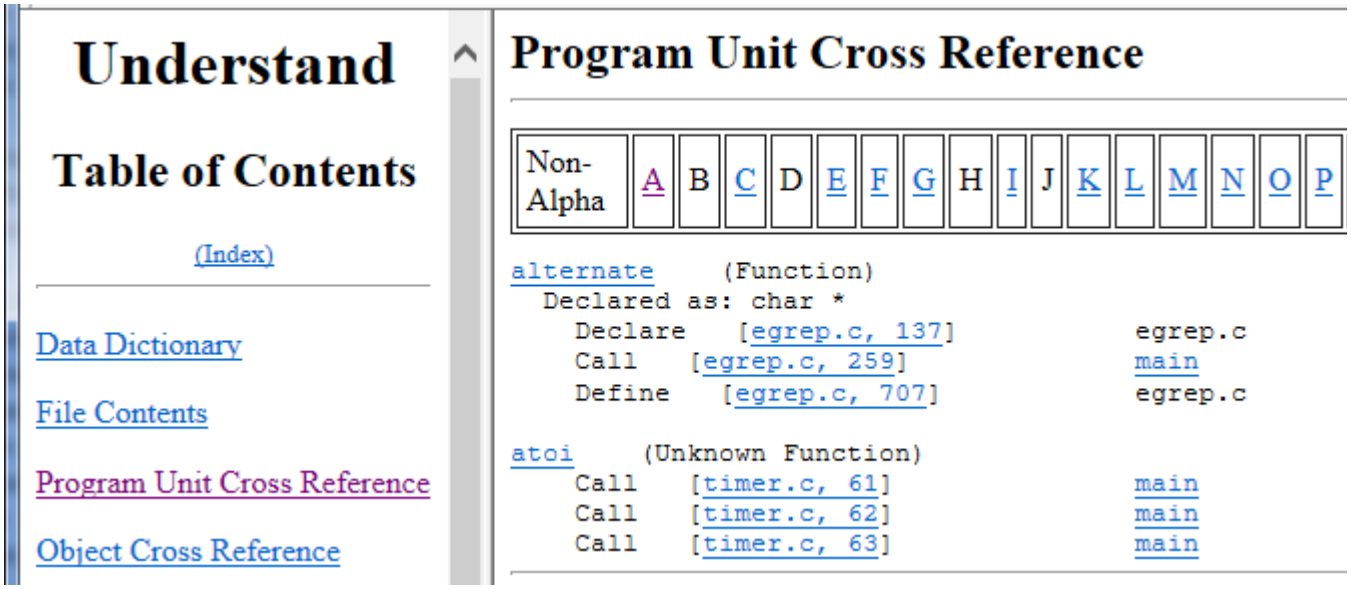
**Hierarchy View**




For details, see *Using Graphical Views* on page 245.

## ASCII and HTML Reports

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




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

See *Using Reports* on page 207 for more information.

## APIs for Custom Reporting

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

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

This manual doesn't cover the APIs. Choose **Help > PERL API Documentation** or **Help > Python API Documentation** for more information. Java API documentation is provided in the doc/manuals/java subdirectory of the *Understand* installation. Tutorials for the APIs are available online at scitools.com/api-tutorials/.

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

# Chapter 3 Configuring Your Project

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

This chapter contains the following sections:

# 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.

## The *Understand* Project Database

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

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

The project 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 Change Log. When you install a build that modifies the database format, existing projects are automatically re-analyzed when you open them.

# Creating a New Project

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

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

To create a new project, follow these steps:

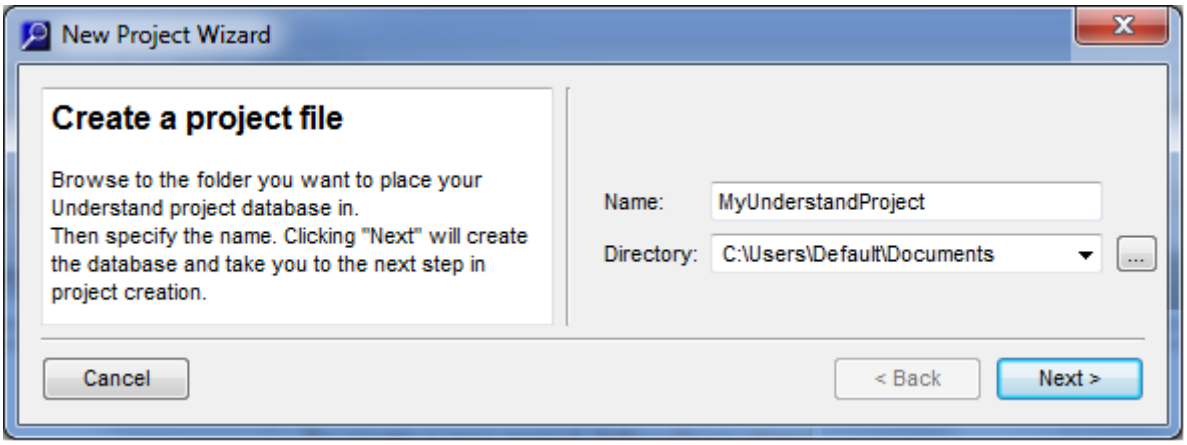
1 Click the **New Project** link in the Getting Started tab that you see when you start *Understand*. Or, choose **File > New > Project** from the menus.

- By default, this opens the New Project Wizard, which is described on page 35.
- Alternately, you may have disabled the option to run this wizard, in which case, you see the “Create new project as...” dialog. Browse to the folder where you wish to create the project database. Type the name of the project in the **File name** field. A *.udb file extension will be added automatically. Click **Save**. You will see the Understand Project Configuration dialog, which is described in page 39.
- Another way to create a project is to add Buildspy to your gcc/g++ build process. This automatically generates an *Understand* project when you compile your project. See *Using Buildspy to Build Understand Projects* on page 329.

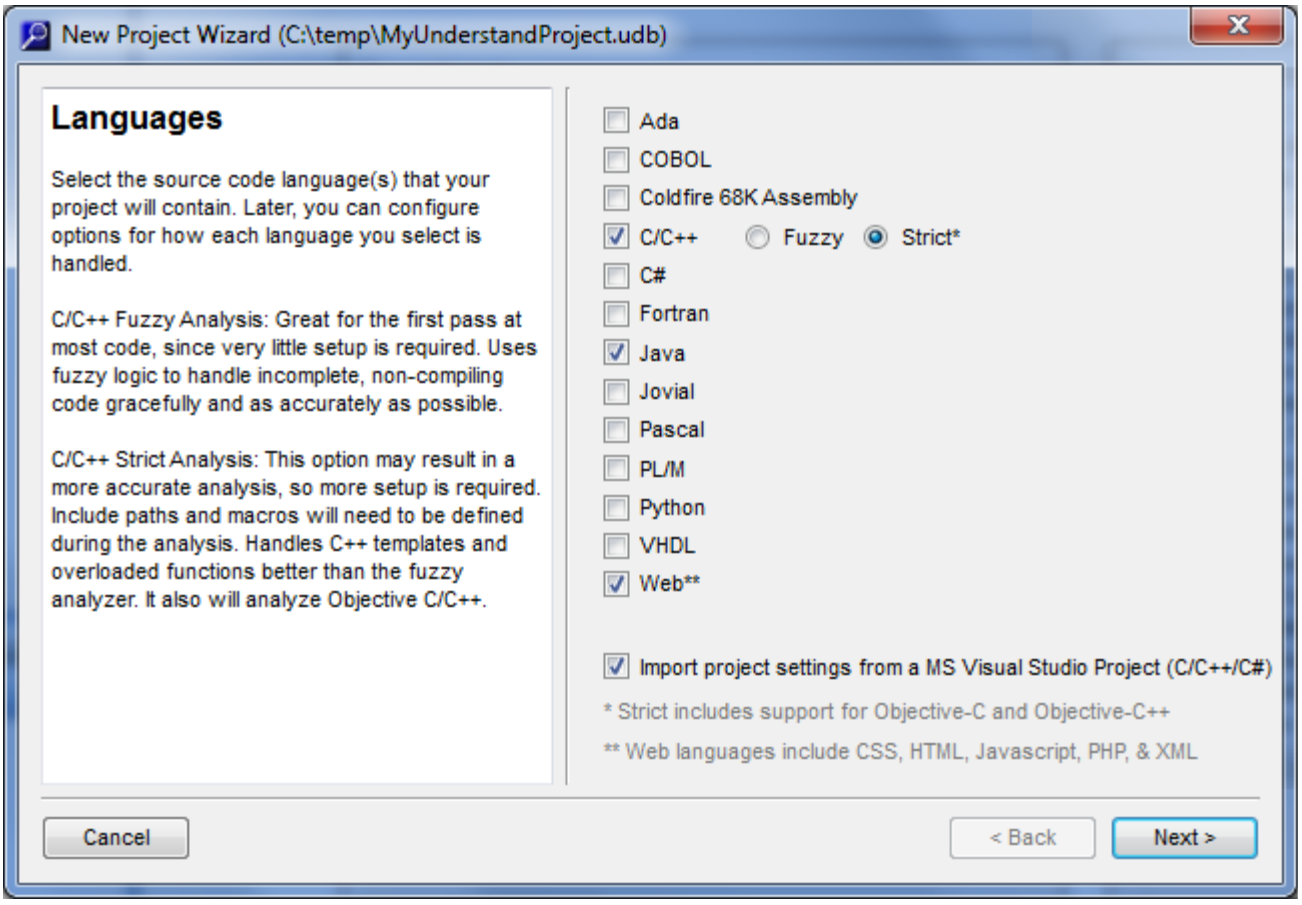
## New Project Wizard

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

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



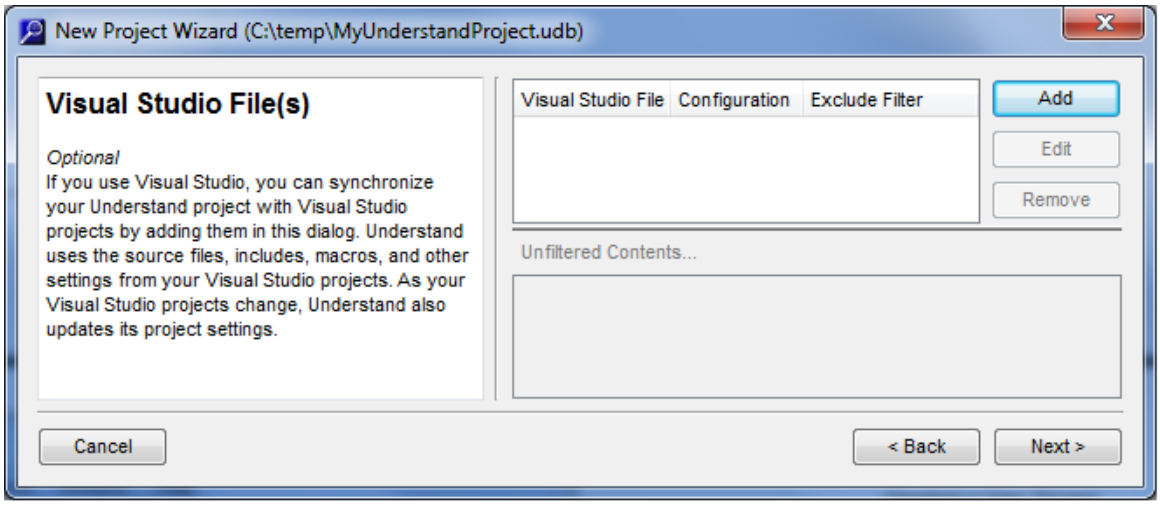
2 Click **Next** to see the Languages page of the wizard. The defaults for this and other configuration categories are the most recent settings you saved for another project.




3 Put checkmarks next to languages used in the source code for this project. See *Languages Category* on page 41 for more information about specific languages and the strict C/C++ option.

4 If you use Microsoft Visual C for your C, C++, or C# code, you can check the box to import project settings. Then click **Next**.

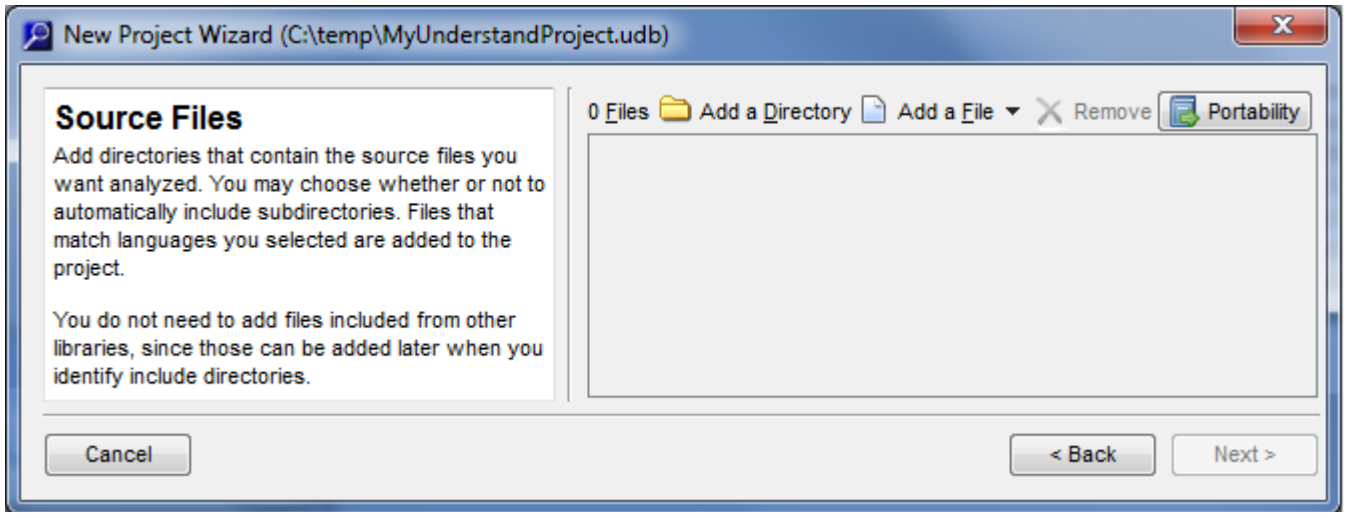
5 If you checked the **Import project settings from a MSVC Project** box, you see the Visual Studio File(s) page. Otherwise, skip to the next step.




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

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

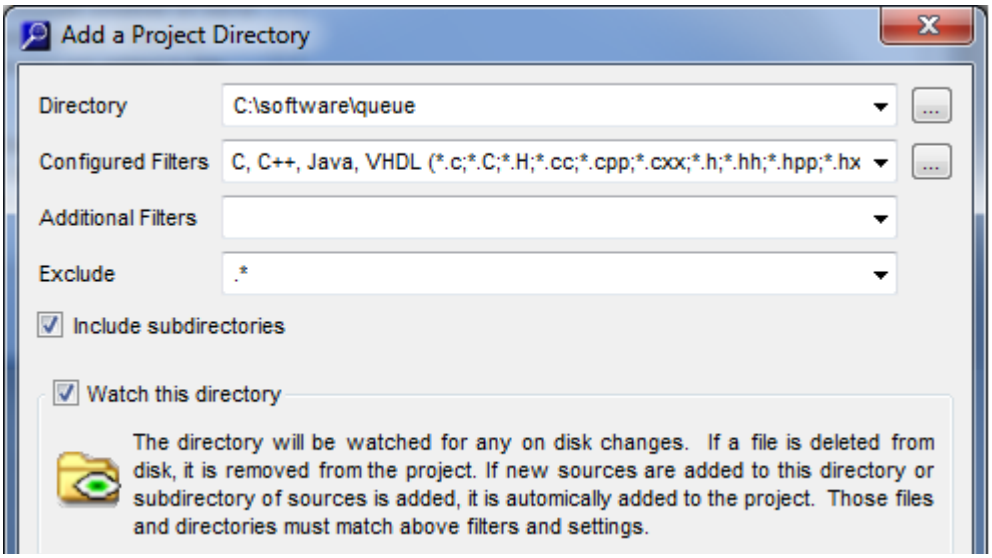
6 In the Source Files page of the wizard, you add source files to a project by clicking **Add a Directory** or **Add a File**.




The **Portability** button lets you choose whether file paths will be stored using absolute paths, relative paths, or user-defined root path variables. See page 47.

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

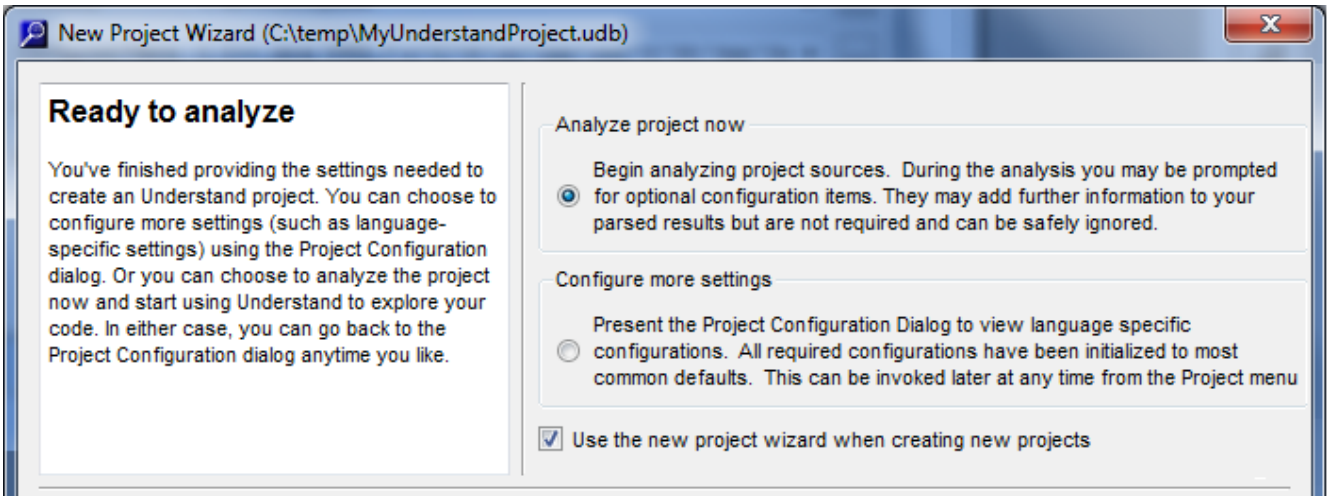
When you add a directory, you can browse for a directory, modify the list of languages used in the source files, add additional filters for file extensions not expected by *Understand*, filter out any files you want to exclude (for example, temp*.*), and choose whether all the subdirectories of this directory should be added. You can also choose whether the directory will be watched for changes. See *Adding Directories* on page 43 for details.




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

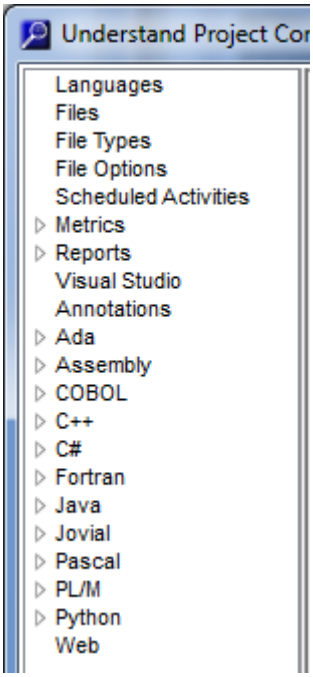
If you want to delete a file or a directory (and its subdirectories), select that item and click the “X” icon. Click **Next** when you have finished selecting files and directories.

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



# Project Configuration Dialog

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

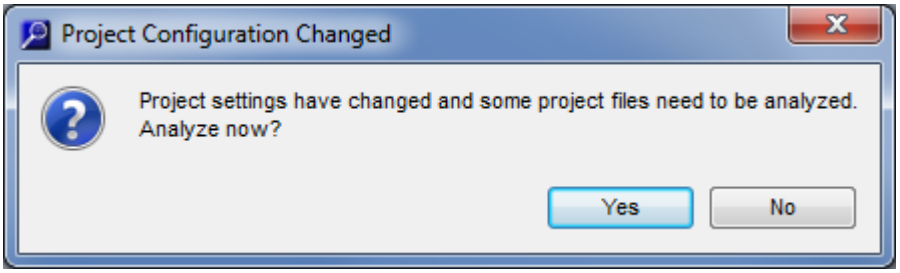



The categories on the left in the Project Configuration dialog allow you to specify various project settings to be used during analysis. The Project Configuration dialog contains the following categories:

- **Languages:** Set the types of languages to be analyzed. For details, see *page 41*.
- **Files:** Set the locations of source files to be analyzed. For details, see *page 42*.
- **File Types:** Set how to handle source file types and what file extensions are used. For details, see *page 48*.
- **File Options:** Set the file encoding and editing mode for source files. For details, see *page 49*.
- **Scheduled Activities:** Schedule events to take place at regular intervals. For details, see *page 50*.
- **Metrics:** Select the metrics you want computed for this project. For details, see *page 52*.
- **Reports:** Select reports you want generated. For details, see *page 54*.
- **Visual Studio:** Select a Visual Studio project to synchronize this *Understand* project with. For details, see *page 57*.
- **Annotations:** Set how to store and display annotations. For details, see *page 59*.
- **Language-Specific Options:** Set options for the languages you selected in the Languages category. For details, see:
  - Ada Options, *page 61*
  - Assembly Options, *page 65*
  - COBOL Options, *page 66*
  - C++ Options, *page 68*
  - C++ Strict Options, *page 75*
  - C# Options, *page 79*
  - Fortran Options, *page 81*
  - Java Options, *page 84*
  - JOVIAL Options, *page 86*
  - Pascal Options, *page 88*
  - PL/M Options, *page 90*
  - Python Options, *page 92*
  - Web Options, *page 93*

For advice about ways to adjust the project configuration to improve the accuracy of project analysis, see the SciTools website.

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



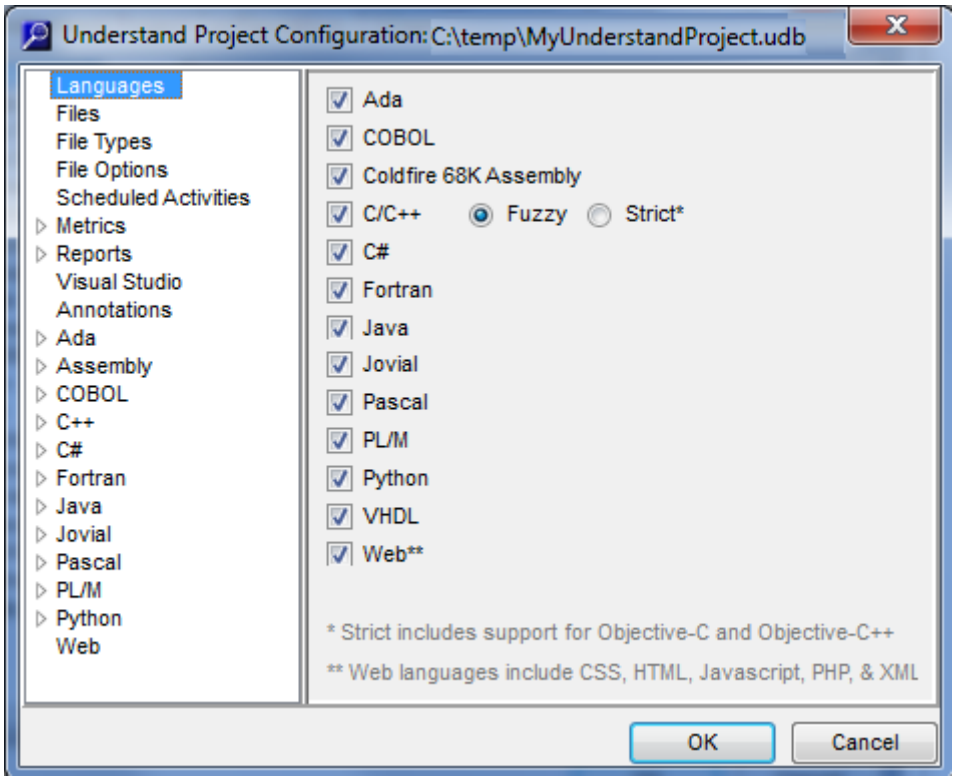

Click **OK** and *Understand* begins analyzing (also called parsing) the code (page 121).

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

If you want to make a copy of the current configuration, for example to create two variants of one configuration, choose **File > Save *project.udb* As**. This makes a copy of the *.udb file and modifies uses of the database name within the configuration as needed.

# Languages Category

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




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

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

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

The “strict analyzer” provides the following features:

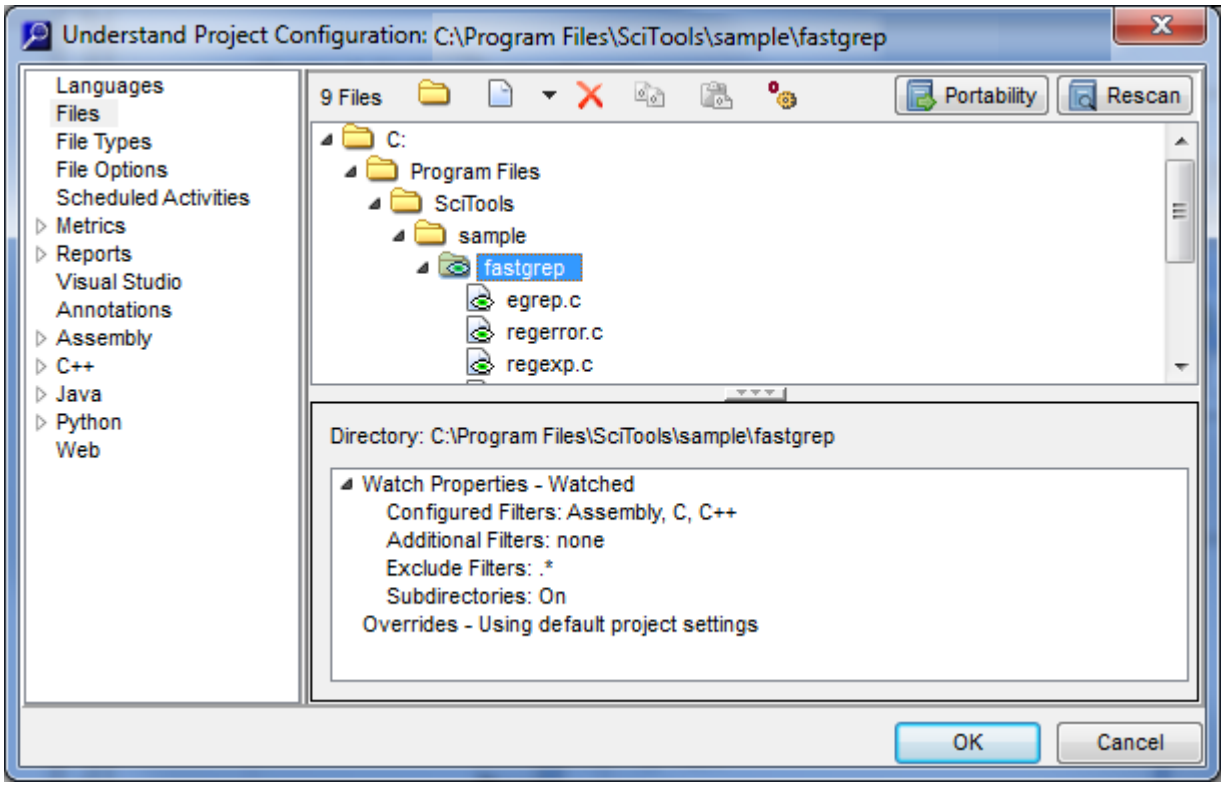
- Support for Objective-C and Objective-C++ (used for Mac OS and iOS) is provided with the strict analyzer, but not with the default C/C++ analyzer.
- Provides better support for Templates.
- Provides better support for Overloaded functions.

The default C/C++ analyzer aims to use fuzzy logic to handle incomplete, non-compiling code gracefully and as accurately as possible. The new analyzer is more strict than the old analyzer and requires a more accurate project definition—for example, by specifying all include paths and macro definitions and including only those files in the project that are used in the software build. For details, see the Creating Accurate C/C++ Projects web page.

If you are using the strict analyzer, see *C++ (Strict) Options* on page 75 for how to configure your project.

## Files Category

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




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

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

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

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

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

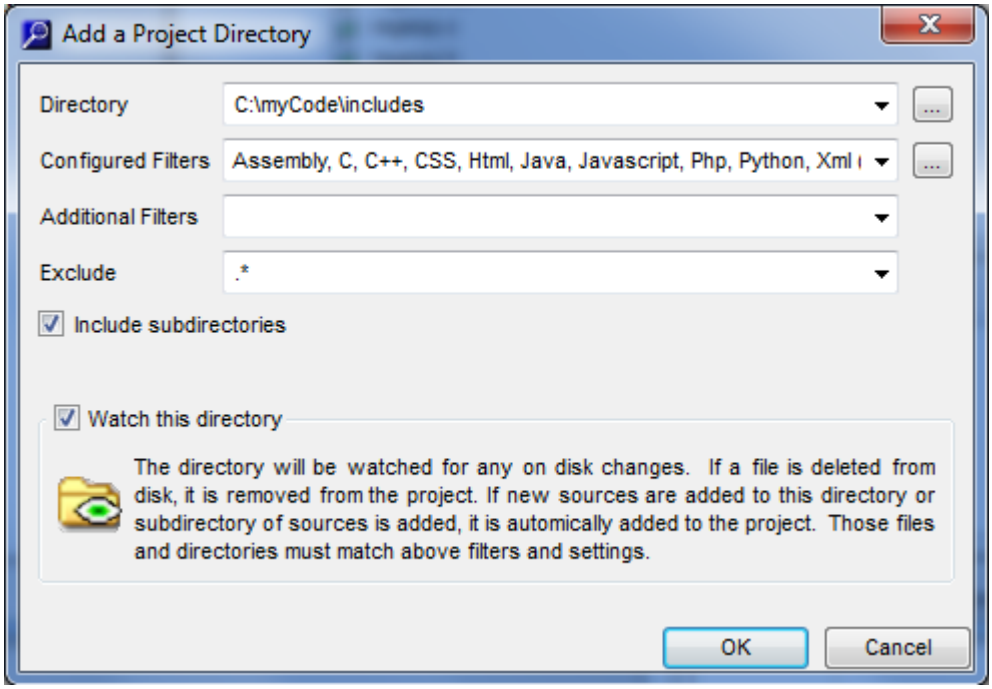
Note that your changes are not saved until you click **OK**.

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

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

## Adding Directories

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




1. In the **Directory** field, type the full directory path. Or, you can click the **...** button and use the Browse for Folder dialog to locate a directory containing source files and click **OK**.
2. In the **Configured Filters** field, click the **...** button if you want to add or delete languages from the list shown. In the Select Filters from Configured File Types dialog, put a checkmark next to any languages you want to be recognized as part of the project. Notice that additional languages are listed beyond those shown in the

Languages category. These include Basic, MSDos Batch, Perl, Tcl, Text, and Verilog.

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

3 In the **Additional Filters** field, type a pattern-matching string that matches only the files you want to keep in the analysis. For example, std*.* includes only files that begin with “std”. You can separate filters with a comma.

4 In the **Exclude** field, type a pattern-matching string that matches files you want to exclude from the analysis. For example, temp*.* excludes all files that begin with “temp”. You can separate filters with a comma.

5 To select and add multiple subdirectories to a project configuration, check the **Include subdirectories** box (on by default). This causes all source files matching the filter in all subdirectories of the specified path to be added to the project.

6 If you want this directory to be watched for any new files or deleted files, check the **Watch this directory** box. Whenever a source file is added to or deleted from this directory, the change is reflected in this project. Watched directories are indicated by the  icon in the files list. Directories excluded from being watched are indicated by the  icon. By default, the subdirectories of a watched directory are also watched. See page 45 for watch setting overrides.

7 After you have set the fields, click the **OK** button to add the source files in that directory to the project. You can click **Cancel** if the add file process is taking too long.

***Tip:*** You may add files from multiple directory trees.

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

All directory paths are absolute.

## Adding Files

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

If you click the  down arrow next to the  icon, you can choose to import a text file that contains a list of source files to import. For example, you might generate such a file from a compiler application or code management system. The file should contain one absolute file path per line. See *Adding Files to a Project* on page 322 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.

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

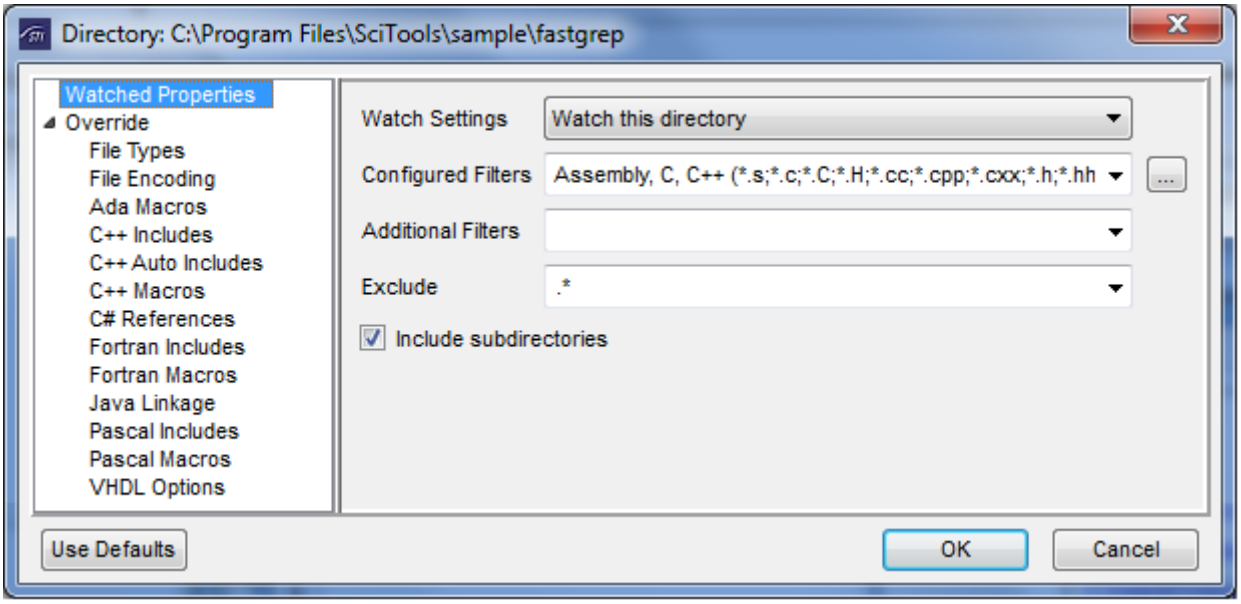
## Setting Overrides

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.

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

1 Select a directory.

2 Click  or right-click and select **Configure override settings**.




3 In the **Watched Properties** category, you can choose how files in this directory should be watched for new files to add to the project or deleted files to remove from the project. For **Watch Settings**, you can choose to watch a directory, not watch a directory, or inherit watch settings from the parent directory. In addition to specifying whether to watch a directory, you can set filters and exclude filters for an individual directory that control what types of new and deleted files will be found.

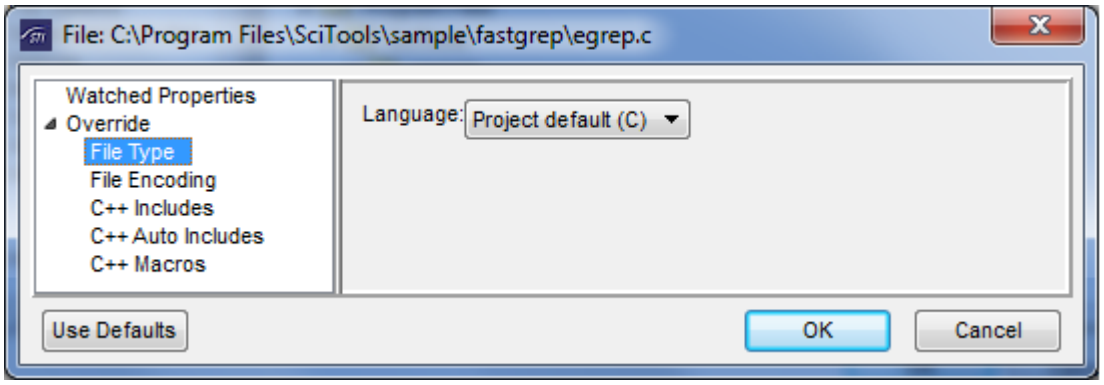
4 In the various **Override** categories, you can make directory-specific language-related settings. The list of categories depends upon the languages enabled in your project.

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

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

1 Select a file.

2 Click or right-click and select **Configure override settings**.




3 In the various **Override** categories, select a category and make changes. The categories available are different depending on the language of the source file. See page 61 through page 90 for details. The **Watched Properties** category is available for file overrides if you are using Relative or Named Root portability.

4 Click **OK** to save your overrides.

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

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

## Scanning Watched Directories

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

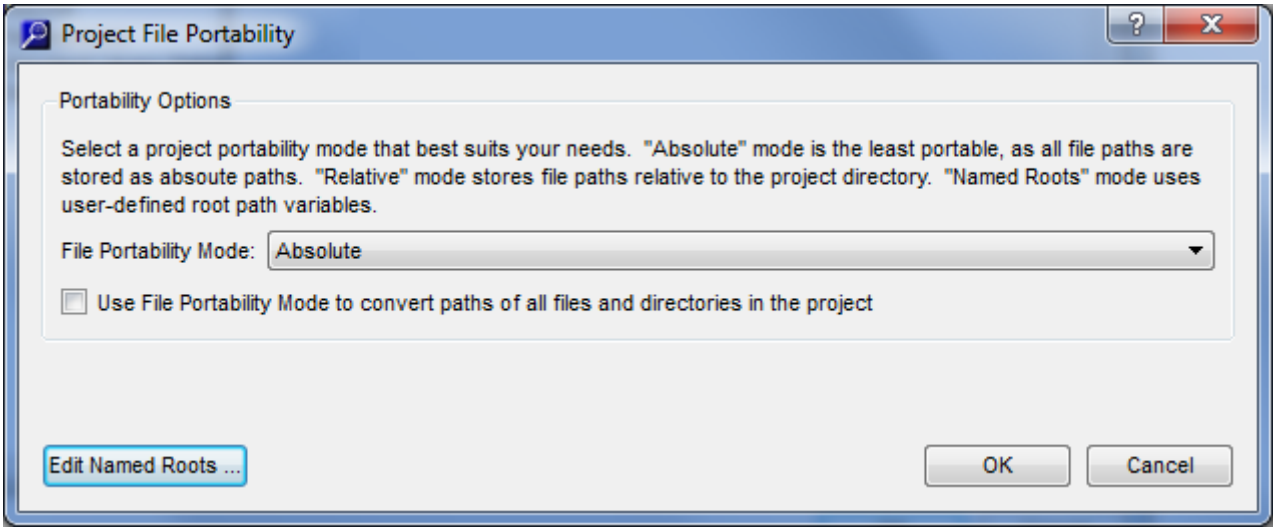
If files are found that you don't want to include in the project, uncheck the boxes next to those files to exclude them from the project configuration.

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

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

### Setting File Portability

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




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

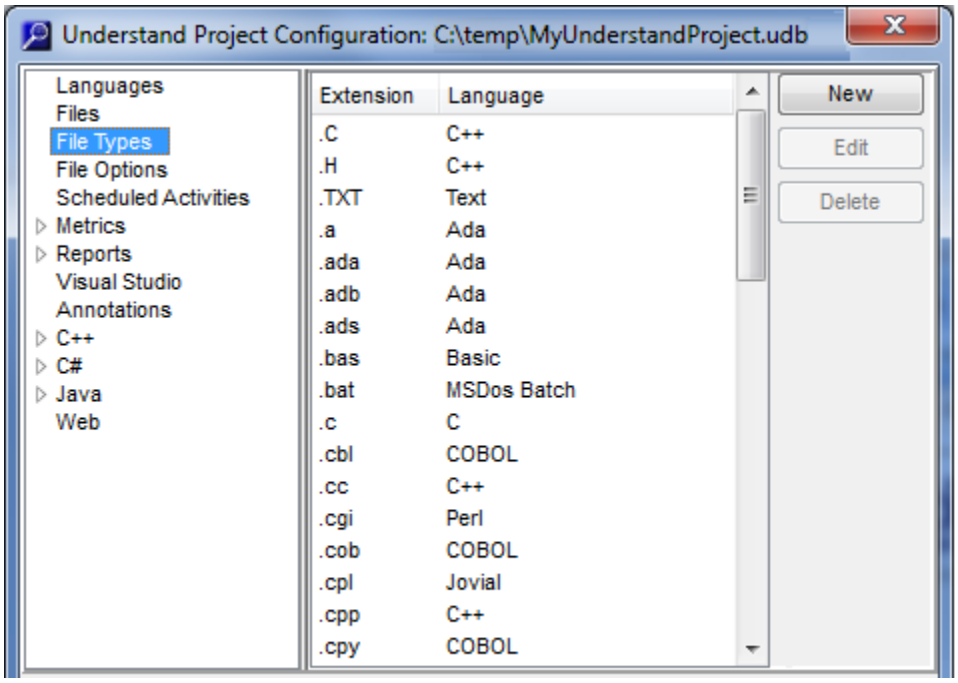
The choices are as follows:

- **Absolute:** This option is the default. It stores full file paths for all directories. If the source files change location, the paths will be incorrect.
- **Relative:** This option stores the relative path to directories from the location of the *Understand* project database. If you store the project database in the source file tree and move it along with the source files, the project can still be used.
- **Named Root:** This option allows you to specify “Named Roots” that are similar to environment variables to point to a root directory. Different users may then use different definitions for a named root. Click the **Edit Named Roots** button and see page 107 for details.

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

# File Types

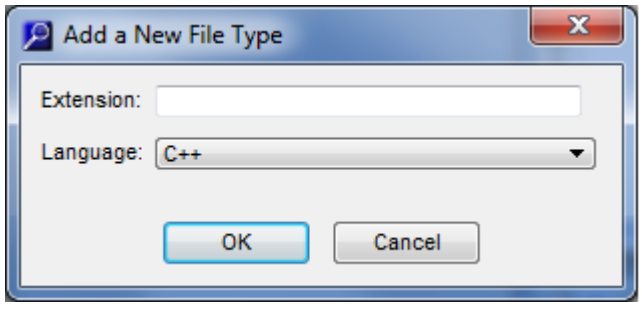
In the **File Types** category of the Project Configuration dialog, you can control how file extensions are interpreted by *Understand*.




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

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

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

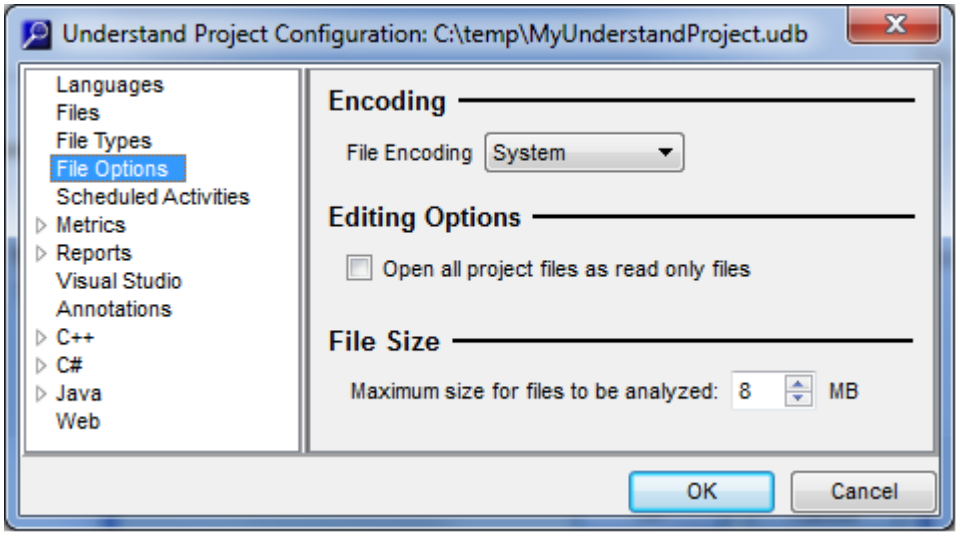



The file extension you type should begin with the period. It can contain simple * and ? wildcards.

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.

# File Options

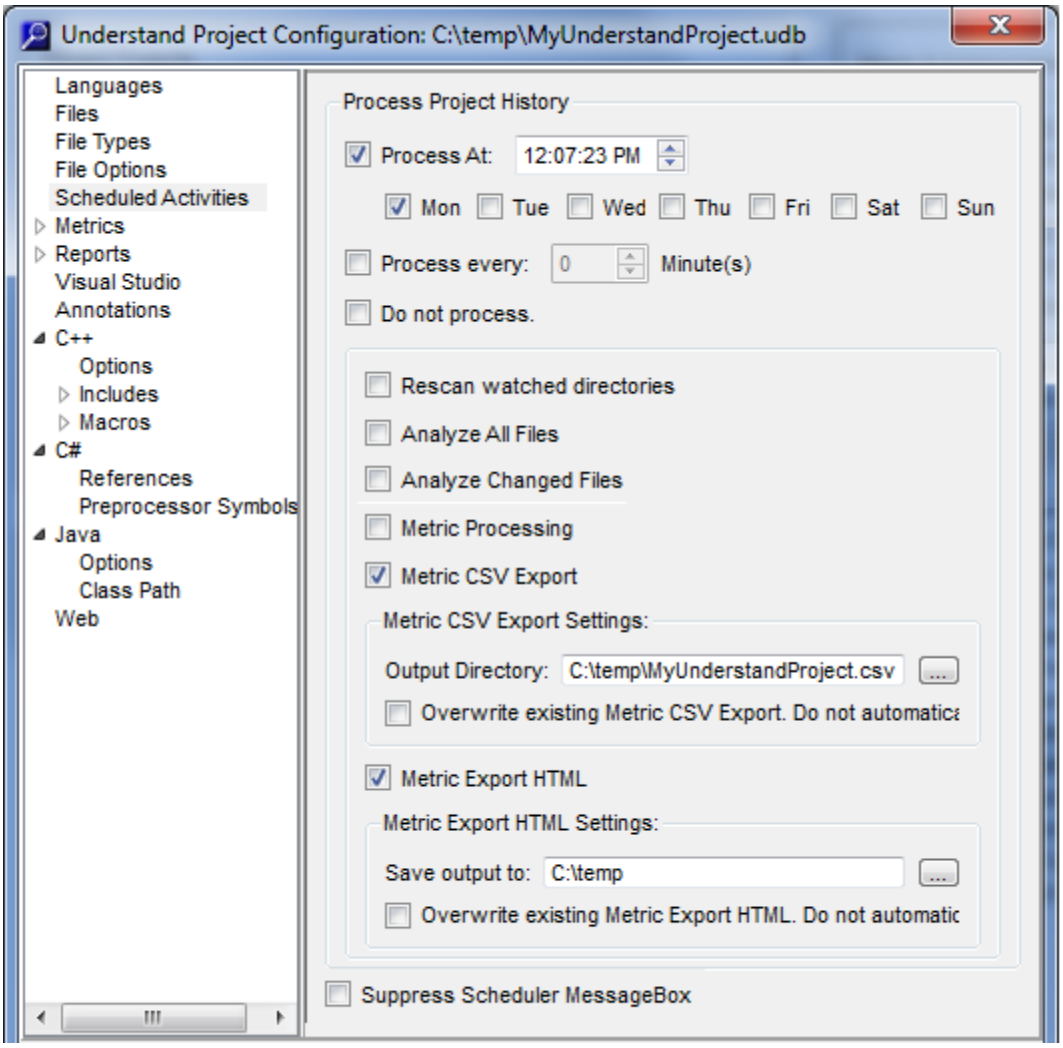
In the **File Options** category of the Project Configuration dialog, you can control how files are opened and saved by *Understand*.




- **File Encoding:** Select the type of encoding to use when saving source files. Many encoding formats are supported. You should change this only if your other applications have problems opening or displaying files saved by *Understand*. See *Editor Category* on page 109 for more information. The default file encoding is “System”, which means the default encoding for your computer. If you change the setting here, new projects you create use the last setting you saved. You can override the file encoding setting on a file-by-file or directory-by-directory basis (see *Setting Overrides* on page 45).
- **Open all project files as read only files:** Check this option if you do not want files to be edited and saved within *Understand*.
- **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.

# Scheduled Activities

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




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

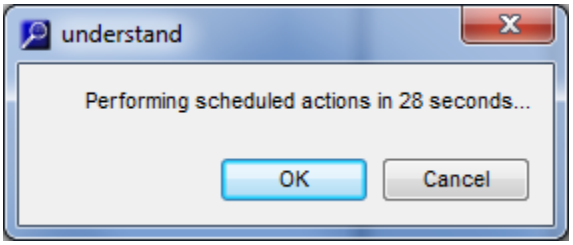
1. Check the **Process At** box.
2. Select either a processing time, a processing interval, or to not process events. For a processing time, check the boxes for one or more days of the week. For a processing interval, specify a number of minutes up to 1440 (24 hours).
3. Check the boxes for the events you want performed. The events occur in the sequence shown. For example, watched directories are scanned before the project is analyzed, and the project is analyzed before metrics are processed.

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

The following activities are available for scheduling:

- **Rescan watched directories:** Check this box to automatically check for files that have been added to or deleted from project directories. See *Adding Directories* on page 43 for how to specify which directories to watch. If you have watched directories, you should always run this task before the “Analyze all files” task. To run this action without scheduling it, choose **Project > Rescan Project Directories**.
- **Analyze all files:** Check this box to automatically analyze all project files as described in *Analyzing the Code* on page 121. Run this task before generating any metrics so that the statistics will reflect the current state of the project. To run this action without scheduling it, choose **Project > Analyze All Files**.
- **Analyze changed files:** Check this box to automatically analyze any project files that have changed as described in *Analyzing the Code* on page 121. Run this task before generating metrics so that statistics will reflect the current state of the project. To run this action without scheduling it, choose **Project > Analyze Changed Files**.
- **Metric processing:** Check this box to automatically calculate project metrics. The metrics selected in *Metrics > Selected Category* on page 53 are processed. Run this task if you plan to schedule either of the following metrics export tasks.
- **Metric CSV export:** Check this box to automatically export metrics as a comma-separated value file. If this box is checked, you can select the directory path and output filename for the export. By default, any existing file with the same name is renamed to provide a backup. You can check the **Overwrite** box if you simply want to replace the old export file. To further configure the export, see *Metrics* on page 52. To run this action without scheduling it, choose **Metrics > Export Metrics** and see *Exporting Metrics to a CSV File* on page 235.
- **Metric export HTML:** Check this box to automatically export metrics as web pages. If this box is checked, you can select the directory path for the export. By default, any existing file with the same name is renamed to provide a backup. You can check the **Overwrite** box if you simply want to replace the old export file. To run this action without scheduling it, choose **Metrics > Project Reports** and see *Exporting Metrics to HTML* on page 234.

When scheduled activities are about to run, you see a dialog that gives you a chance to cancel the action. You can avoid this message by checking the **Suppress Scheduler MessageBox** in the configuration.



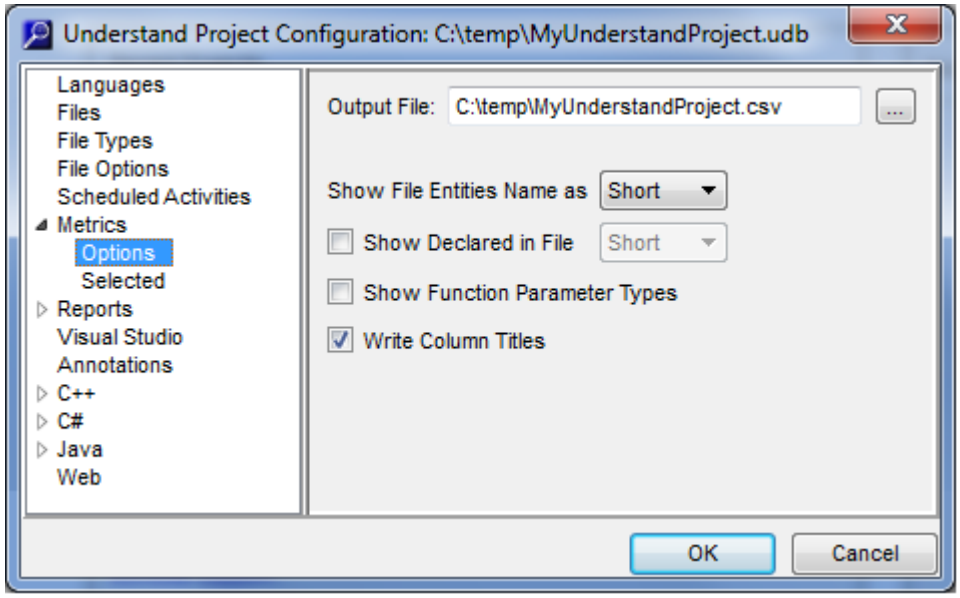

If you schedule activities, you see a message that asks if you are sure you want to prevent the scheduled activities from running when you exit *Understand*.

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

# Metrics

In the **Metrics** category of the Project Configuration dialog, you can control how metrics are generated when a CSV file is exported. These options set the defaults for both manual updates (page 235) and scheduled automatic updates (page 50).

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



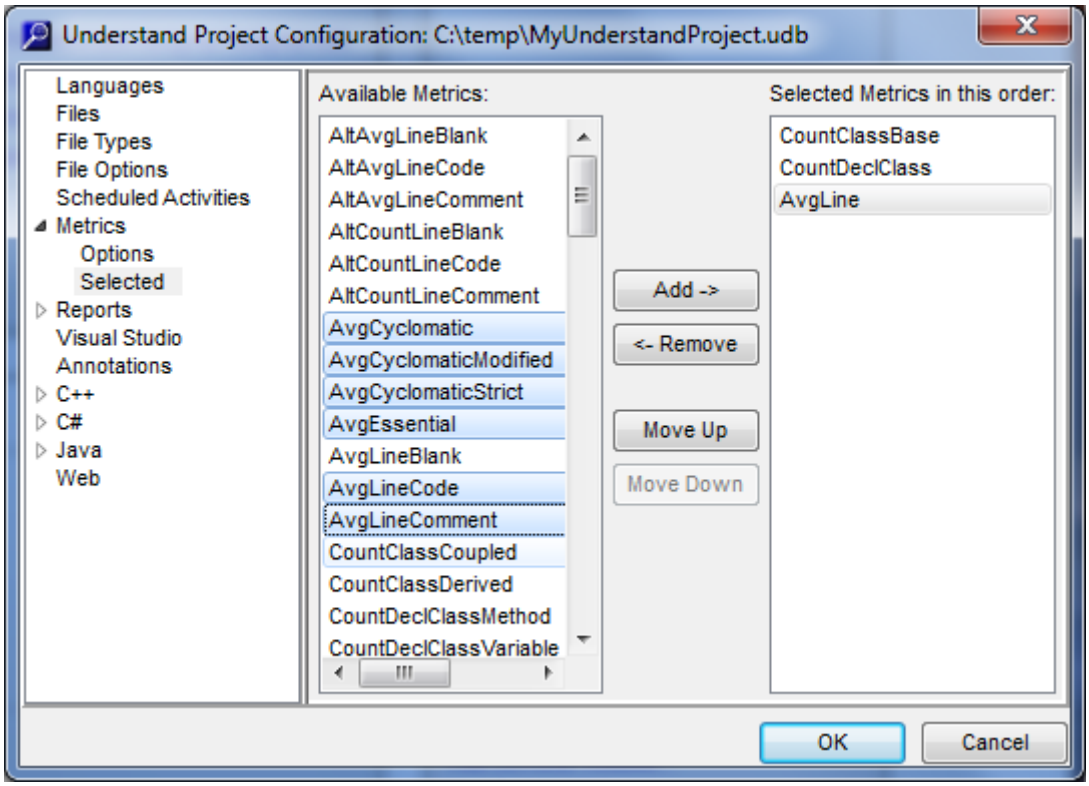

You see this window when you choose the **Project > Configure Project** menu item and then the **Metrics** category. If you attempt to generate metrics before configuring metrics, this window opens automatically.

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

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

### Metrics > Selected Category

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

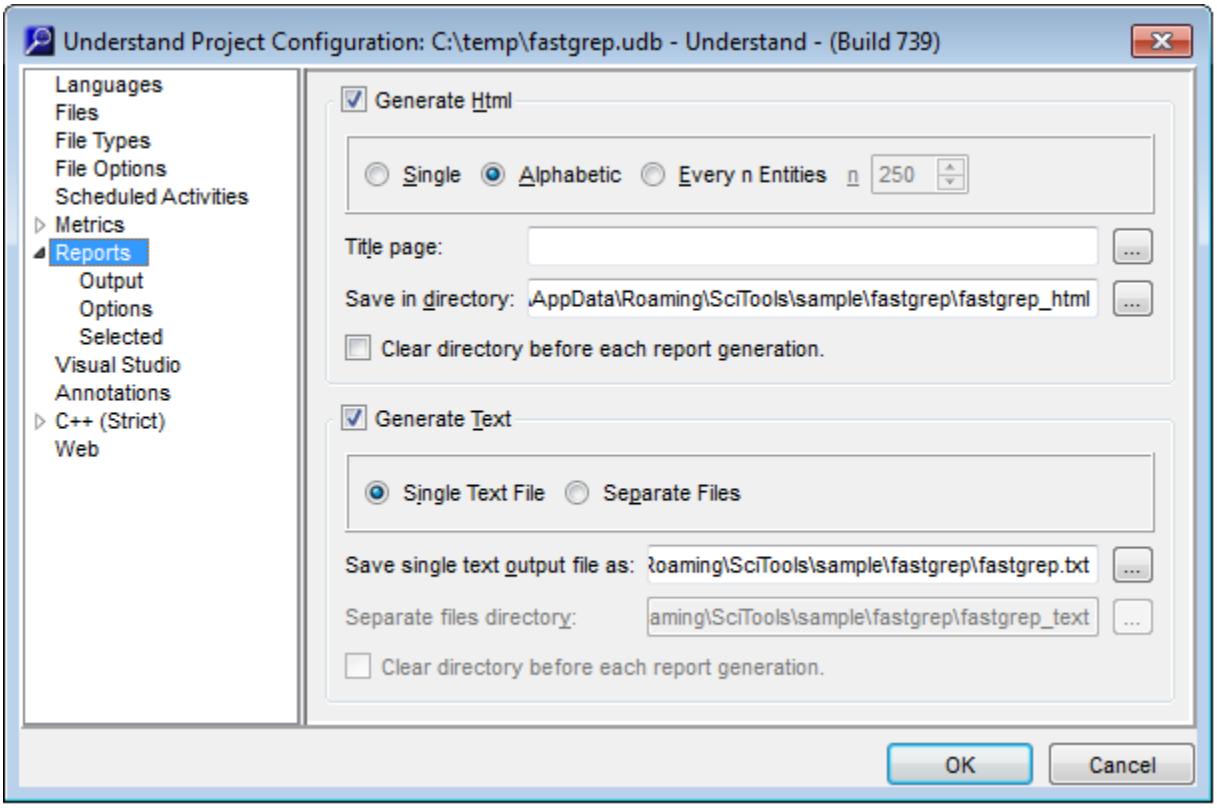



1. In the **Available Metrics** list (left), select metrics you want to include in the output you generate. You can hold down Shift to select a continuous group or Ctrl to select discontinuous items.
2. Click **Add** to copy the selected metrics to the right column.
3. You can reorder the metrics in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

The metrics available depend on the languages used in your project. See scitools.com/support/metrics_list/ for descriptions.

# Reports

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




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

You can control the colors and font styles in HTML reports as described in *Customizing Report Colors* on page 210.

### Reports > Output Category

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

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

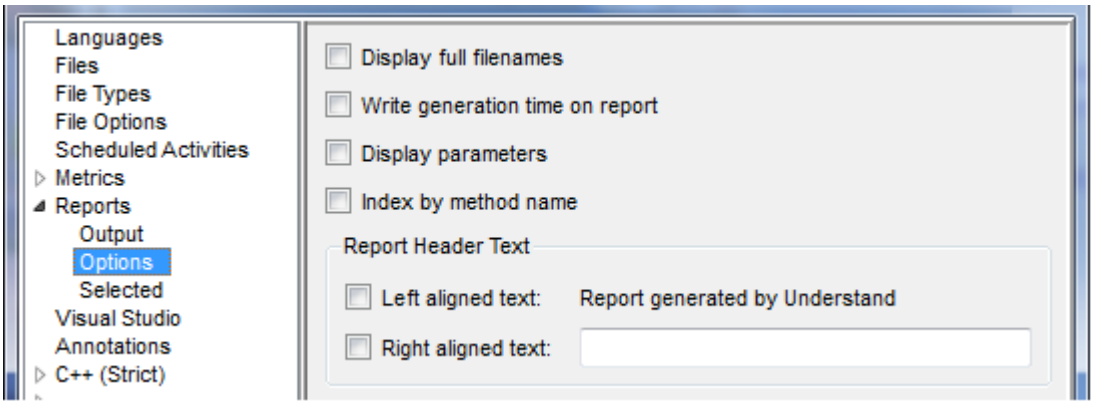
- The “home” page for reports is index.html. You can select an alternate **Title Page**.
- The default **Save in directory** is the <*proj_file*>_html folder below the folder where your *.udb file is stored, but you can select an alternate location.
- You can choose to clear the contents of the previously generated reports and anything else in the selected directory at the beginning of the report generation.

- **Generate Text:** This option causes the report generation to create simple text files containing the report data.
  - You may generate one text file of the specified location and name (by choosing **Single Text File**). Alternately, you may generate multiple text files (by choosing **Separate Files**) and specify a directory to contain all the files. The file extensions of each text file will denote the separate reports. Depending on which option you select, you can also select either a file or directory location for the output.

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

### Reports > Options Category

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

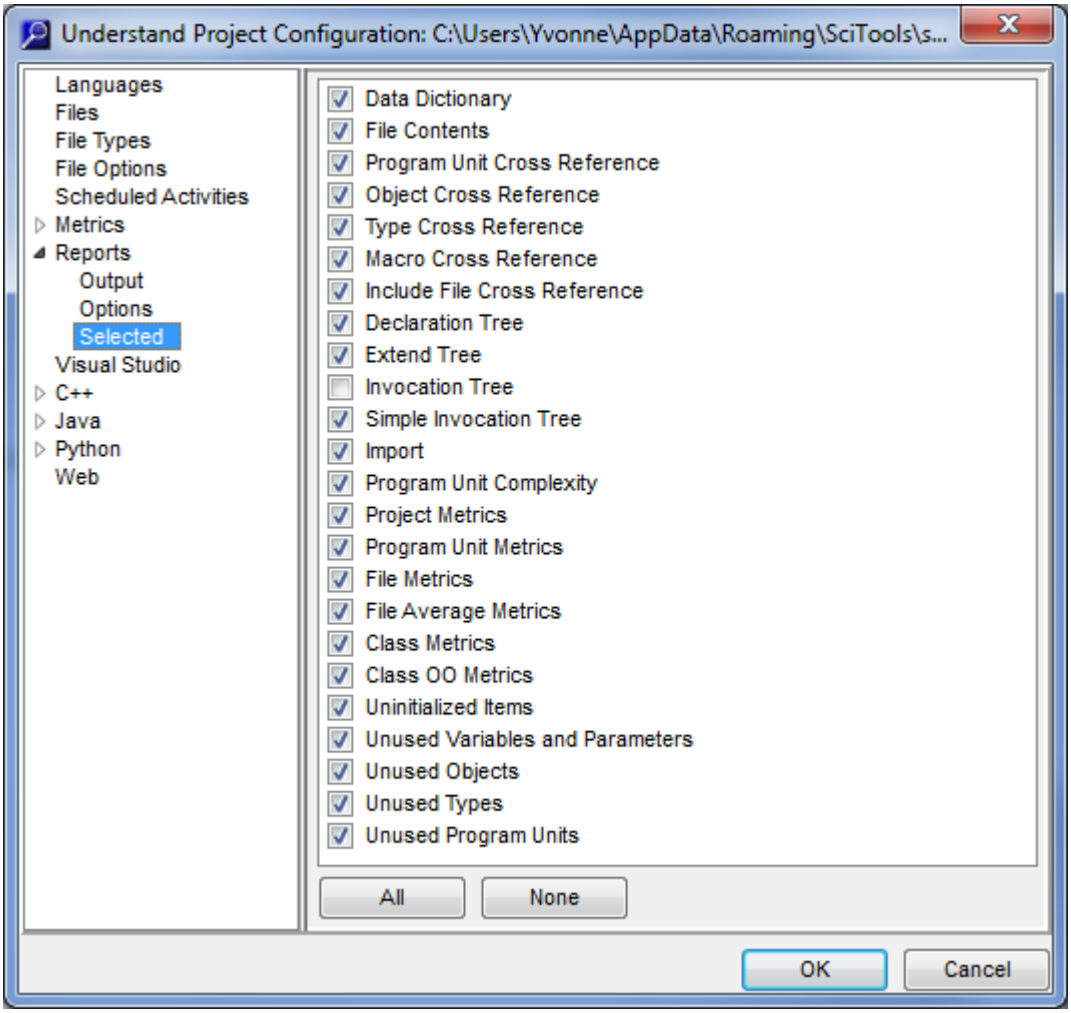



The **Options** category has the following fields:

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

### Reports > Selected Category

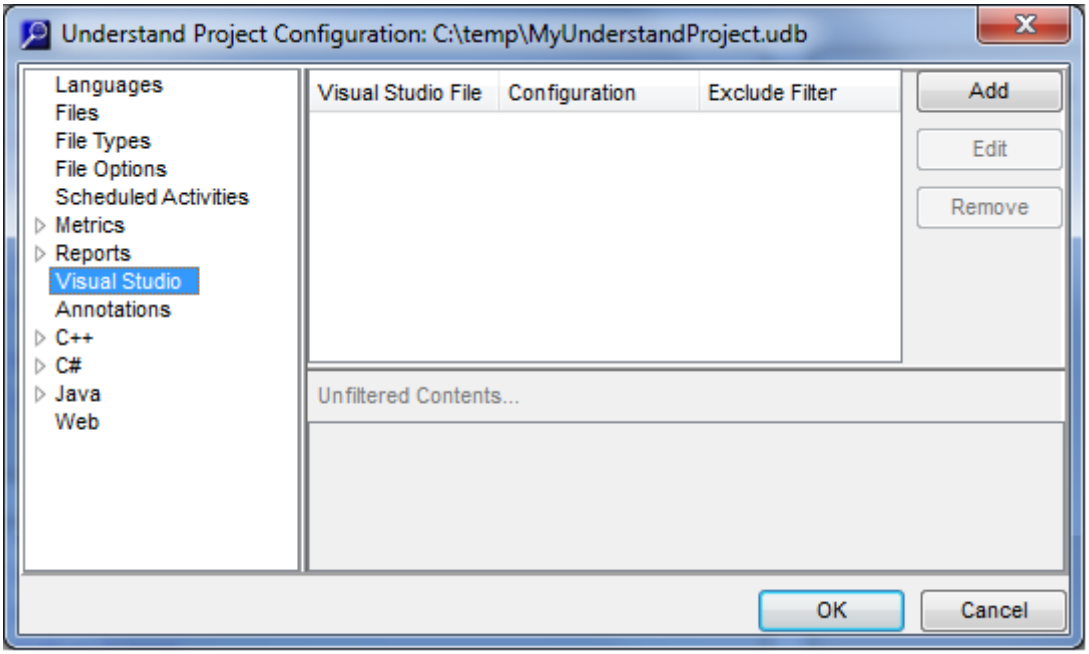
The **Selected** subcategory lets you check the boxes for the reports you want to generate. The list of reports differs depending on which languages are used in your project. See Chapter 8 for descriptions of these report formats.



# Visual Studio

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

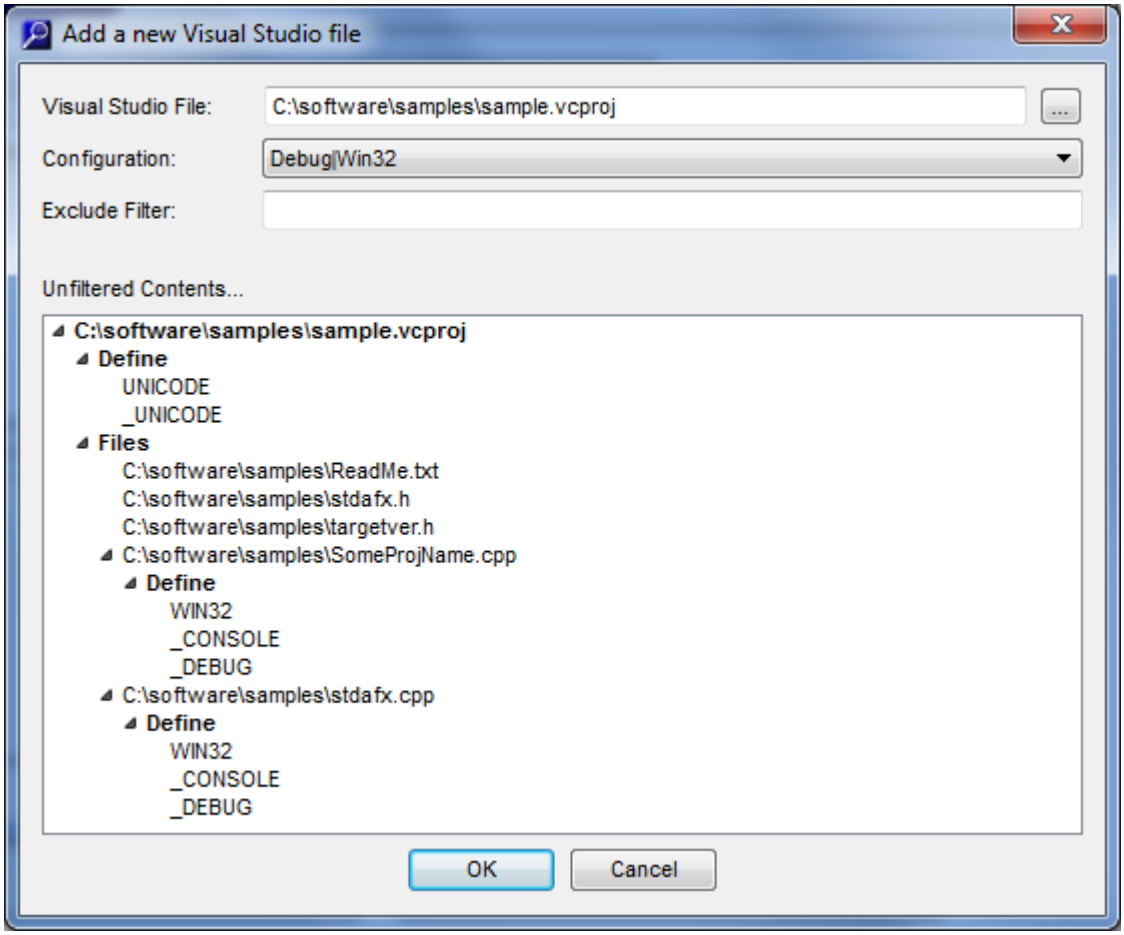
You see this window when you choose the **Project > Configure Project** menu item and select the **Visual Studio** category.




Follow these steps:

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

5 You can expand the **Unfiltered Contents** list to see the includes, defines, and files for the configuration currently selected.




6 Click **OK** to add this to your project.

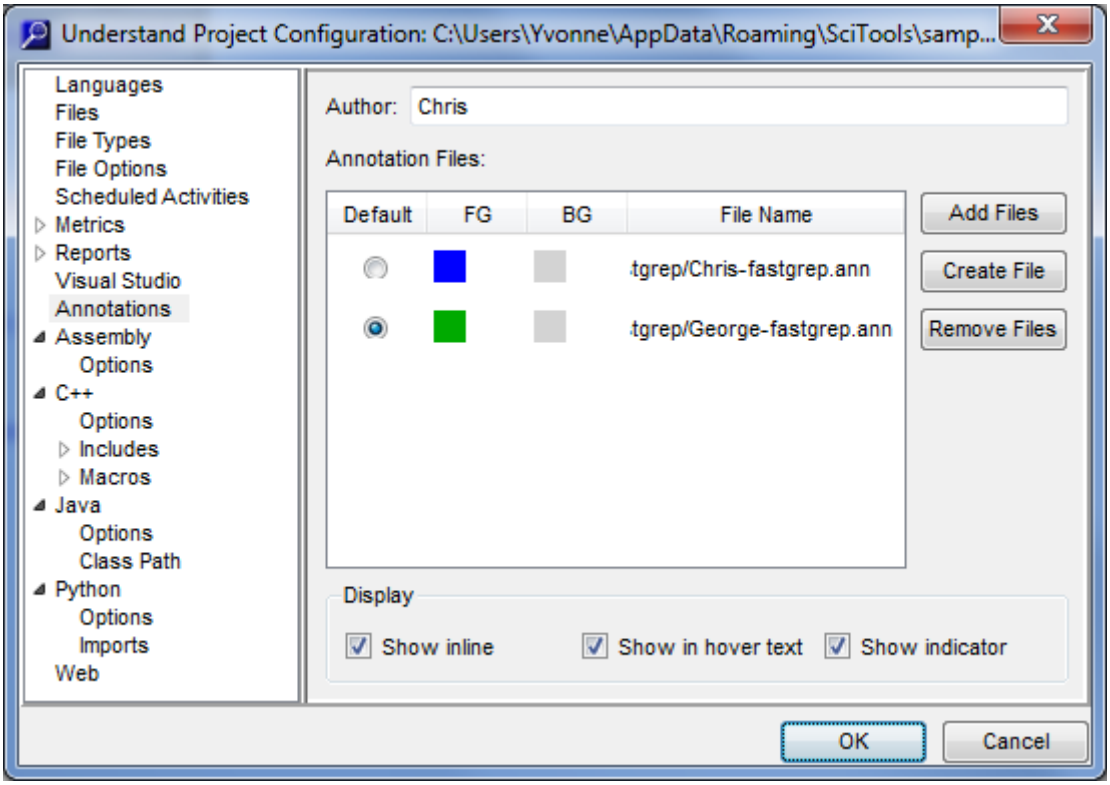
***Note:*** If you sync with a Visual Studio workspace file, the default target is used because there is no mechanism for specifying targets for each .dsp project within a .dsw file.

Once set, the source files, macros and include paths from the Visual Studio project are used by *Understand*. This is in addition to any project settings you configure in the other categories.

***Note:*** Settings in other categories for include path and macros take priority over the Visual Studio project settings. This permits you to use the bulk of the Visual Studio settings while selectively overriding as your needs require.

# Annotations

The **Annotations** category of the Project Configuration dialog lets you control how annotations are stored and displayed. See page 184 for details on using annotations.




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

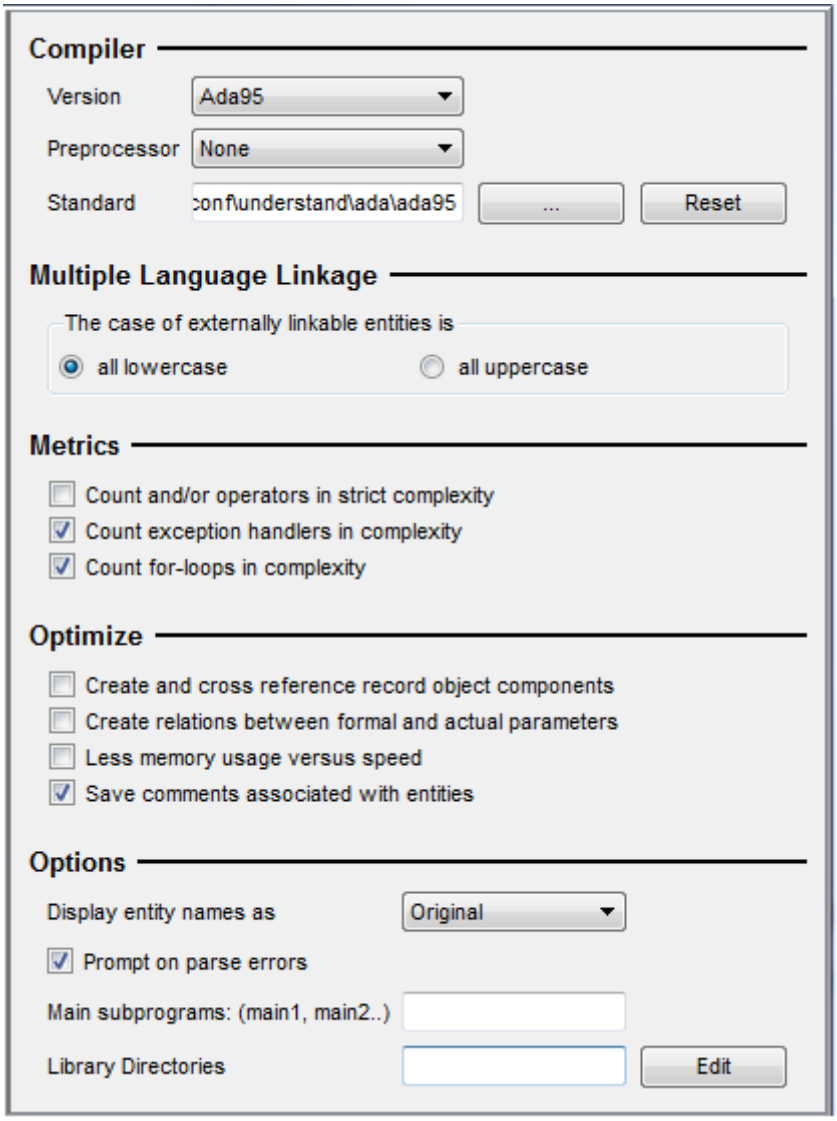
- **Show inline:** When this box is checked, annotations are shown following the place where the associated entity is defined in the code. You can also turn this display feature on and off by choosing **Annotations > Display Inline** from the menus.
- **Show in hover text:** When this box is checked,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

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

Multiple projects can reference the same *.ann files. Sharing annotation files across projects allows the same annotation to appear in multiple projects that share one or more source code files.

# Ada Options

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




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, 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 library used by this project. Default standards are provided in <install_directory>/conf/understand/ada.

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

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

### Ada > Macros Category

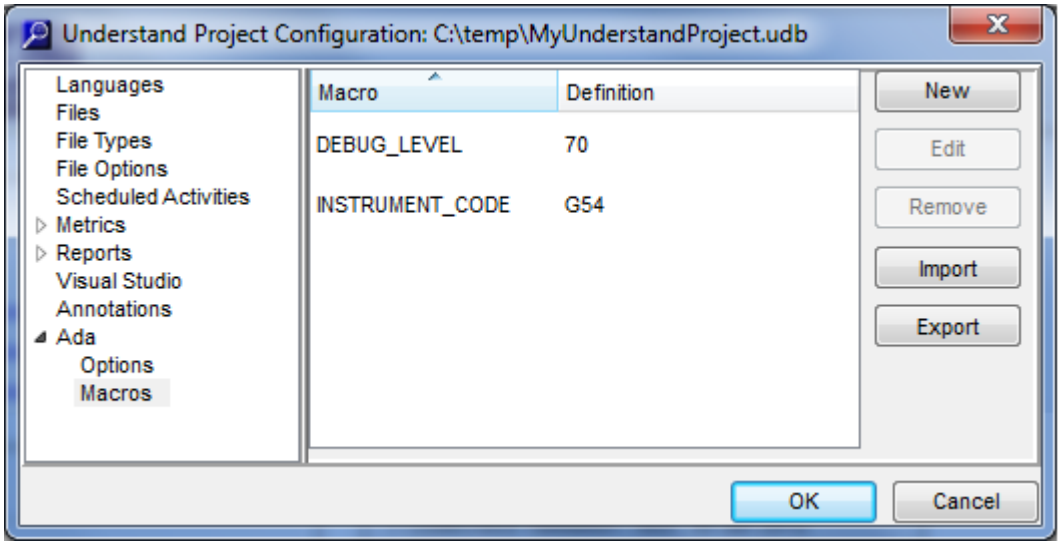
Ada code may contain conditional compiler instructions in pragma statements. For example:

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

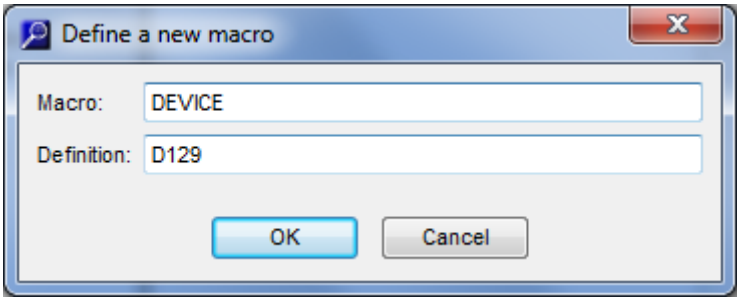
The supported pragmas are IF, IFDEF, ELSIF, ELSE, and ENDIF. These pragmas are similar to preprocessor directives such as #ifdef in C code.

For *Understand* to successfully analyze your software it needs to know what macro definitions should be set. For more about ways to configure macro definitions, see *Using the Undefined Macros Tool* on page 125 and the SciTools website.

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



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




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

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

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

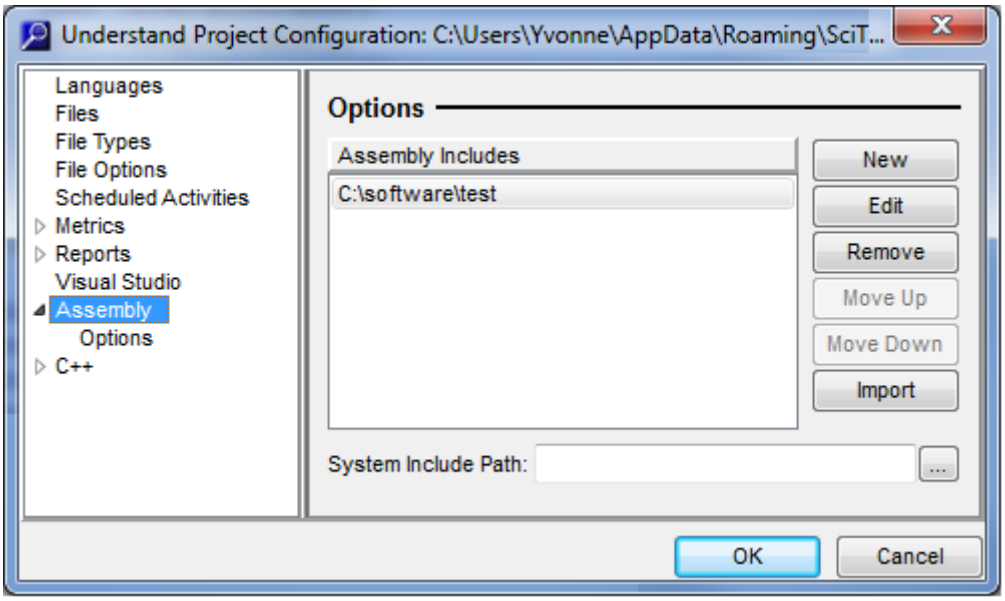
You can import or export a list of macros and their optional definitions by clicking **Import** or **Export** and selecting the file. The file must contain one macro definition per line. A # sign in the first column of a line in the file indicates a comment. Separate the macro name and its definition 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

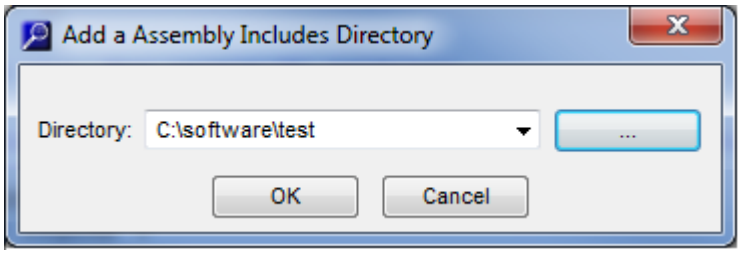
In the **Assembly > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze assembly source code. You see this window when you choose the **Project > Configure Project** menu item and select the **Assembly** category. Currently the only setting is the assembler, and Coldfire 68K is the only assembler supported.

The **Assembly > Includes** category in the Project Configuration dialog (which you open with **Project > Configure Project**) 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 are defined here. For project-level includes that you want to be analyzed, add those include files as source files in the **Files** category.

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




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

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

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

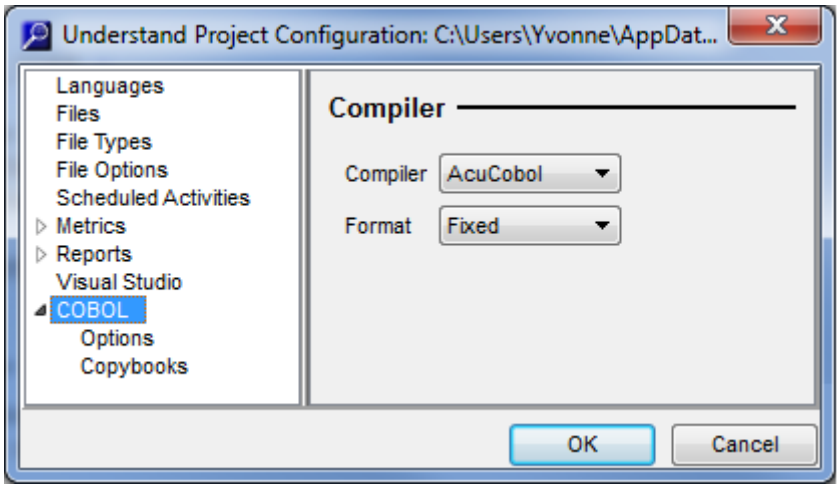
Include paths are not recursively searched; that is, any subdirectories will not be searched for include files unless that subdirectory is explicitly specified in the list of include directories.

You may use environment variables in include file paths. Use the $var format on Unix and the %var% format on Windows. You can also use named root in include file paths (see page 107).

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

## COBOL Options

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



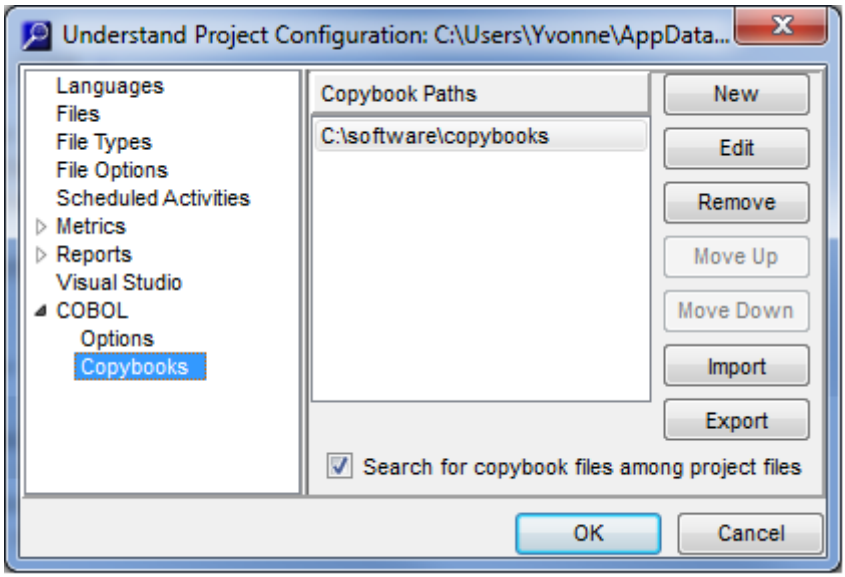

The field in the **COBOL > Options** category is as follows:

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

## COBOL > Copybooks Category

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

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




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

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

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

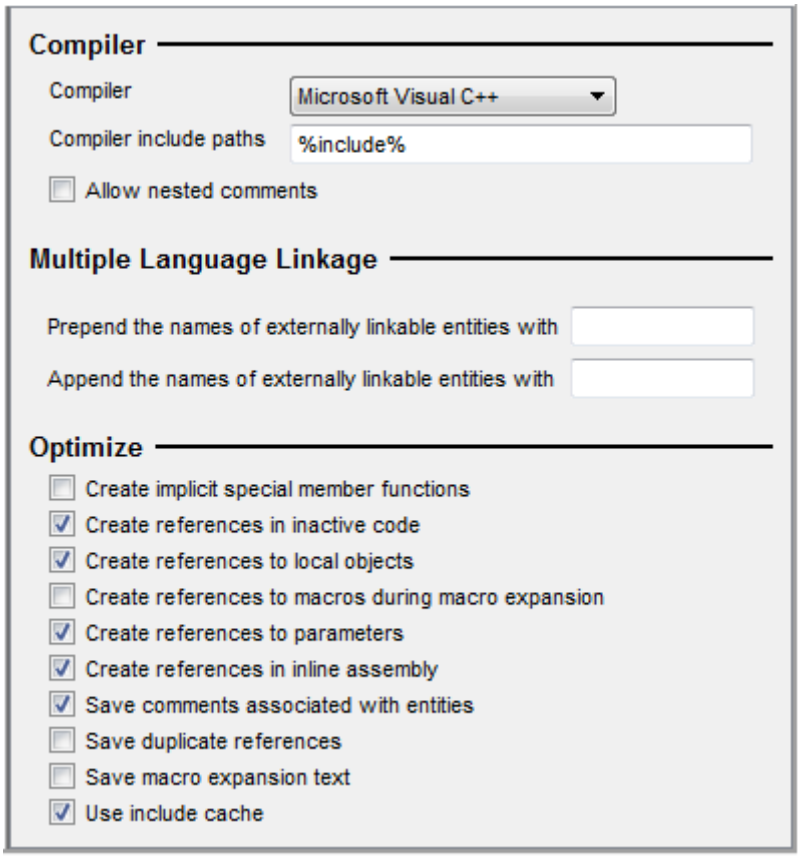
Copybook paths are not recursively searched; that is, any subdirectories will not be searched for copybook files unless that subdirectory is explicitly specified in the list of copybook directories.

You may use environment variables in copybook file paths. Use the $var format on Unix and the %var% format on Windows. You can also use named root in copybook file paths (see page 107).

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

# C++ (Fuzzy) Options

In the **C++ > Options** category of the Project Configuration dialog, you can tell *Understand* how to analyze C and C++ source code. If you selected the **Fuzzy** option for C/C++ in the Languages category, you see the following window when you choose the **Project > Configure Project** menu item and select the **C++** category.




(If you selected the **Strict** option in the Languages category, see *C++ (Strict) Options* on page 75 for how to configure your project.)

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

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

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

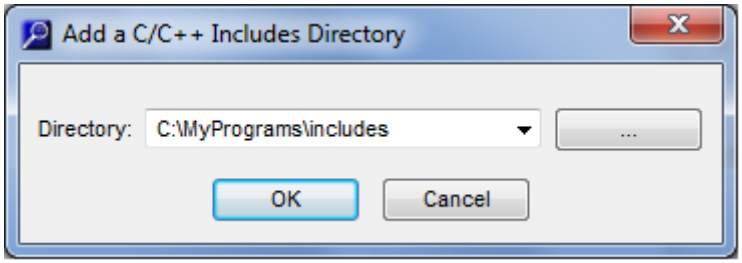
## C++ > Includes Category

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

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 123.

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

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




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

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

You may use environment variables in include file paths. Use the $var format on Unix and the %var% format on Windows. You can also use named roots in include file paths (see page 107).

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

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

you analyze the project. If you want to be prompted for how to handle missing files during the analysis, you must choose **Tools > Options** and enable the **Allow prompting for missing include files on a per project basis** checkbox in the **Analyze** category (page 105). Then, you will see this field in the Project Configuration dialog. If you then check the **Prompt for missing include files** box, you may choose to add a directory to the include path, ignore the missing file, or stop warning about missing files during the analysis.

- **Search for include files among project files:** This option directs the analyzer to look among project files as a last resort for missing include files. The default is on.
- **Treat system includes as user includes:** This option tells the 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:** Check this option if you want to ignore any directory specifications in #include statements and instead use the include file wherever it is found in the project. The default is off.
- **Use case-insensitive lookup for includes:** This option tells the analyzer whether to ignore the case of filenames in #include statements. The default is off. (Not available on Windows; Windows lookups are always case-insensitive.)

### C++ > Includes > Auto Category

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

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

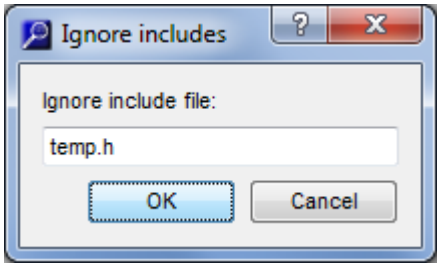
You can import or export a list of auto include files from a text file by clicking **Import** or **Export** and selecting the text file that contains one file path per line.

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

### C++ > Includes > Ignore Category

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

To add a file to be ignored, click **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 will be added to this list.

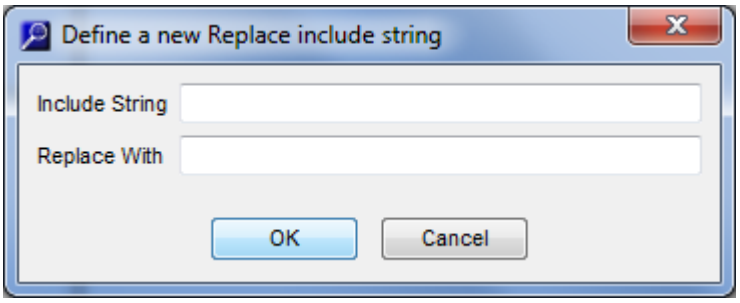
You can import or export a list of files to ignore from a text file by clicking **Import** or **Export** and selecting the text file that contains one filename per line.

### C++ > Includes > Replacement Text

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

### C++ > Macros Category

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

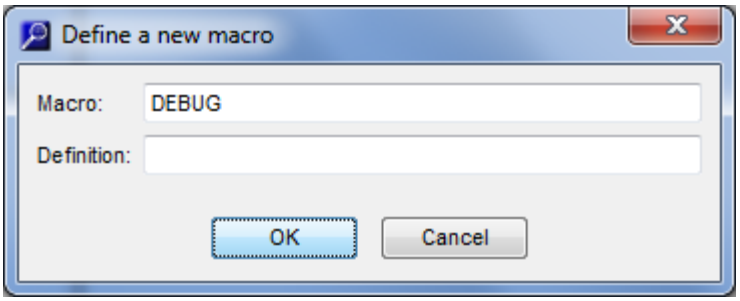
```
#define INSTRUMENT_CODE
#ifdef INSTRUMENT_CODE
... statements ...
#endif
```

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

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 125 and the SciTools website.

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

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




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

***Note:*** A number of preprocessor macros are automatically supported. In additions to the common macros, *Understand* supports the following macro formats for embedded assembly code if you are using the "fuzzy" 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)

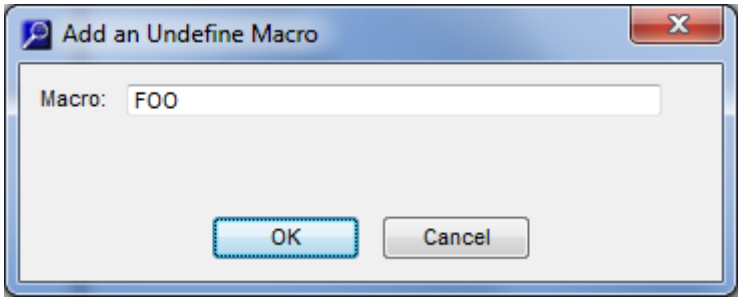
**5** Project defines (Macros category)

**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**.



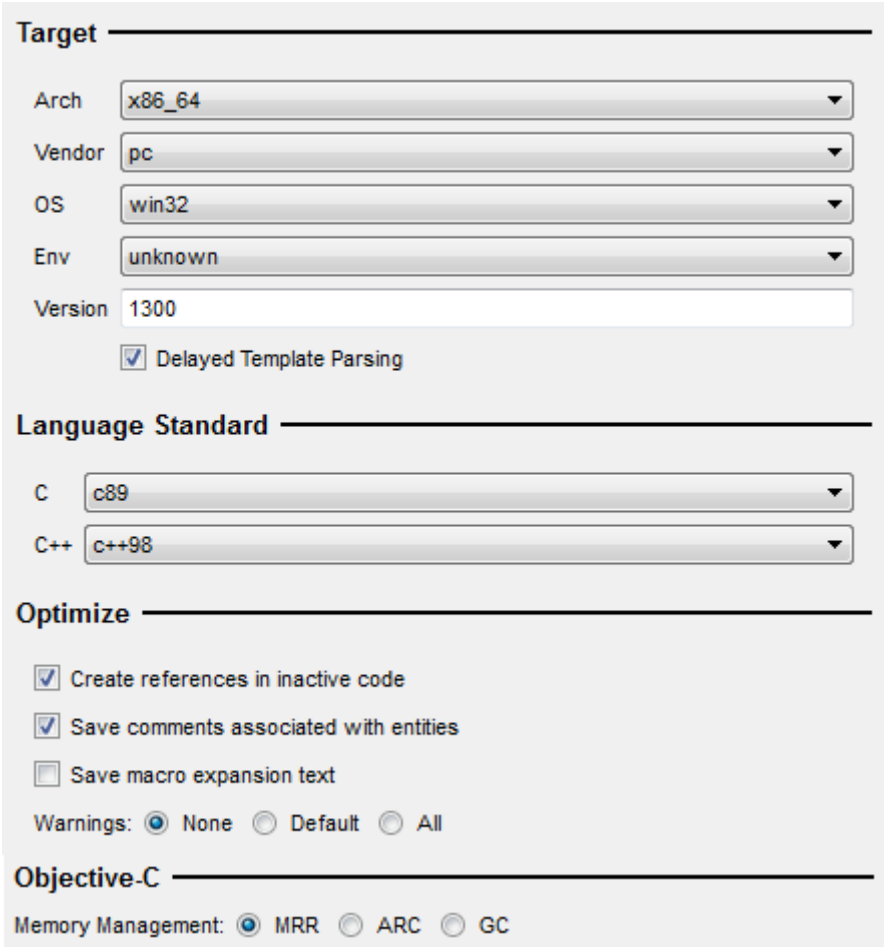

You can import or export a list of undefined macros from a text file by clicking **Import** or **Export** and selecting the file. The file must contain one macro name per line. A # sign in the first column of a line in the file indicates a comment.

# C++ (Strict) Options

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

***Note:*** If you did not select the **Strict** option in the Languages category next to the C/C++ box, see *C++ (Fuzzy) Options* on page 68 for how to configure your project.

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




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

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

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

- **Arch:** Select the architecture of the chip for which your project is written. Examples of the many supported options include ARM, PowerPC64, and x86_64.
- **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:** Specify the appropriate version number. If your OS is IOS or MacOSX, specify the operating system version number. If your OS is Win32, 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 Win32, you can choose whether to delay parsing of template files. This option is required for compatibility with MSVC. However, be aware that unreferenced template code will not be analyzed at all if you enable delayed template parsing.
- **C Language Standard:** Select the C standard to which you want your C code to conform.
- **C++ Language Standard:** Select the C++ standard to which you want your C++ code to conform.
- **Create references in inactive code:** If you wish to exclude cross-reference information for code that is IFDEFed out by the current macro settings, turn this option off. By default, this option is on and cross-reference information for inactive code is included.
- **Save comments associated with entities:** Choose whether source code comments that occur before and after an entity should be associated with that entity.
- **Save macro expansion text:** If you put a check in this box, you can right-click on a macro and choose **Expanded Macro Text** from the context menu to see how the macro expands.
- **Warnings:** Choose how many of the warnings provided by the strict analyzer you want reported. These warnings indicate potential problems in the source code. Choosing to see some or all warnings is likely to slow down the project analysis somewhat.
- **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).

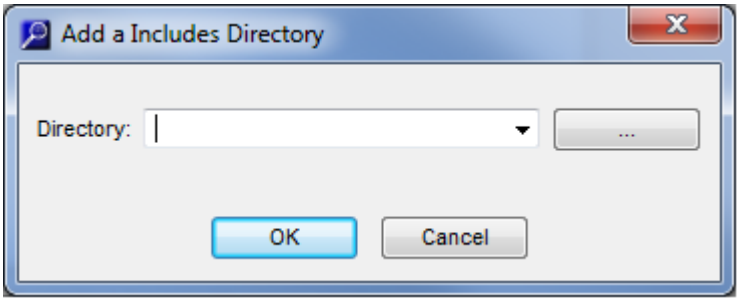
## C++ (Strict) > Includes Category

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

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 123 and the SciTools Support website.

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

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




During analysis, 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 page 107).

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

- **Sysroot:** 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:** Enabling this option causes include files found during project analysis to be added to the project automatically. This allows you to see more detailed information about such include files. The default is off.
- **Add found system include files to source list:** If you choose to add include files that are found to the source list, you can also choose whether system include files should be added. The default is off.
- **Prompt for missing include files:** If any include files cannot be found during analysis, you will normally see the **Include Paths** button in the Analysis Log after you analyze the project. If you want to be prompted for how to handle missing files during the analysis, you must choose **Tools > Options** and enable the **Allow prompting for missing include files on a per project basis** checkbox in the **Analyze** category (page 105). Then, you will see this field in the Project Configuration dialog. If you then check the **Prompt for missing include files** box, you may choose to add a directory to the include path, ignore the missing file, or stop warning about missing files during the analysis.
- **Search for include files among project files:** This option directs the analyzer to look among project files as a last resort for missing include files. The default is on.
- **Ignore directories in include names:** Check this option if you want to ignore any directory specifications in #include statements and instead use the include file wherever it is found in the project. The default is off.
- **Compare files by content instead of path:** Check this option if you want include files to be compared by their contents rather than by their file path. The default is off.

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

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

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

To search a directory and its subdirectories for a framework, click **Search**.

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

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

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

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

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

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

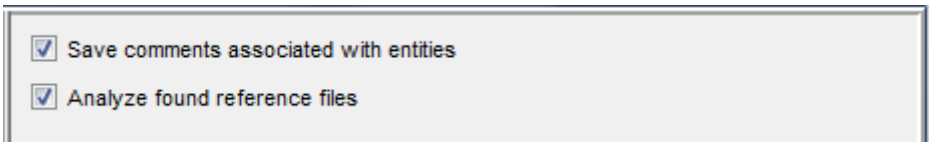
**C++ (Strict) > Macros Category**

For information about the **C++ (Strict) > Macros** category, see *C++ > Macros Category* on page 72.

For information about the **C++ (Strict) > Macros > Undefines** category, see *C++ > Macros > Undefines Category* on page 74.

# C# Options

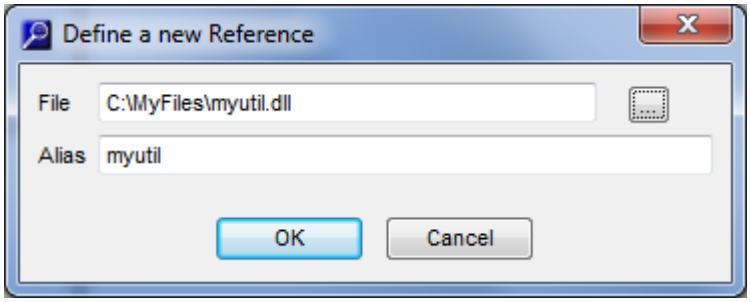
In the **C# > Options** category of the Project Configuration dialog, you can control how C# source code is analyzed. You see this window when you choose the **Project > Configure Project** menu item and select the **C#** category.




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

- **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, methods in reference libraries are not counted for the purpose of computing metrics. The default is on.

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



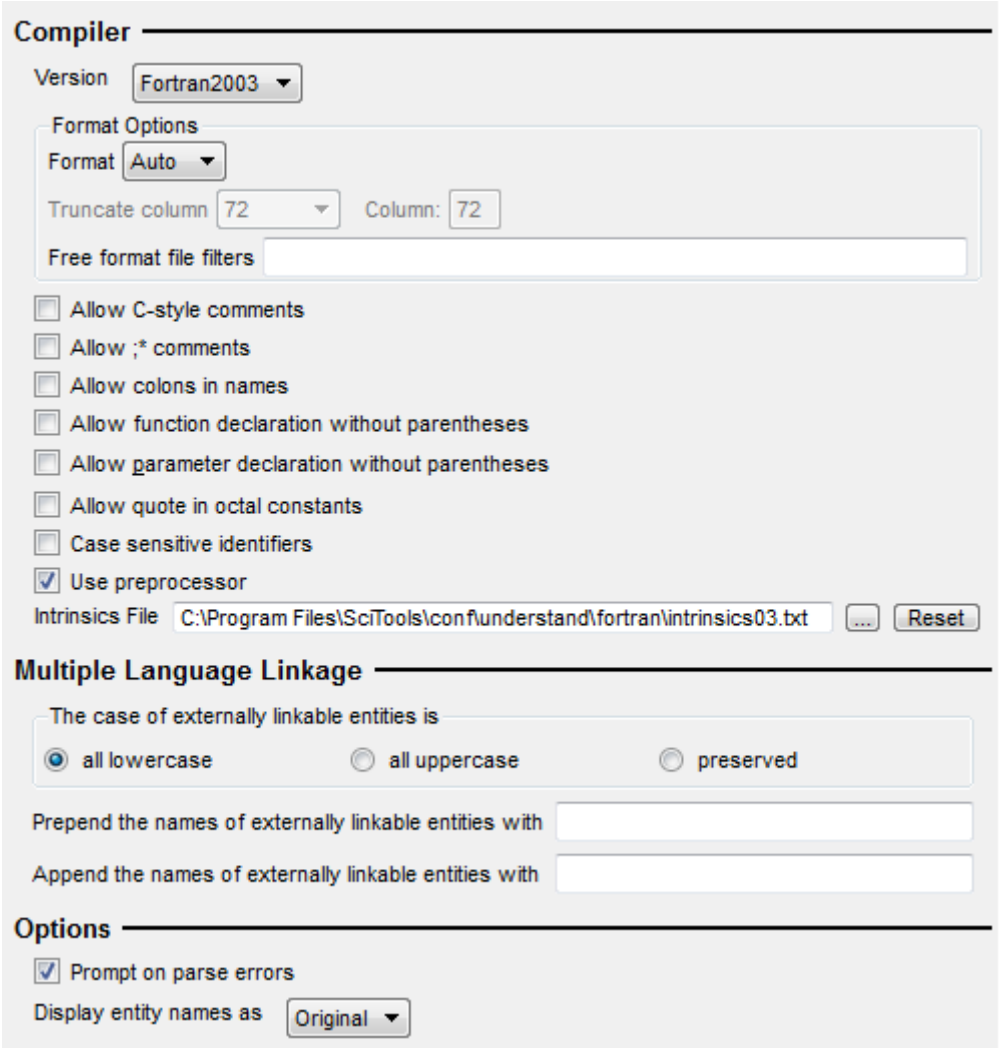

You can import or export a list of reference files and their aliases from a text file by clicking **Import** or **Export** and selecting a file that contains one reference and its alias per line. The file should separate the reference file and its alias with an equal sign (=).

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

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

# Fortran Options

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




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

- **Version:** Select the variant of Fortran used by the source code in this project. If you change the version after creating a project, the project will be reanalyzed when you click OK. The choices are Fortran77, Fortran90, Fortran95, and Fortran2003. If you have a mix of code, choose the newest language variant. That is, if you have F77 and F95 code, choose F95. The default is Fortran95.
- **Format:** Some older Fortran variants and all new variants permit *free form* statements, which may cross lines). Fixed form statements are terminated by a line end or column number. The default is “auto format,” which automatically detects the

parsing format (fixed or free) on a file-by-file basis. This allows you to mix free and fixed format. Auto format also determines the correct truncation point for fixed format files. Choose “fixed” or “free” only if all your source files have the same format. Blocks of freeform code can be used within a fixed format file if you bracket the blocks with !dec$freeform and !dec$nofreeform.

- **Truncate column:** If you choose fixed form, you may choose what column terminates statements. Common columns 72 and 132 are available or you may specify a column or no truncation.
- **Allow C-style comments:** Check this option if your Fortran code contains comments of the form /* ... */.
- **Allow ;* 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 that contains intrinsic functions you want to be analyzed. Default intrinsics files are provided in the <install_directory>/conf/understand/fortran directory: intrinsics77.txt, intrinsics90.txt, and intrinsics95.txt.
- **Case of externally linkable entities:** Choose which case should be used for “exporting” entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as “MYITEM” and you choose “all lowercase” here, other languages would be expected to call that entity as “myitem”.
- **Prepend the names of externally linkable entities with:** You may optionally type a string that you want used as a prefix to reference all linkable entities in other source code languages.
- **Append the names of externally linkable entities with:** You may optionally type a string that you want used as a suffix to reference all linkable entities in other source code languages.

- **Prompt on parse errors:** By default, a prompt asks how to handle any parsing errors. When prompted during analysis, you may choose to ignore that error or all future errors. Turn this option off to disable this prompting feature. If you turned it off during analysis, but later want to turn error prompting back on, check it here.
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

### Fortran>Includes Category

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

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 123 and the SciTools Support website.

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

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

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

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

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

For more information, see *C++ > Includes Category* on page 70.

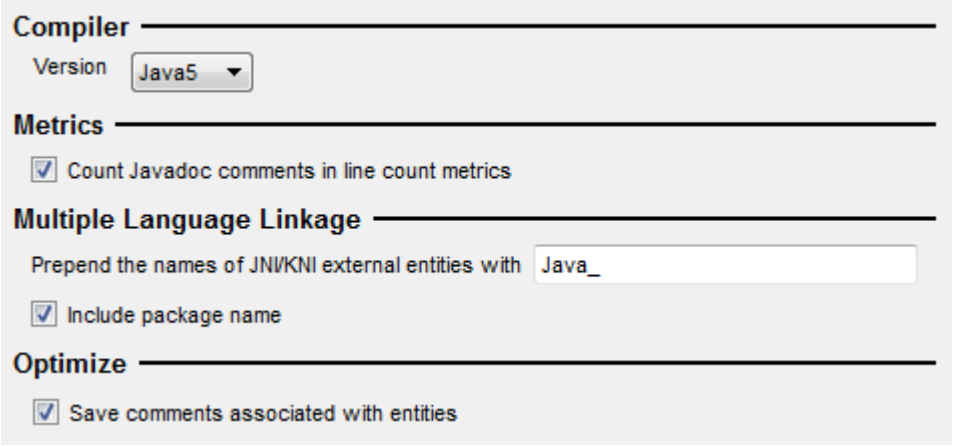
### Other Fortran Categories

For information about the **Fortran > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 72.

For information about the **Fortran > Macros** category, see *C++ > Macros Category* on page 72. 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, and 8.
- **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.
- **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:** Choose whether source code comments that occur before and after an entity should be associated with that entity.

### Java > Class Paths Category

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

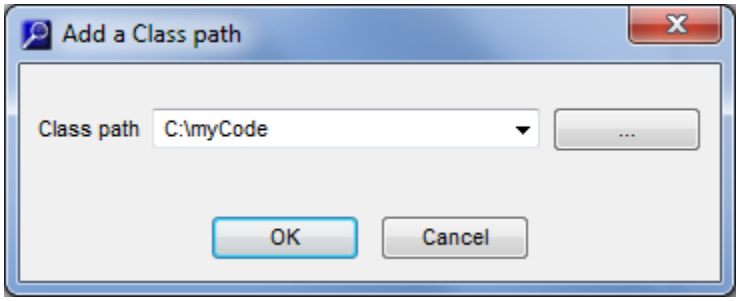
Both .jar files and .class files are supported. Jar files contain compressed .java (source) files. Class files contain compiled sources. By default, the src.jar (or src.zip) file provided by the Java Developers Kit is located. You can add other .jar files as needed.

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

**1** Click **New Path**.

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

3 Click **OK**.




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

1 Click **New Jar**.

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

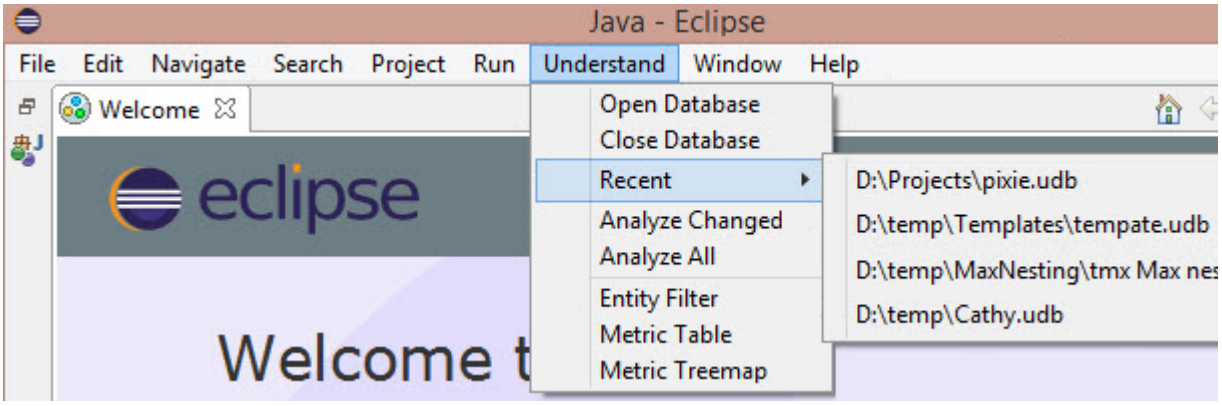
3 Click **Open**.

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

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

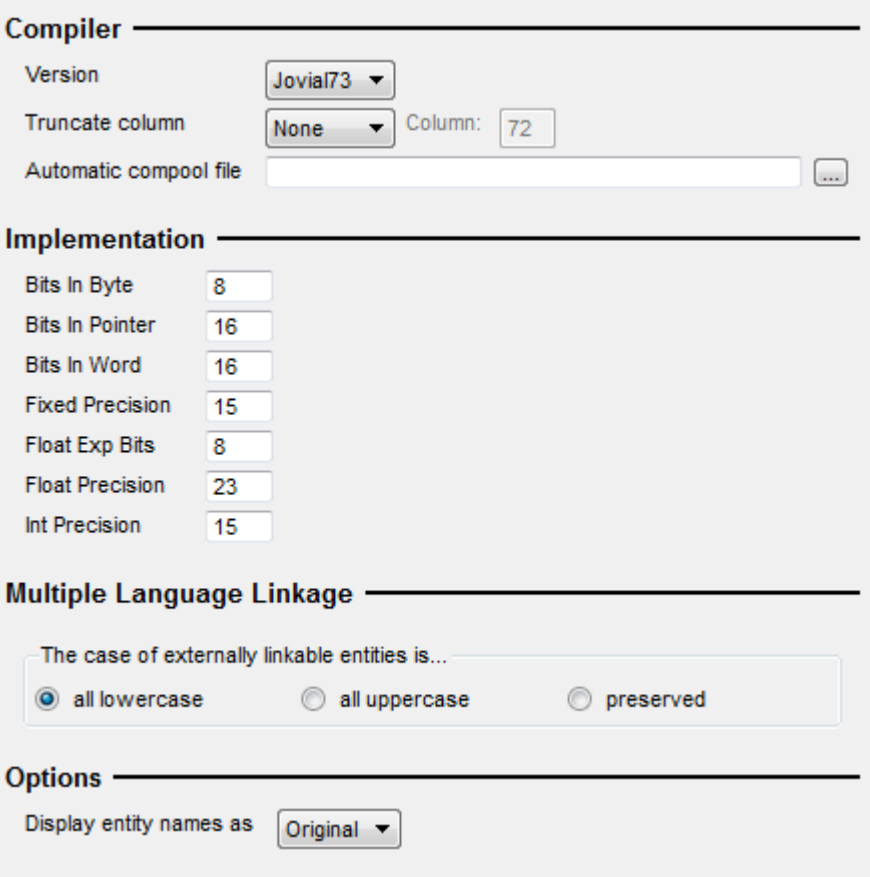
## Eclipse Plugin

If you use the Eclipse IDE for code development, you can access a number of *Understand* features within the Eclipse IDE by installing the Understand plugin for Eclipse. These features include the Entity Filter, Information Browser, Metrics, Treemaps, Butterfly graphs and Control Flow graphs. See http://scitools.com/eclipse/ for details on installing this plugin.



# 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.




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

- **Version:** Select the JOVIAL version you use. JOVIAL73 and JOVIAL3 are supported.
- **Truncate column:** By default, statements are not truncated by column location. You may choose to truncate statements at column 72 or at some other user-defined column.
- **Automatic compool file:** Click **...** and browse to the compool file you want to use. The file extension can be *.txt, *.cpl, or *.jov. The selected file is automatically imported into all other files in the project.
- **Implementation fields:** The fields in this section allow you to specify the sizes and precision of various datatypes. These sizes vary with different implementations of JOVIAL. The sizes are used to determine data overlay. You can specify the number

of bits in a byte, number of bits in a pointer, number of bits in a word, precision for fixed datatypes, number of bits in a floating exponent, precision for floating datatypes, and the precision for an integer.

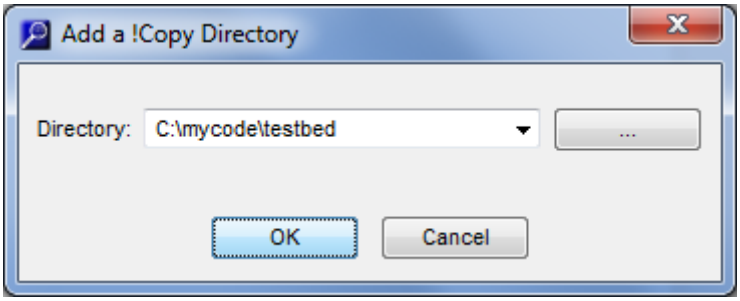
- **Case of externally linkable entities:** Choose which case should be used for "exporting" entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as "MYITEM" and you choose "all lowercase" here, other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

## Jovial > !Copy Category

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

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

**1** Click the **New**.




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

**3** Click **OK**.

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

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

# Pascal Options

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


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

- **Version:** Select the version of Pascal used by the source code in this project. The choices are Delphi, Compaq/DEC/HP, 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) that contains predeclared routines, types, constants, and parameters used in your source code. Two versions of this file are provided in the <install_directory>/conf/understand/pascal directory: predeclared.txt and predeclareddelphi.txt. The default is set according to your choice in the Version field.
- **dfm converter exe:** Browse for and select the executable to be used to convert binary Delphi Form (DFM) files in the project to text files. The text files will then be 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 "exporting" entities in this language that can be linked to (for example, called as functions) by other languages. For example, if an entity is declared in this language as "MYITEM" and you choose "Lowercase" here, other languages would be expected to call that entity as "myitem".
- **Display entity names as:** Choose whether entity names should be displayed in *Understand* with the same case as the source code (original), all uppercase, all lowercase, only the first letter capitalized, or mixed case.

### Pascal > Macros Category

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

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

For more information about the **Pascal > Macros** category, see *C++ > Macros Category* on page 72.

### Pascal > Namespaces Category

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

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

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

To add a namespace directory, follow these steps:

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

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

### Pascal > Standard Library Paths Category

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

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

```
Uses System;
```

The standard libraries are not used when computing project metrics.

To add a directory, follow these steps:

1 Click the **New** button.

2 Click the **...** button and browse to a directory. Then click **OK**.

3 You can click **Move Up** or **Move Down** to change the precedence order in which the standard libraries are checked.

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

**Pascal > Search Paths Category**

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

1 Click the **New** button.

2 Click the **...** button and browse to a directory. Then click **OK**.

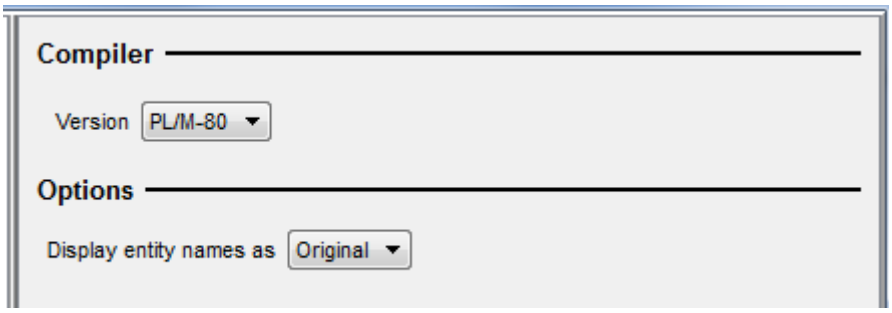
3 You can click **Move Up** or **Move Down** to change the precedence order in which the standard libraries are checked.

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

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

## PL/M Options

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




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

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

## PL/M>Includes Category

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

The configuration of your include file directories is important to improving the accuracy of project analysis. For more about ways to configure these directories, see *Using the Missing Header Files Tool* on page 123 and the SciTools Support website.

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

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

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

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

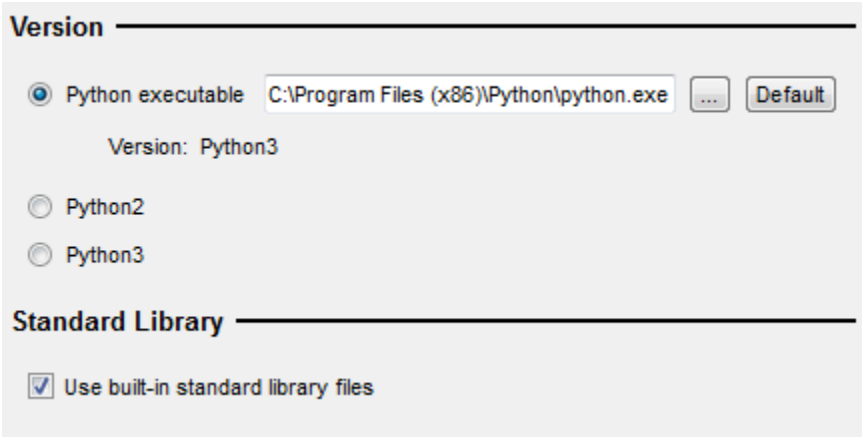
You can import or export a list of include directories from a text file by clicking **Import** or **Export** and selecting the file. The file must contain one directory path per line. (In all such imported text files, a # sign in the first column of a line in the file indicates a comment. Full or relative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

For more information, see *C++ > Includes Category* on page 70.

For information about the **PL/M > Includes > Replacement Text** category, see *C++ > Includes > Replacement Text* on page 72.

# Python Options

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




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

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

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

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

### 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 70.

# VHDL Options

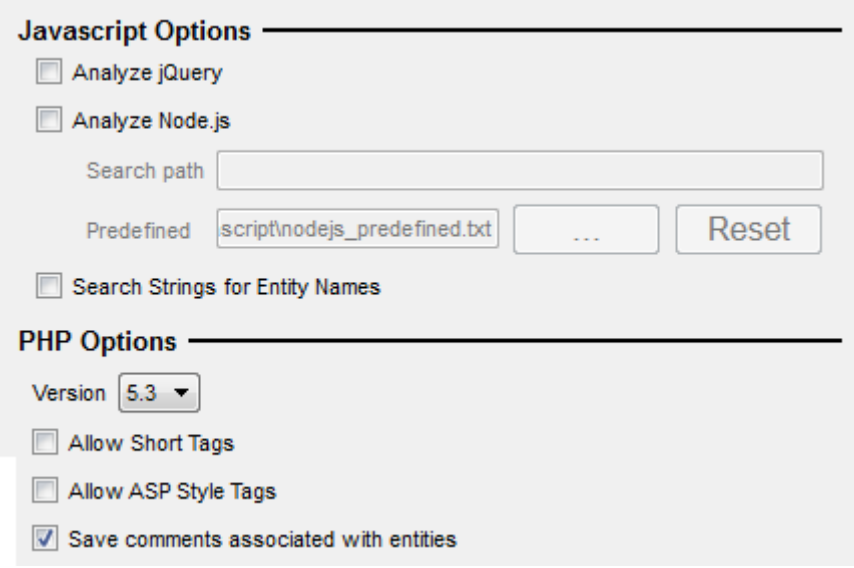
There is currently no Project Configuration category for VHDL.

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

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

# Web Options

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



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

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

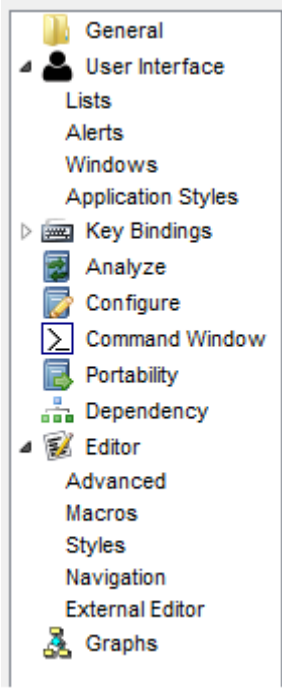
- **Analyze jQuery:** Check this box if you want the analysis to interpret a $(...) call as a jQuery call. A **JQuery Selector** entity is created for the string literal parameter passed to "$". For example, $(".foo") would create a JQuery Selector entity named ".foo". Also, a **jQuery Selector Uses** 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 if you want the analysis 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 that are 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 allows you to specify the name of a file containing a list of predefined node.js modules. This field defaults to the path to scitools\conf\understand\javscript\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.
- **PHP Version:** Select the version of PHP used by your project. The options are 5.3, 5.4, 5.5, and 5.6.
- **Allow Short Tags:** Check this box if your PHP code ever uses the short form of PHP tags.
- **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:** Choose whether source code comments that occur before and after an entity should be associated with that entity.

# Setting General Preferences

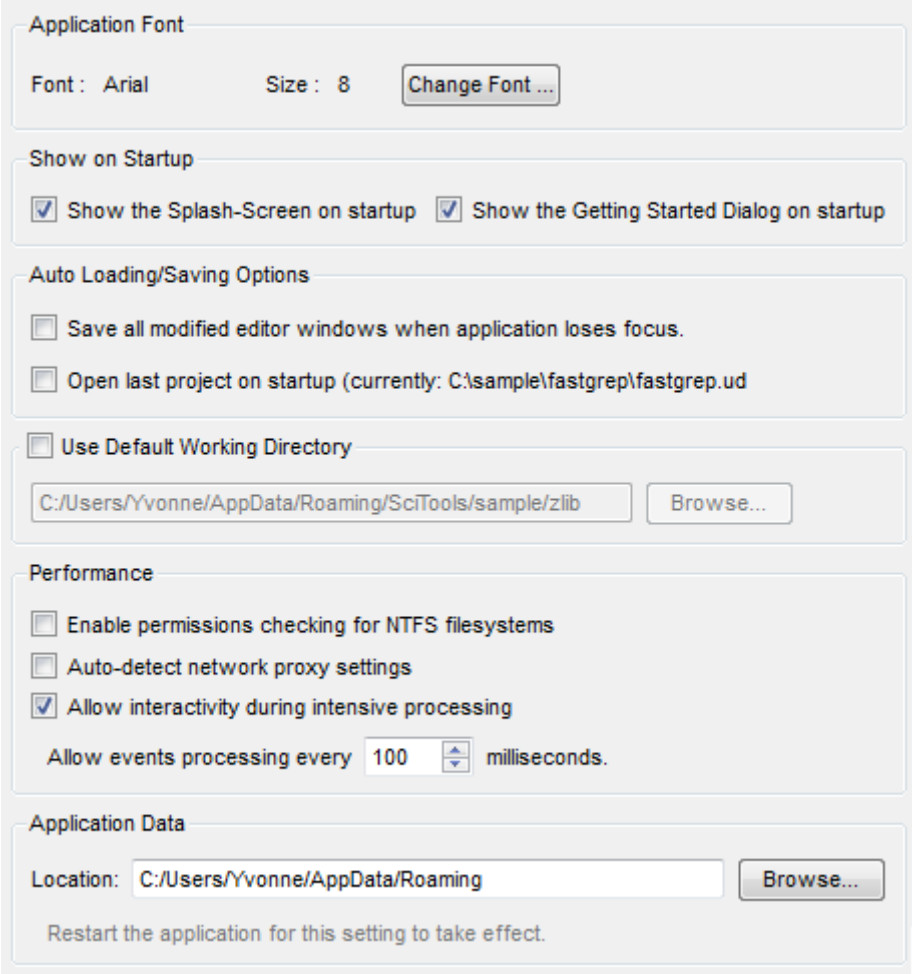
*Understand* allows you to control a number of aspects of its operation using the Understand Options dialog. To open this dialog, choose **Tools > Options**. This dialog provides options to set in the categories shown to the right:




The subsections that follow describe each of the categories:

### General Category

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

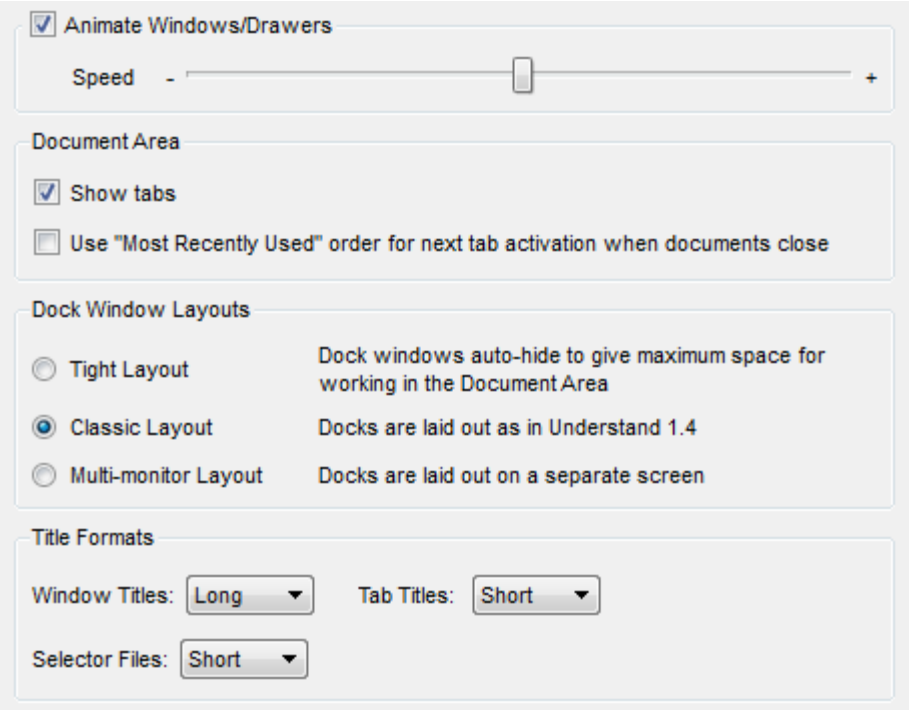



- **Application font:** To change the font used in dialogs and lists in *Understand*, click **Change Font** and select the font, font style, and font size you want to use and click **OK**.
- **Show the Splash-Screen on startup:** If checked (on by default), the logo is shown while *Understand* is starting.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

- **Use default working directory:** If checked (off by default), you can select an alternate default directory. This will be the starting place when you are browsing for other directories and the directory to which relative directory specifications relate. The default is the directory where your project is saved.
- **Enable permissions checking for NTFS filesystems:** If you check this box, file permissions are checked on NTFS filesystems when you use the editor to modify files. This option is off by default, since this checking can significantly degrade performance in some cases.
- **Auto-detect network proxy settings:** If the Getting Started tab does not show the Scientific Toolworks blog feed, you can check this option to have your system's proxy settings checked so that the feed can be loaded. However, scanning for proxy settings takes some time and slows down the *Understand* startup process. This option is disabled by default. After you enable this option, you will need to restart *Understand* in order for it to take effect.
- **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.
- **Application Data Location:** This field shows where files used internally by *Understand* but not associated with a specific project are stored. You can browse to change this location. You will need to restart *Understand* to have changes to this directory location take effect.

**User Interface Category**

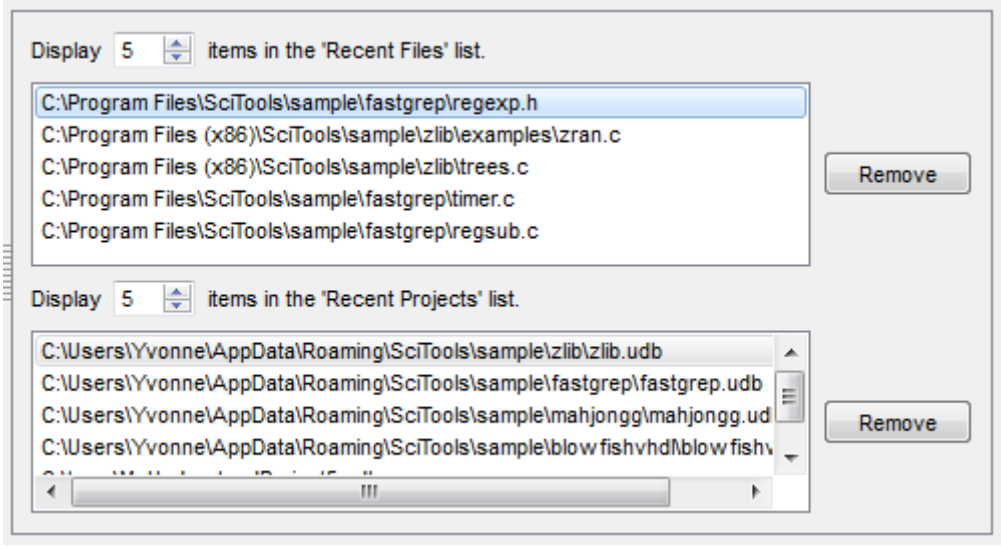
The following options can be set from the **User Interface** category of the **Tools > Options** dialog:




- **Animate Windows/Drawers:** If checked (on by default), opening and closing windows and tabbed areas (drawers) is animated. You can choose a faster or slower speed than the default.
- **Show tabs:** If checked (the default), tabs are shown at the top of the document area for each of the windows open in that area. This includes the source editor windows, graphical views, and other windows.
- **Use “Most Recently Used” order for next tab activation when documents close:** If this box is checked, the most recently used window becomes the current window when you close another. If this box is unchecked (the default), the tab to the left becomes the current window.
- **Dock Window Layouts:** Choose which window layout you would like to use as the default. The **Tight Layout** is useful if you will be opening several source files and want plenty of screen space for that. The **Classic Layout** is similar to earlier versions. The **Multi-monitor Layout** allows you to take advantage of multiple screens if you have them.
- **Title Formats:** Choose whether you want filenames in the title areas of windows, tabs, and selector files to be short names, long (full path) name, or relative to the project database.

**User Interface > Lists Category**

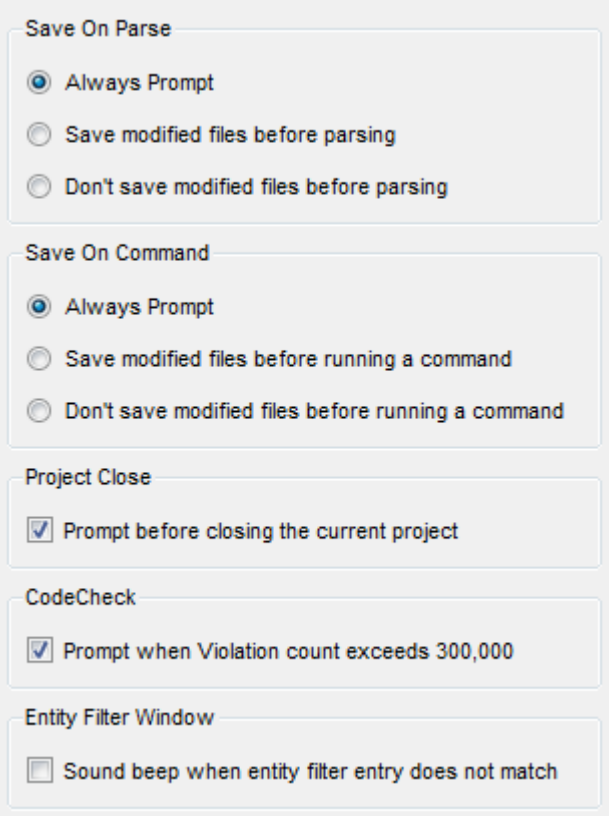
The following options can be set from the **User Interface > Lists** category of the **Tools > Options** dialog:




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

### User Interface > Alerts Category

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

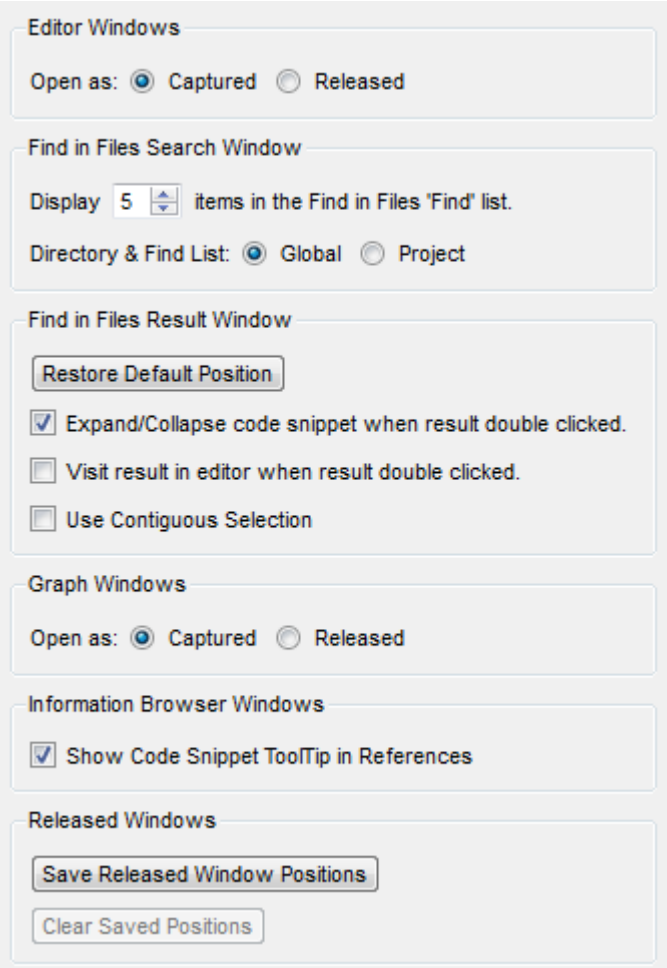



These options can be used to re-enable warnings that you have disabled in a warning dialog box.

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

### User Interface > Windows Category

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




You can choose various options for different types of windows.

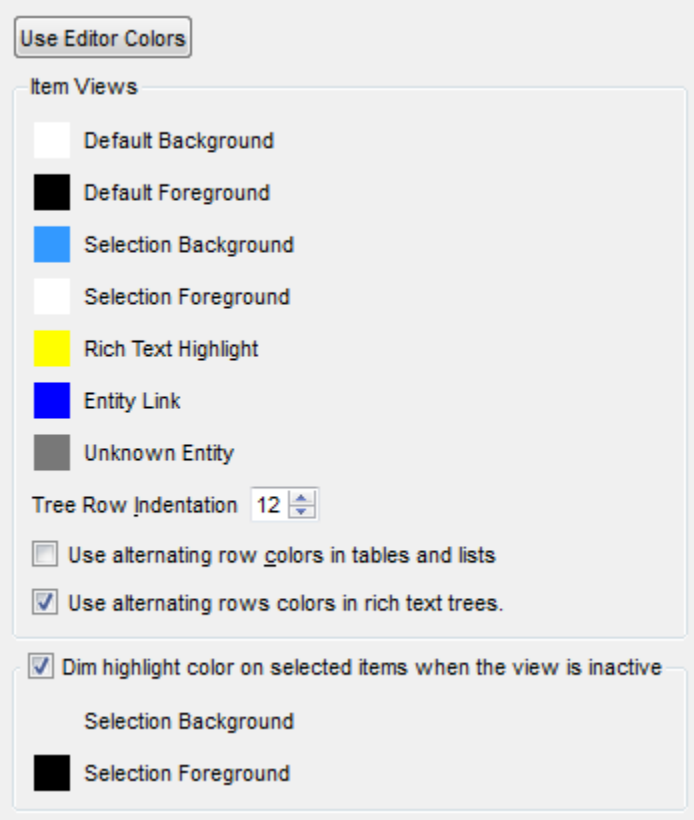
- **Editor Windows:** Choose whether to open source code files as captured windows contained within the document area of the *Understand* window (MDI windows) or as released windows you can move anywhere on your desktop (SDI windows). The default is captured.
- **Find in Files Search Window:**
  - **Display:** 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.
- **Find in Files Result Window:**
  - Click **Restore Default Position** if you want to re-dock the results for the Find in Files dialog to the bottom of the *Understand* window.

  - **Expand/Collapse code snippet when result double-clicked:** By default, code found in the Find in Files results is expanded to show surrounding lines of code when you double-click. Uncheck this box if you don't want this behavior to occur.
  - **Visit result in editor when result double-clicked:** If you check this box, the code is shown in the Source Editor when you double-click on a result.
  - **Use alternating row colors:** By default, the results for Find in Files and CodeCheck have a slightly darker background for every second row. You can turn off this shading by unchecking this box.
  - **Use Contiguous Selection:** Enable this option if you want the Find in Files results to allow you to select multiple rows while holding down the Ctrl or Shift key. This is useful if you want to copy multiple result rows to the clipboard for pasting into some other document. This option is off by default.

- **Graph Windows:** Choose whether to open graphical views as captured windows contained 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.

### User Interface > Application Styles Category

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

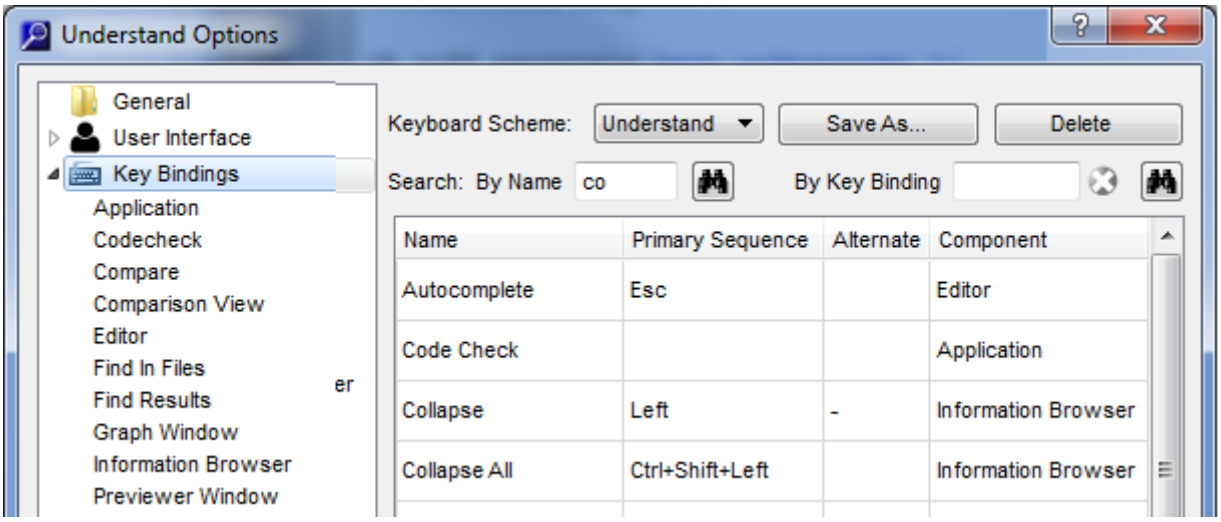



You can choose various options for different types of windows.

- **Use Editor Colors:** Click this button to apply the source editor colors from the Editor > Styles category (page 115) to item views, such as the Information Browser.
- **Item View 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.
- **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 have a slightly darker background for every second row. You can turn off this shading by unchecking this box.
- **Dim highlight color on selected items when the view is inactive.** The Windows default is to dim the highlighting for the selected object when a windows loses focus. If this makes it difficult for you to read the selected object, you can change the behavior by unchecking this box or by changing the background and foreground colors for such items.

## Key Bindings Category

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



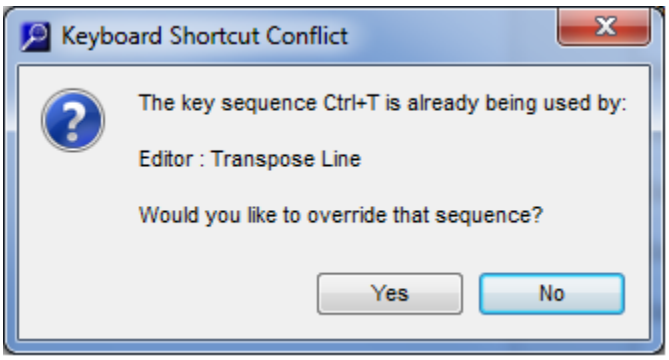

- **Keyboard Scheme:** This field allows you to choose groups of keyboard settings that are similar to other applications. The default settings are those native to *Understand*. Other choices are Visual Studio .NET key bindings and the Emacs editor key bindings. If you choose a scheme and click **OK**,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

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

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

1. Use the Component categories or the Search fields to find a command whose key binding you want to change.
2. Put your cursor in the **Primary Sequence** or **Alternate** column for the command you want to modify.
3. Press the key combination you want to use to perform that action.
4. You can't use normal editing keys like Backspace or Delete to edit the keys shown in these fields. To delete the key combination you have entered, click the **X** in the red circle.

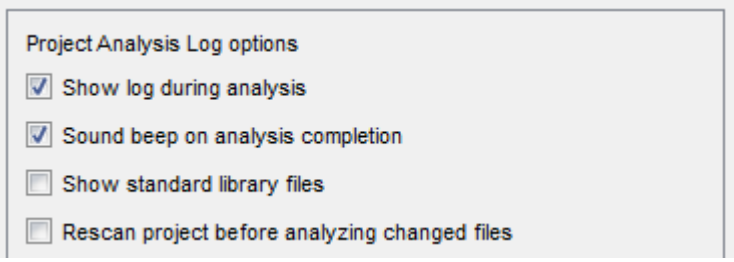
5 When you move focus away from a key binding you changed, you may see a warning message if the key combination you chose is already used. For example:




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

## Analyze Category

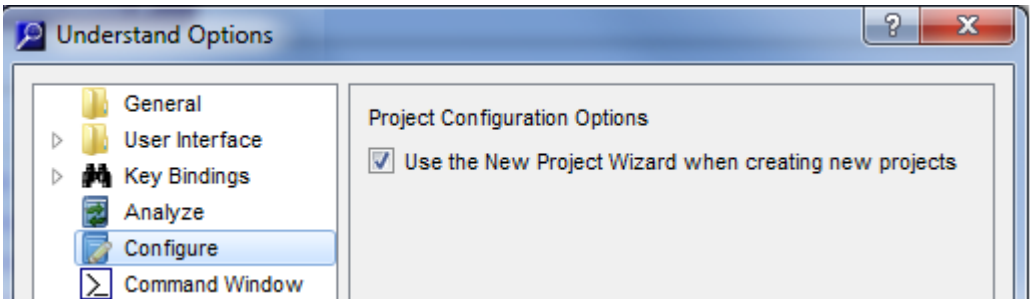
The **Analyze** category of the **Tools > Options** dialog allows you to specify options for how the project is analyzed.




- **Show log during analysis:** By default, the Analysis Log is shown while the analysis is being performed. If you uncheck this box, the Analysis Log area is not shown when an analysis is running.
- **Sound beep on analysis completion:** By default, a beep notifies you when the analysis is complete.
- **Show standard library files:** For languages whose standard libraries are analyzed by *Understand* (such as Ada), if you check this box the standard library files are shown in the Analysis Log. By default, this box is not checked, and the Analysis Log is shorter.
- **Rescan project before analyzing changed files:** If you check this box, *Understand* scans for files that have been added to project directories and to any Visual Studio projects referenced by this *Understand* project before analyzing the files currently in the project. This has the same effect as using the **Project > Rescan Project Directories** menu command before analyzing the project. By default, this option is off.

### Configure Category

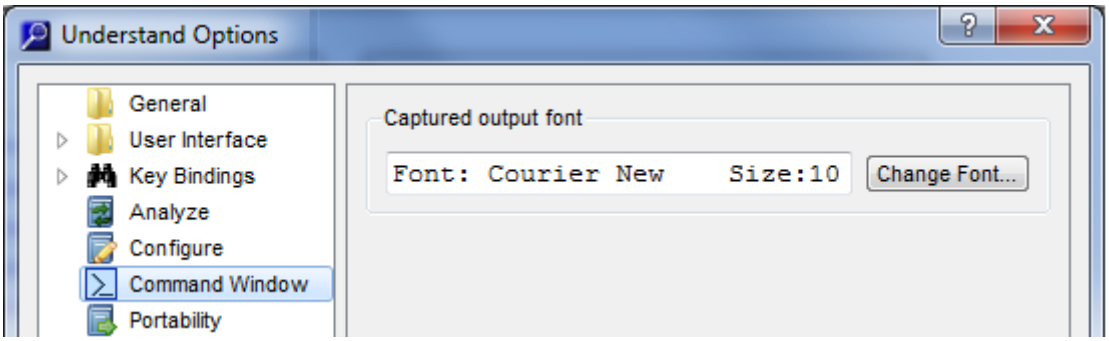
The following options can be set from the **Configure** category of the **Tools > Options** dialog:




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

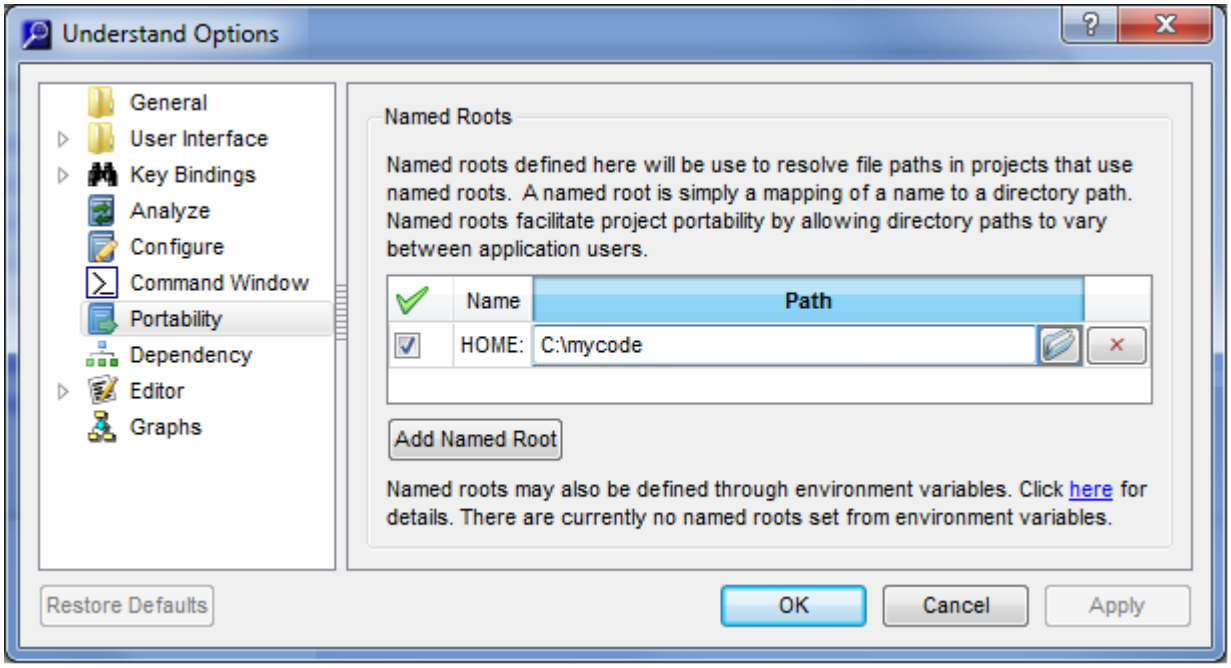
### Command Window Category

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




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

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




After you have defined a named root, you can use that name in other *Understand* dialogs, such as the Project Configuration, and in “und” command lines (see page 320). This is useful, for example, if you want to share projects with people who reference project files over a network using different paths.

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

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

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

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

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

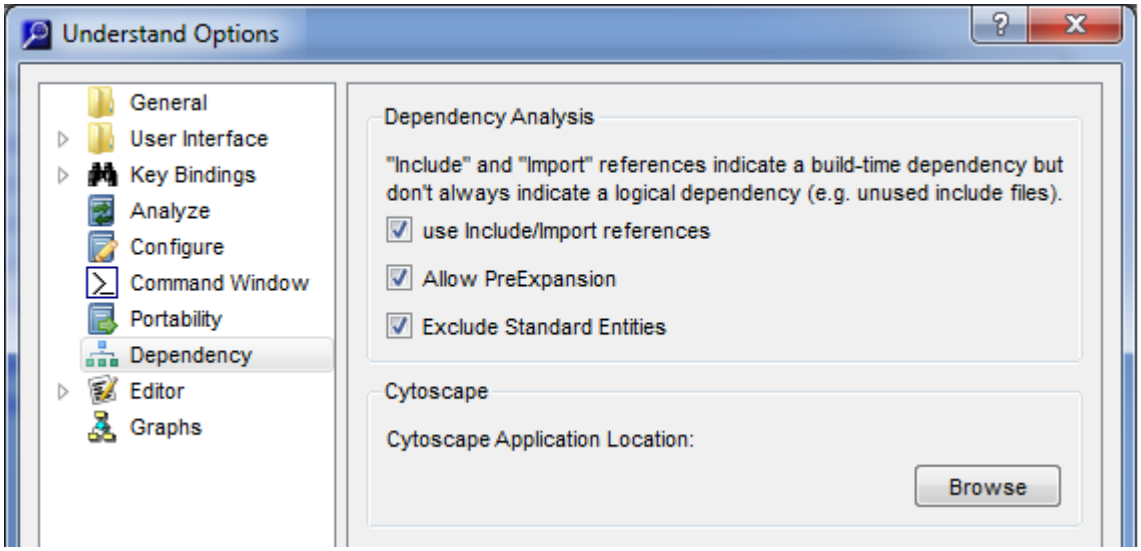
The named root that you would then use within *Understand* is “SOURCEDIR”.

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

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

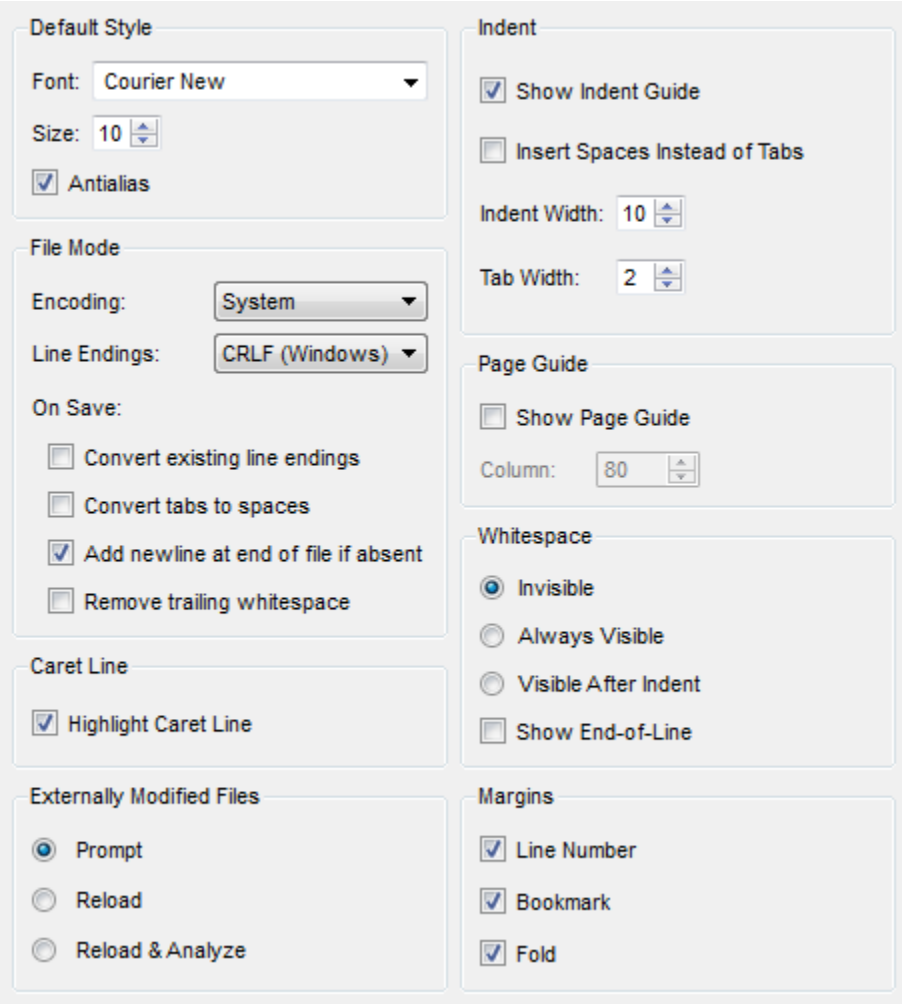
## Dependency Category

The Dependency category of the **Tools > Options** dialog lets you set options related to the Dependency Browser (page 140), dependency graphs (page 196), and dependency exports.




- **Use Include/Import References:** By default, “includes” and “imports” are treated as dependencies. However, you may want to omit such relationships from dependency lists if they are required for building but are not logically dependent.
- **Allow PreExpansion:** By default, nodes in the Dependency Browser are expanded. You can uncheck this box to have them closed by default.
- **Exclude Standard Entities:** By default, entities in the standard libraries are excluded from dependency reports, dependency graphs, and the dependency browser. Uncheck this option to include such standard entities.
- **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 244.

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

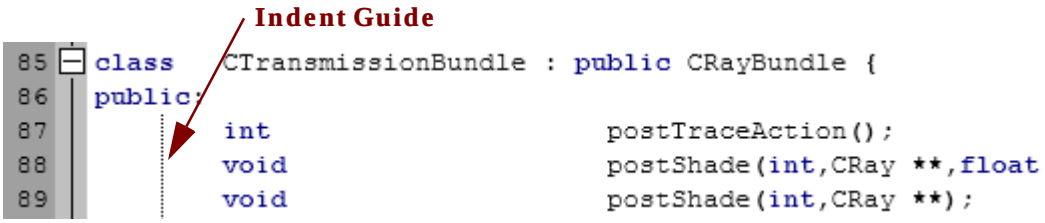



- **Default style:** Use the **Font** pull-down list to select a font for Source Editor windows. The fonts shown are the fixed-width fonts available on your system. Select a **Size** for the Source Editor text. If you check the **Antialias** box, the font is smoothed. The fields in this area set the default size. You can change it on a per-file basis by choosing one of the **View > Zoom** menu options.
- **File Mode:** Select the type of **Encoding** to use when saving source files and the **Line Endings** character you want used. Many encoding formats are supported. The “System” encoding 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 referred to as a linefeed (LF).
- **Classic Macintosh** line-endings are terminated with a single carriage return (CR).

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

- **Caret Line:** Check the **Highlight Caret Line** box if you want the full line on which your cursor is located to be highlighted.
- **Externally Modified Files:** If an open file is changed in some other program, *Understand* detects this. Choose **Prompt** if you want to be notified and asked to load that changed version. **Reload** and **Reload & Analyze** do these actions without prompting.
- **Indent:** Check the **Show Indent Guide** box if you want a dotted line to show to column to which lines should be indented.

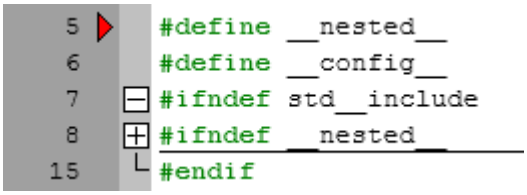



By default, the **Insert Spaces Instead of Tabs** box is off; turning it on adds spaces to a source file when you press <Tab>.

For **Indent Width**, specify the number of columns in an indentation level. For **Tab Width**, specify the number of columns for each tab stop. For example, if you set the Tab Width to 4, each <Tab> moves 4 columns to the right. If you set Indent Width to 6 and Tab Width to 4, each automatic indentation level is made up of one <Tab> and 2 spaces. You can set a tab width for a specific file to override the project-wide tab width (see page 180). Also, see *Editor > Advanced Category* on page 111 for advanced indentation options.

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

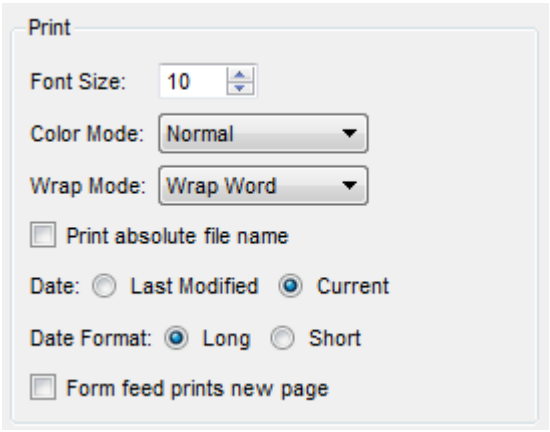
- **Margins:** Check **Line Number** (on by default) to turn on line numbering in the source view. Check **Bookmark** (on by default) if you want bookmarks (red arrows) shown in the margin next to line numbers. Check **Fold** (on by default) to turn on the ability to “fold” source code entity blocks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




### Editor > Advanced Category

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

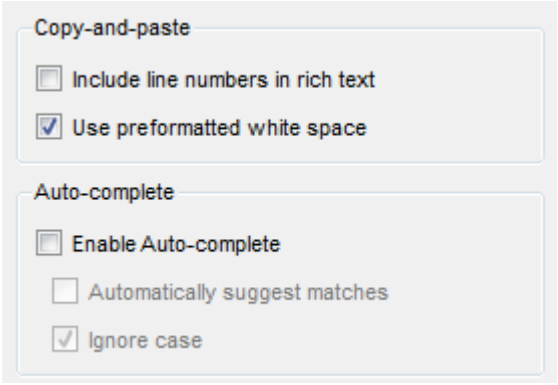
The following options control how source code looks when you print it from an editor window:




- **Font Size:** Choose the size of the source code you want to use for printing. To zoom in and out in an individual source code window, see page 183.
- **Color Mode:** Choose a color mode for printing. The choices are as follows. Note that colors other than black and white are printed only if you are using a color printer and the printer driver is set to print in color.
  - “Normal” matches the current display appearance.
  - “Invert Light” prints black as white and white as black. This is useful if you set the background to a dark color and the text to light colors for your display.
  - “Black on White” prints black code on a white background regardless of the current display appearance.
  - “Color on White” prints colored code on a white background regardless of the current display appearance.
- **Wrap Mode:** Choose the wrap mode you want to use for printing. The default is to wrap words to the next line, but you can choose to truncate lines or wrap at the character level, which breaks words across lines. The line breaks displayed are for printing only; no actual line breaks are added to your source file. See *Line Wrapping* on page 180 to change the wrap mode for screen display.
- **Print absolute file name:** Check this box if you would like the full file path printed at the top of a source file printout, rather than just the filename.
- **Date:** Choose whether to show the date a file was last modified or the current date when printing. The default is the current date.

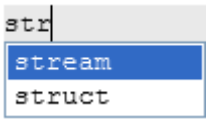
- **Date Format:** Choose whether to print the date in long or short format. Your system's preferred long and short date format are used.
- **Form feed prints new page:** If this box is checked, a form feed character in the source code file causes a page break. If you uncheck this box, form feed characters are printed as "FF" and no page break occurs.

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

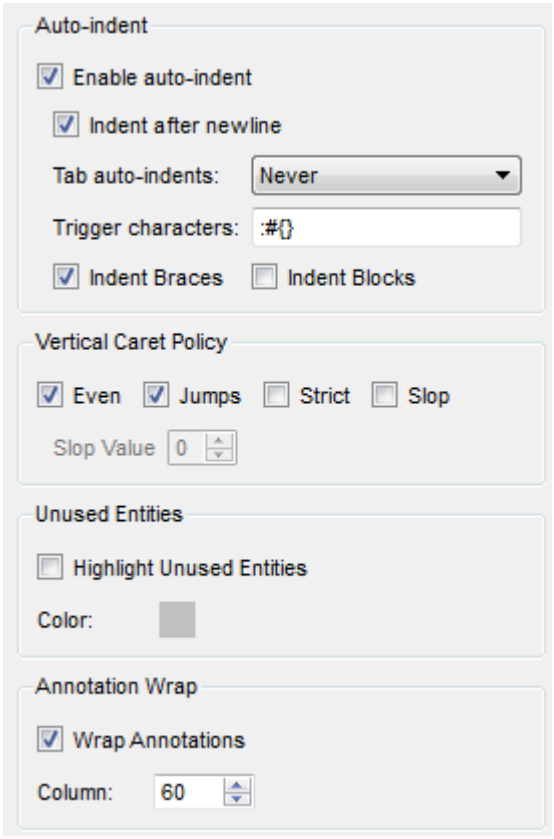



- **Include line numbers in rich text** pastes line numbers (in bold). HTML is used to format the pasted text. This option is off by default.
- **Use preformatted white space** pastes code using HTML <pre> tags to preserve whitespace. If you disable this option, whitespace is preserved using   (non-breaking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

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




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




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

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

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

- **Indent blocks:** If you check the **Indent braces** box, you can also check this box, which causes the block of code within the braces to be indented further as in the following example:

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

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

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

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

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

### Editor > Macros Category

You can record, save, and replay Source Editor macros as described page 180. After you have saved Source Editor macros, you can rename and delete macros using the Options dialog. Follow these steps:

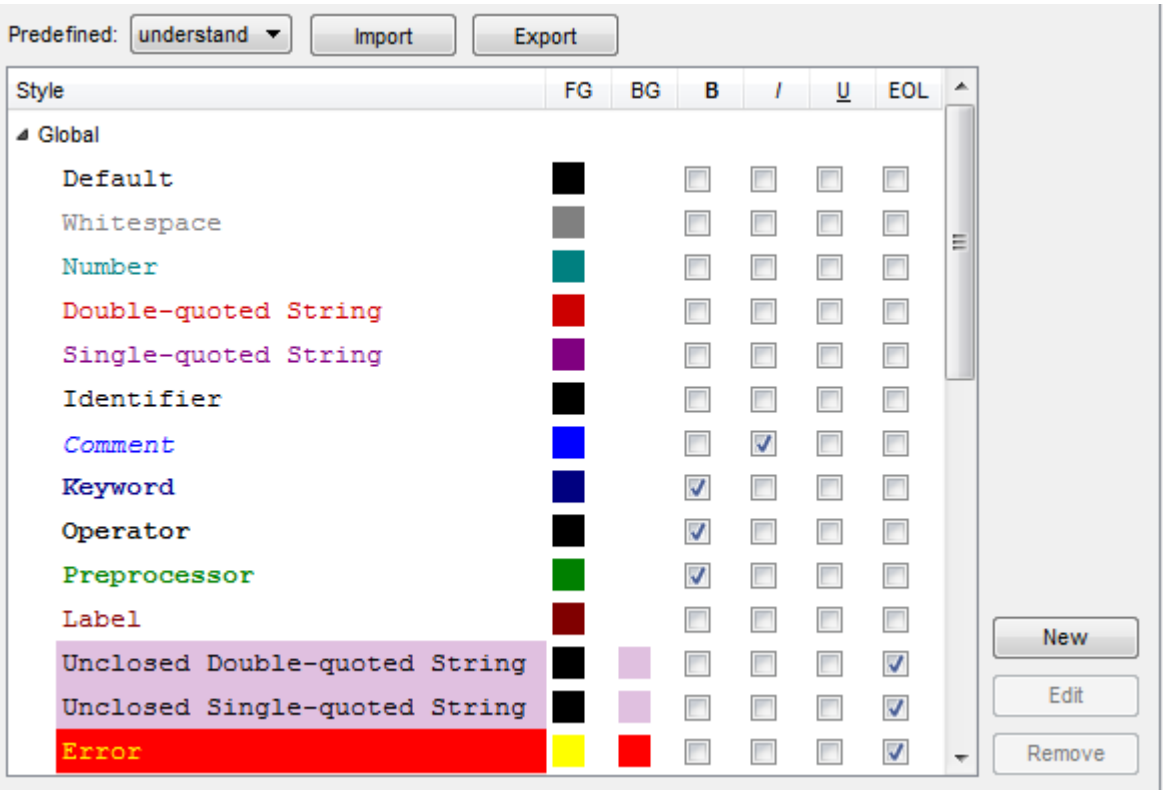
1. Choose **Tools > Options**, expand the **Editor** category, and select the **Macros** category.
2. In the top box, choose the macro you want to configure.
3. Click **Edit** if you want to rename the macro or assign a different key sequence to trigger it. Note that you cannot edit the actions performed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

## Editor > Styles Category

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

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

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




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

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

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

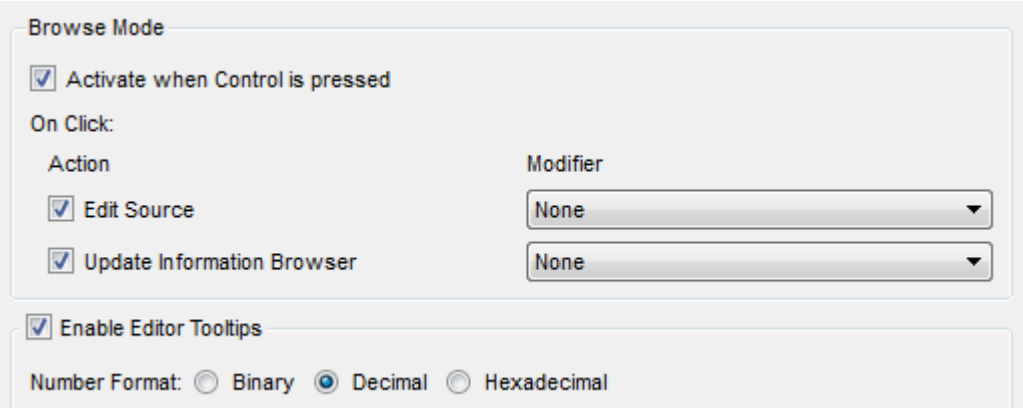
- **Dark blue text:** Used for language keywords
- **Red text:** Used for characters and character strings
- **Italic blue text:** Used for comments
- **Green text:** Used for preprocessor statements
- **Black text:** Used for all other source text and for line numbers
- **White background:** Used for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

Additional items are available for customization depending on your source code language. For example, with C++, you can customize class, enumerator, and namespace 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 package names, subprogram names, and type names.

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

### Editor > Navigation Category

You can control the behavior of Browse Mode (see page 168) in the Source Editor. To see this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Navigation** category.




- **Activate when Control is pressed:** If this 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 an entity when you click on it. You can choose a key that must be pressed along with the click to have this action occur. The default is that no key is required along with the click.
- **Enable Editor Tooltips:** Check this box if you want to see brief information when the mouse cursor hovers over and 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.

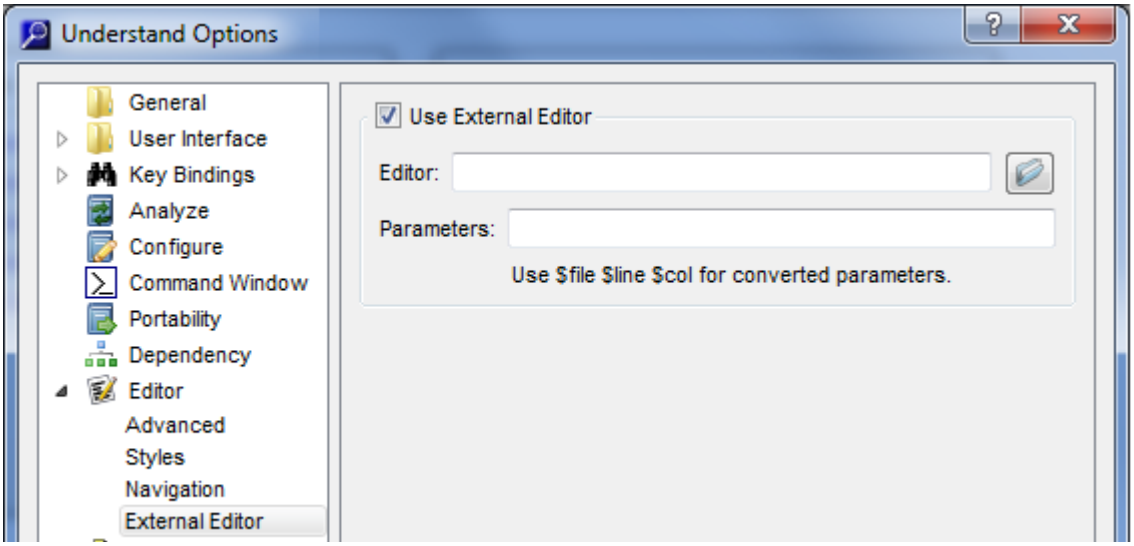
### Editor > External Editor Category

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. In the Select an External Editor dialog, check the **Use External Editor** box if you do not want to use *Understand* for editing.




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

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

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

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

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

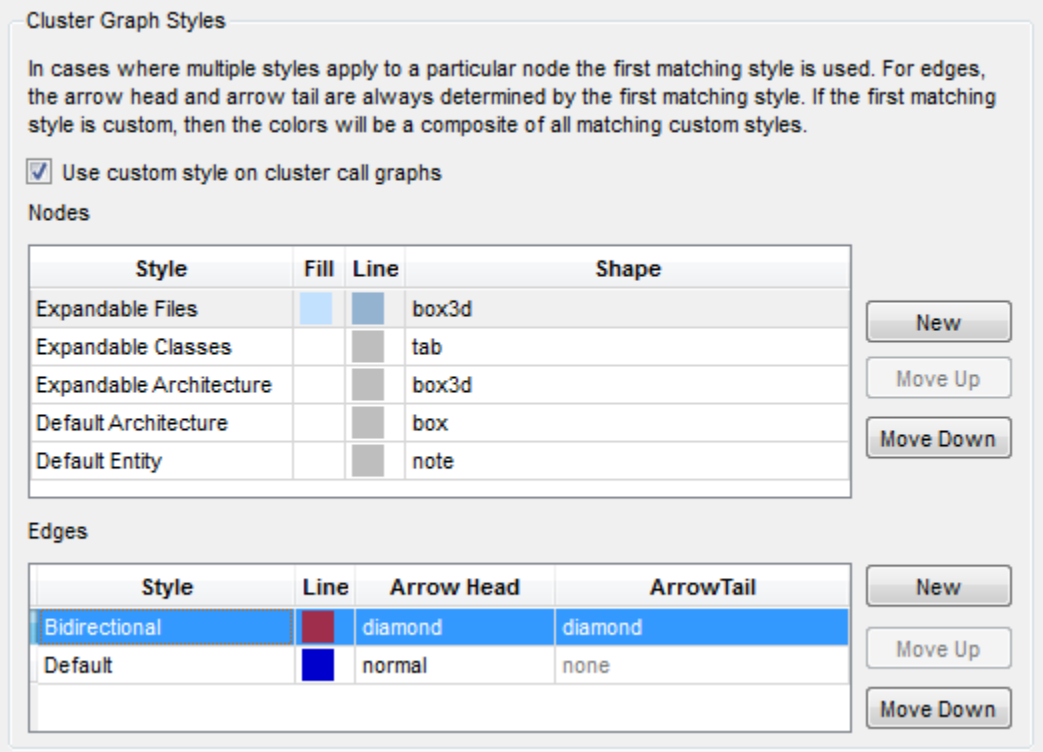
The *Understand* context menus (also called right-click menus) can be made usable in external editors. Steps for doing this are provided in the SciTools blog. For EMACS, vi, and Visual Studio, see scitools.com/support/understand-context-menu-in-ema. For SlickEdit, see scitools.com/support/using-understand-with-an-exter-2.

## Graphs Category

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


- **Highlight edges on hover:** If this option is enabled, relationships (connecting lines) within a dependency 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 196.
- **Colored cluster backgrounds:** By default, the background of a cluster is colored. Uncheck this box to hide such colors.
- **Animate Transitions:** By default, when you expand or compress a node in a cluster graph, 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.
- **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.
- **Maximum References Shown in Edge Context Menu:** If you right-click on an 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 **Graphs** category also lets you customize the display colors, shapes, and arrows for cluster graphs. By default, the settings you make in this area apply only to Architecture Dependency graphs (page 196). If you check the **Use custom style on cluster call graphs** box, your style settings also apply to cluster call graphs (page 272).




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

If multiple node styles apply, the first matching style in the list is used. You can use the **Move Up** and **Move Down** buttons the change the order of the styles.

You can create custom styles for nodes as follows:

**1** Click the **New** button for Nodes.

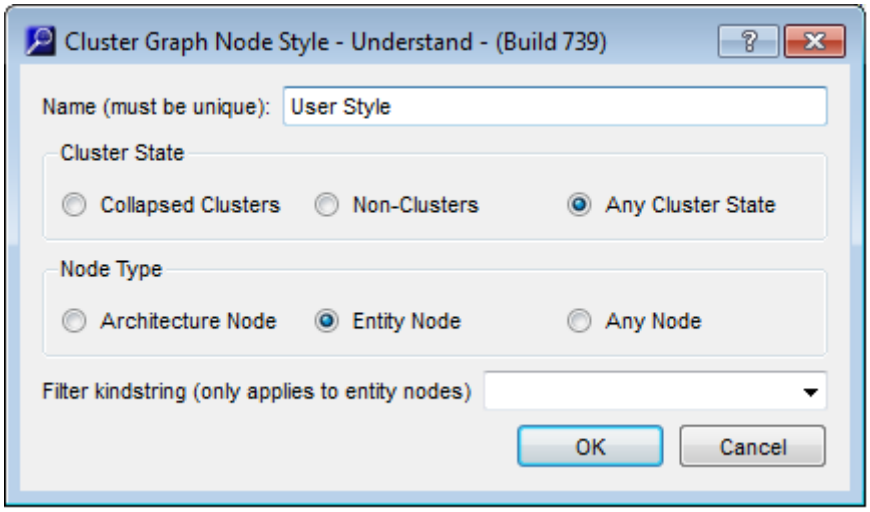
**2** Type a unique name for the new style.

**3** Choose whether this style should apply to only collapsed clusters, only non-clusters, or any cluster.

**4** Choose whether this style should apply to only architecture nodes, only entity nodes, or all nodes.

**5** If this is an entity node style, you can choose a way to filter entities that should have this style. Several sample filters are provided, and you can modify the suggested filters to create your own. For example, the “local object” filter applies the style to

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




If multiple edge styles apply and the first matching edge style is "Default" or "Bidirectional", then only the first style is shown. However, if the first matching style is a custom style, then multiple edges are drawn for each matching custom style. The arrow head and arrow tail for the first matching custom style are used for all arrows.

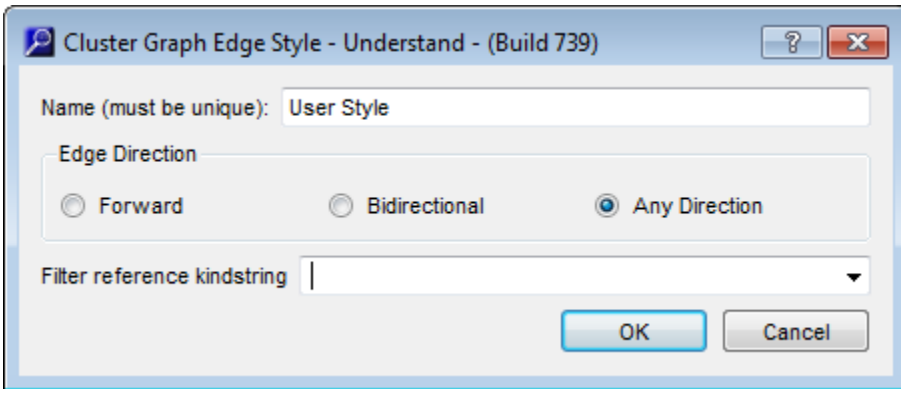
You can create custom styles for edges as follows:

**1** Click the **New** button for Edges.

**2** Type a unique name for the new style.

**3** Choose whether this style should apply to only forward references, bi-directional references, or all references.

**4** You can choose a way to filter the edges that should have this style. Several sample filters are provided, and you can modify the suggested filters to create your own. For example, the “inherit, inheritby” filter applies the style only to inheritance relationships. The “call ~inactive, callby ~” filter applies the style only to call and callby relationships that are active.

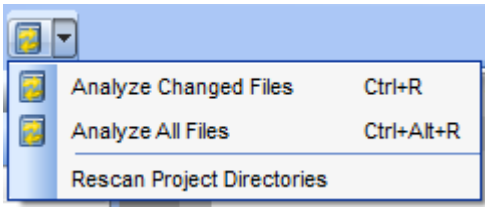


# Analyzing the Code

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

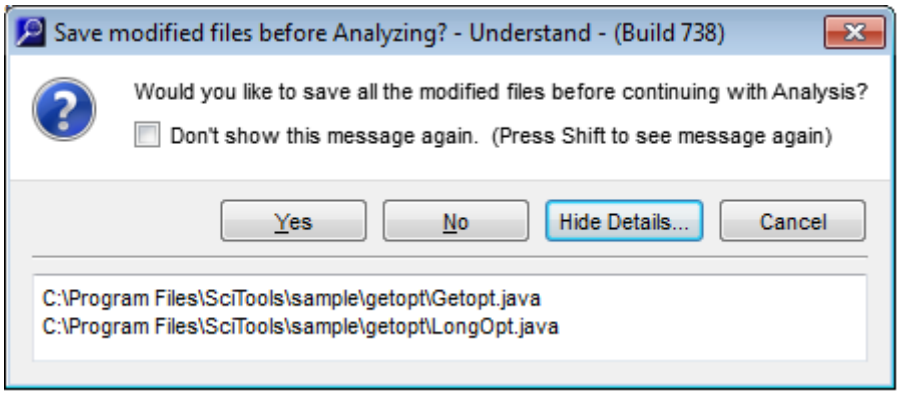
When you save or modify the project configuration, a prompt to analyze the project appears automatically. You can also analyze the project in the following ways:

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



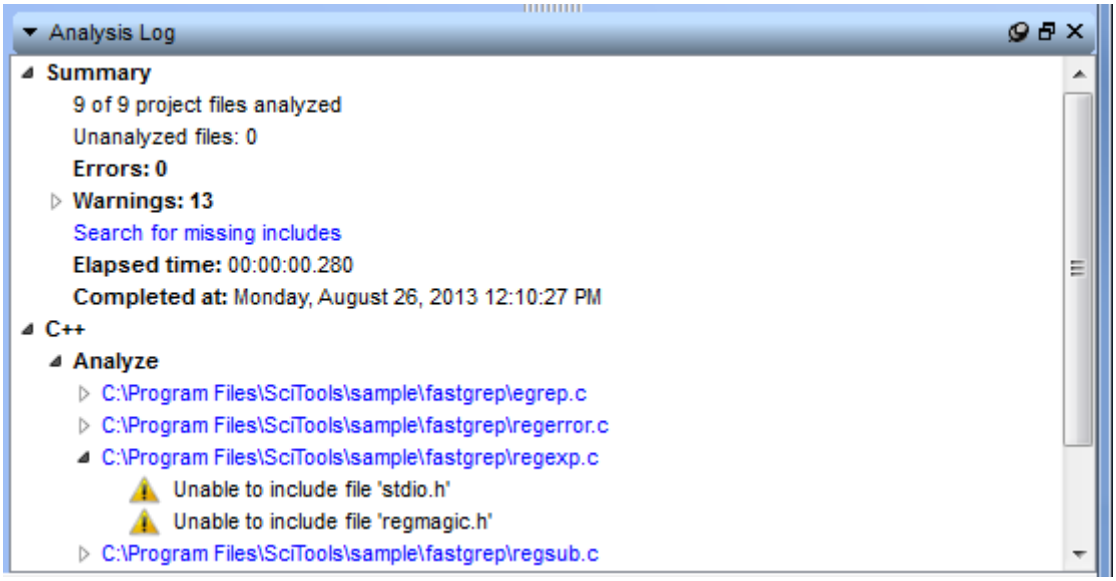

**Project > Analyze All Files:** This menu command forces a full analysis of all project files, whether they have changed since the last analysis or not. (Ctrl+Alt+R)

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




Analyzing a large project can take some time. If you click **Cancel** while the project is being analyzed you will see a message that says this action will leave the project in an incomplete state. You will need to analyze the project in order to explore it. You can choose to **Abort** the analysis and revert to the previous analysis, **Stop** the analysis and keep the incomplete data, or **Continue** the analysis process.

For either command, the *Analysis Log* shows the results as the analysis is being performed.




When the analysis is complete, the source code for any errors or warnings may be examined by double-clicking on the message in the *Analysis Log* window. 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 123).

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** to 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 105 for options that affect the project analysis.

You can schedule automatic project analysis. See page 50 for details.

***Tip:*** A configured project may be analyzed in batch mode using the command line program *“und”*. Refer to *Using the und Command Line* on page 320 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 123 for how to use the Missing Header Files tool.

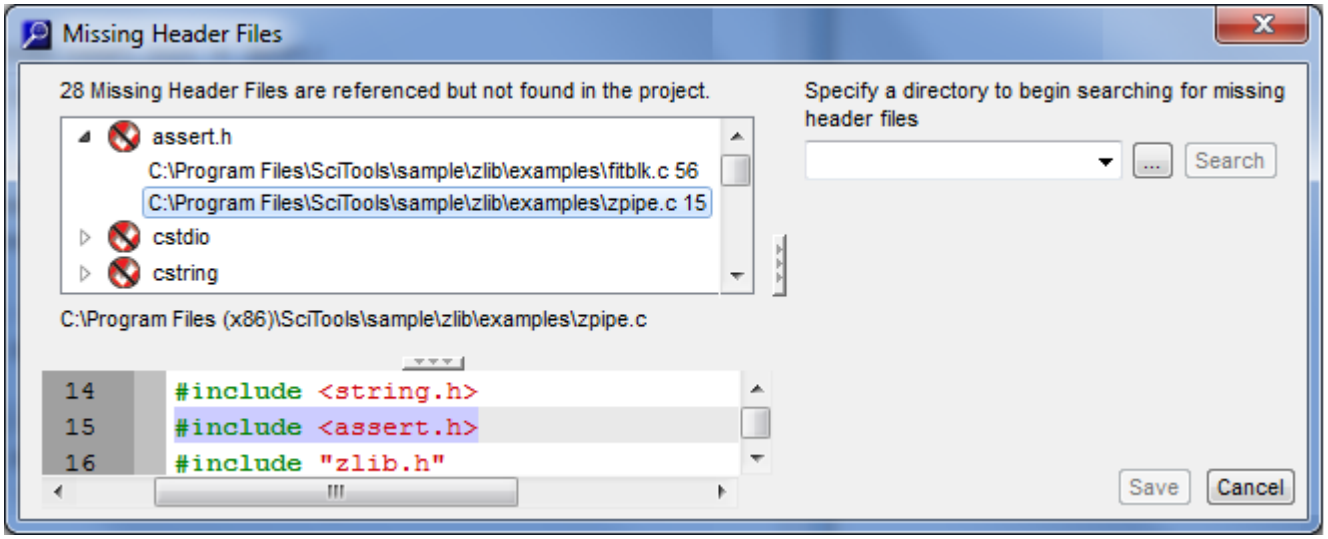
If your project analysis results in warnings about undefined macros, choose **Project > Improve Project Accuracy > Undefined Macros** and see page 125 for how to use the Undefined Macros tool.

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

## Using the Missing Header Files Tool

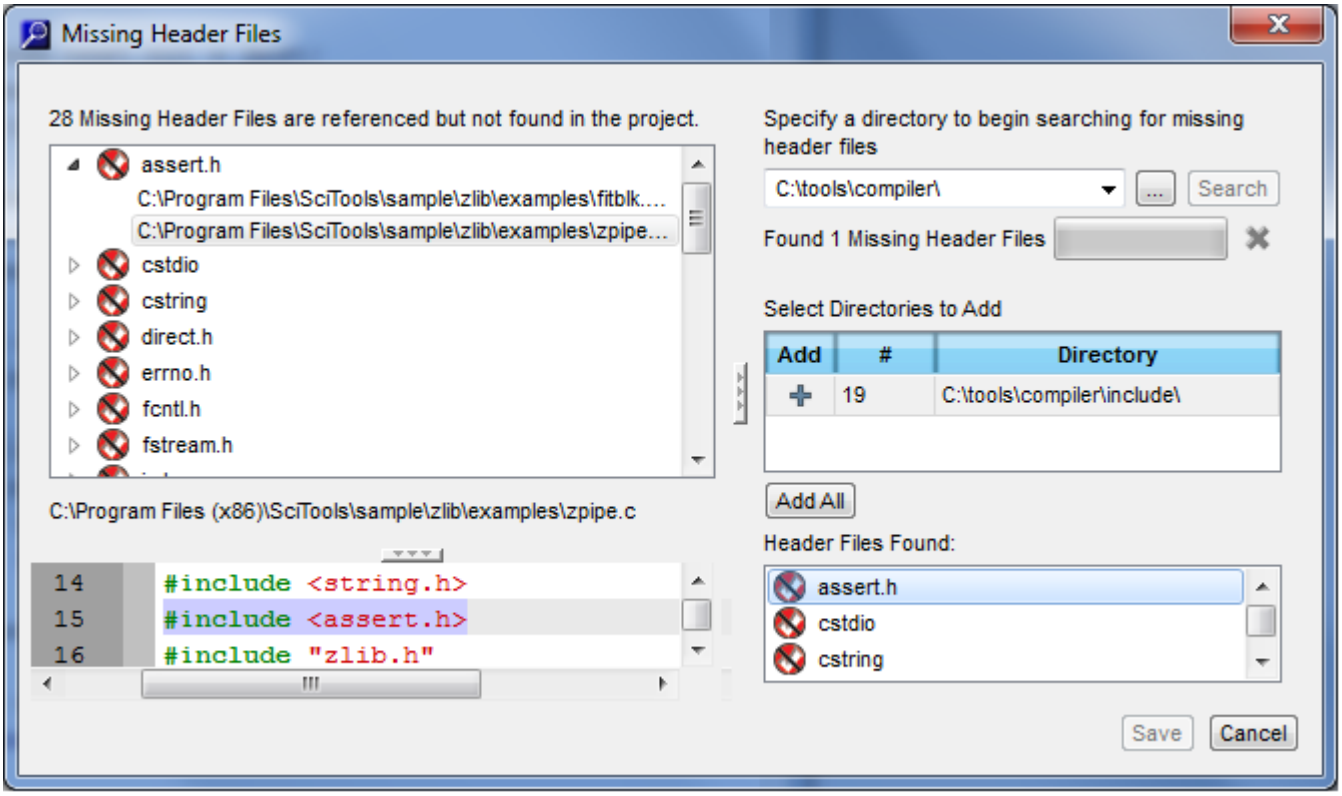
Configuring your include file directories is important to improving the accuracy of project analysis. If your project analysis results in warnings about missing files, use the Missing Header Files tool as follows:

1. Choose **Project > Improve Project Accuracy > Missing Includes** or double-click the link to “Search for missing includes” in the Analysis Log.
2. 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.




3. Click the **...** button in the upper-right corner of the dialog 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 Missing Header Files dialog.




5 If any of the header files you want to use in the analysis are found, click the **+** icon next to a directory you want to add or the **Add All** button below the list.

6 The list of missing headers files is updated to show which header files remain missing with a red icon and which files have been found with a green icon. You can continue searching and adding additional directories as needed.

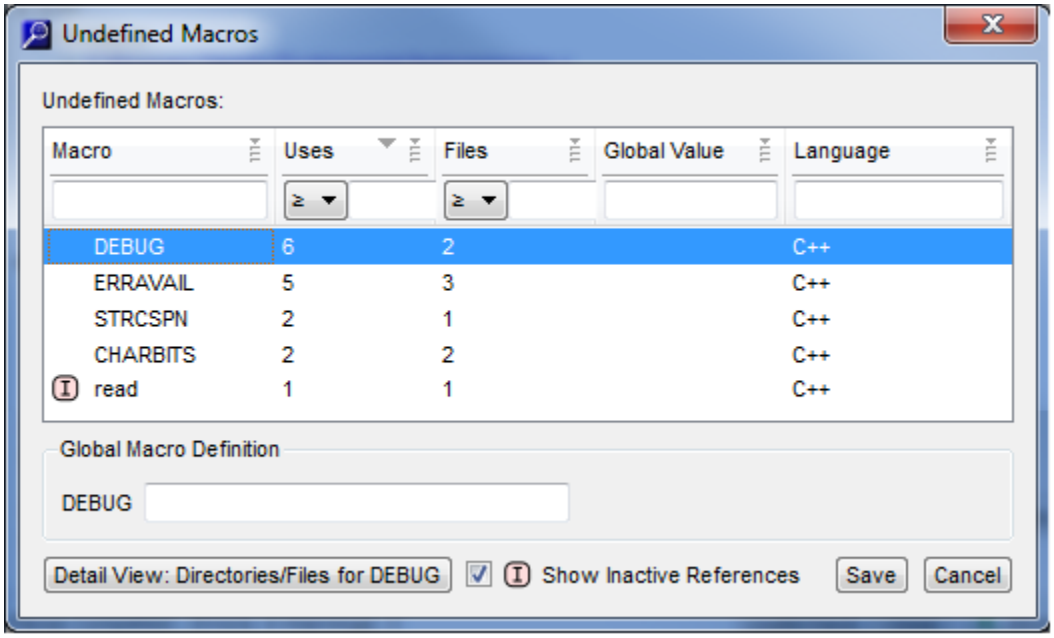
7 Click **Save** to apply your changes to the project configuration.

8 Click **Yes** at the prompt that asks if you want to analyze the project now.

## Using the Undefined Macros Tool

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

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




2 Select a macro from the list. You can use the headings and fields at the top to sort and filter the list and the **Show Inactive References** box to show or hide such macros. See *Filtering the List* on page 157 for more about using these filter fields.

3 Type a definition for the macro in the Global Macro Definition area.

4 You can click the **Detail View** button to see the code where the selected macro is used. In this view, you can define a macro value for a specific file or folder instead of project-wide.

5 You can select other macros and type definitions for them before saving your changes.

6 Click **Save** to apply your changes to the project configuration.

7 Click **OK** in the Project Configuration dialog to save the project configuration.

8 Click **Yes** at the prompt that asks if you want to analyze the project now.

# Chapter 4 Exploring Your Codebase

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

Details on the use and operation of the Entity Locator and Find in Files for searching for and locating entities are provided in the chapter *Searching Your Source* on page 146.

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

This chapter contains the following sections:

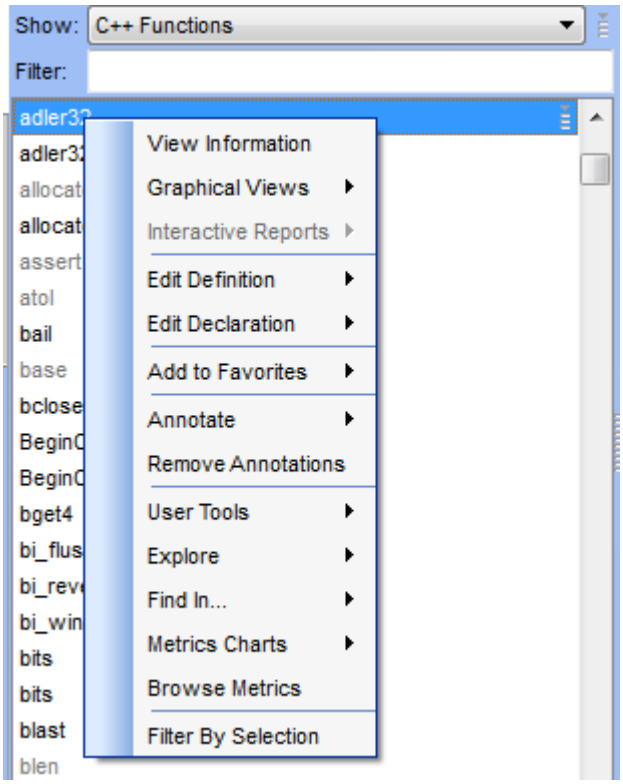
# PLEASE RIGHT-CLICK

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

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

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

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




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

**Right-click almost anywhere to bring up a menu.**

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

# Various Windows Explained...

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

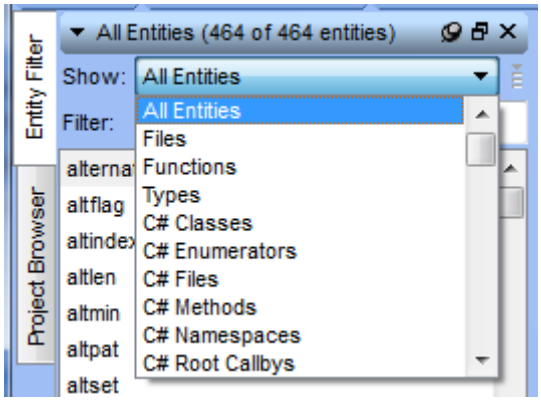
The tools available for finding and exploring entities are:

- **Entity Filter:** Provides an alphabetic list of entities of the selected type. See page 129.
- **Information Browser:** Provides an explorer for entity characteristics and connections. See page 131.
- **Project Browser:** Lets you browse a hierarchical file list. See page 137.
- **Exploring View:** Lets you browse a relationship hierarchy. See page 139.
- **Dependency Browser:** Lets you browse dependency relationships. See page 140.
- **Favorites:** Lets you provide quick links to frequently-used entities. See page 142.
- **Entity Locator:** Lets you filter all entities in a project in complex ways. See page 155.
- **Find in Files:** Searches multiple files. See page 150.
- **Source Editor:** Shows source code. See page 165.
- **Contextual Information Sidebar:** Show context information about the current source editor file. See page 164.
- **Scope list:** Lists the functions or similar constructs in a file. See page 167.
- **Architectures:** Defines named regions and views of the project. See Chapter 7.
- **Graphical Views:** Shows connections and structures of entities. See Chapter 10.
- **Reports:** Generate reports about entities. See Chapter 8.
- **Metrics:** Generate statistics about entities. See page 226.

# Entity Filter

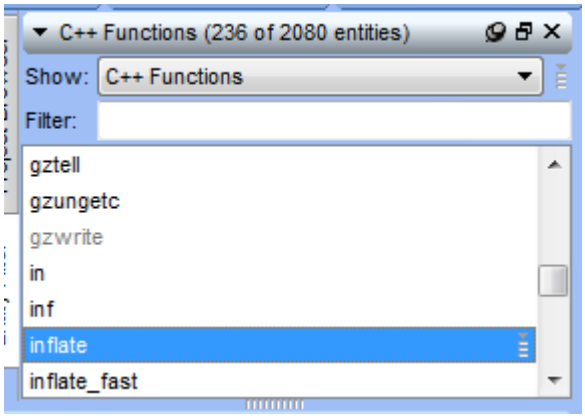
The *Entity Filter* provides a quick list of the selected entity type. You can filter this list to match a text string.

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




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

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




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

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

### Using the Filter Field

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

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

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

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

- **Fixed String:** This is the default behavior.
- **WildCard:** With this option selected, you can use * (any characters) and ? (any single character) wildcards for pattern matching. See page 157 for examples.
- **Regular Expression:** With this option selected, you can use Unix-style regular expressions. See page 157 for an overview.

To see only unknown or unresolved entities, click the drop-down icon and choose **Filter Unresolved Entities > Hide Resolved Entities**. To see only resolved entities, click the drop-down icon and choose **Filter Unresolved Entities > Hide Unresolved Entities**.

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

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

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 drop-down icon and choose **Entity Name as > Short Name.**

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

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

### Root Filters

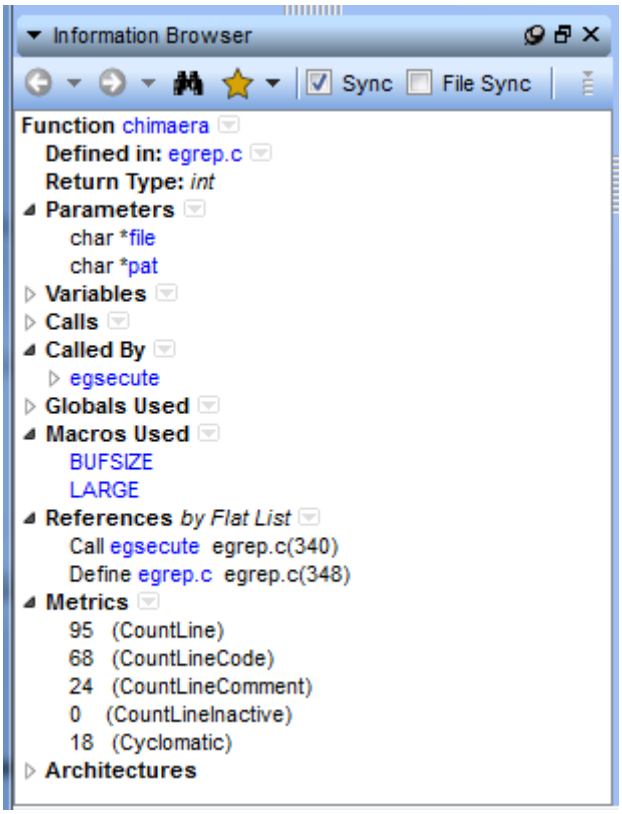
Notice that there are the filter type names that contain “Root”, as in **Root Calls, Root Callbys**, and **Root IncludeBys**. These “Root” types show only the top of a given tree. The tops (or bottoms) of relationship trees are often helpful points to begin exploring code that is new to you.

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

## Information Browser

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

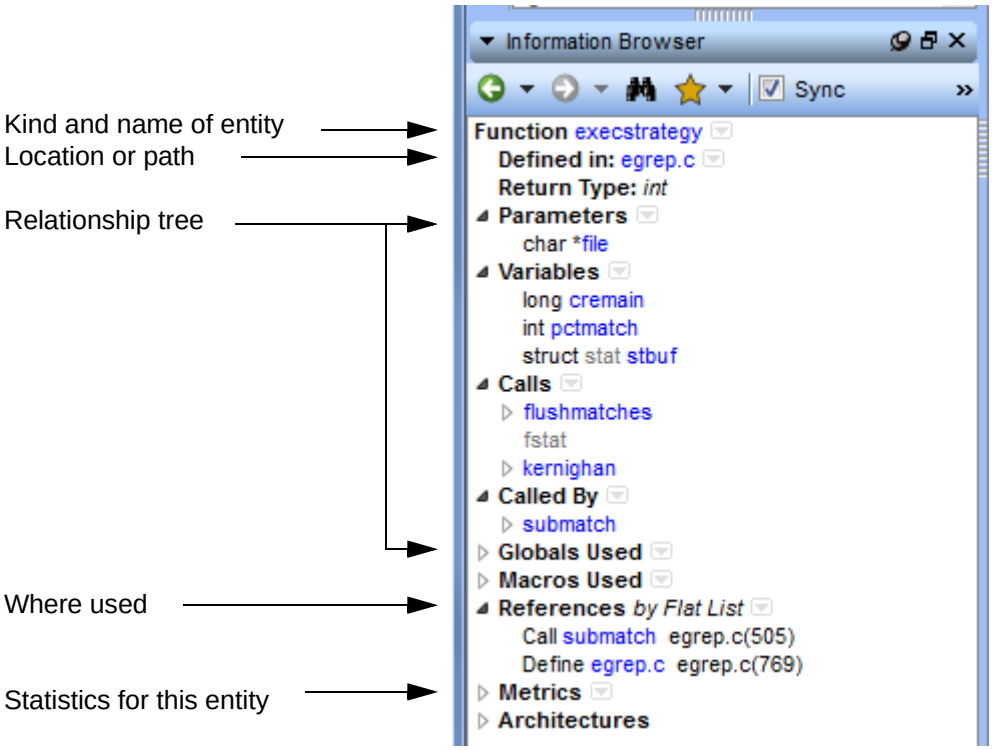
If the Information Browser isn’t open, you can open it by either clicking on an item in the Entity Filter or Project Browser. You can also right-click on an item anywhere and choose **View Information**. Or, choose **View > Information Browser** from the menus.



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

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

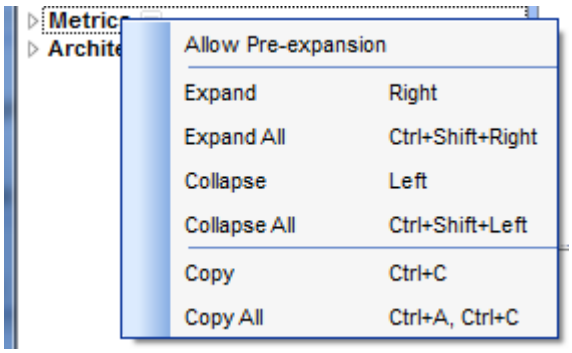
As you drill down you can change which entity you are learning about. Each time you change the entity, it is remembered in the Information Browser history for quick backtracking.




## Drilling Down a Relationship

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

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




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

See *Saving and Printing Information Browser Text* on page 136 for details on the other options in this context menu.

### Displaying More or Less Information

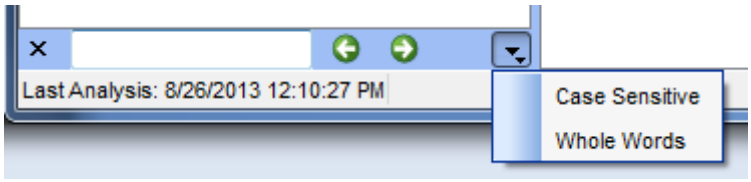
If you click the icon next to a bold heading such as **Calls**, **Called By** or **References** in the Information Browser (or right-click on the heading), you'll see options that let you modify how that entity is listed. These options include:

- **Fullname:** If checked, the fully-qualified name of the entity is shown.
- **Parameter:** Lists the parameters.
- **Reference:** Choose "Full" to include the file and line location of the reference.
- **Return Type:** Lists the return type.
- **Sort:** Controls the sort order of the list.
- **Type:** If checked, the datatype is shown.
- **Filename:** Controls whether the reference format is short, long, or relative to the project database.
- **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.
- **Group by:** For C++ classes, you can choose whether to sort class members by the type of access available (public or private) or the kind of member (function or object).

### Searching the Information Browser

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

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




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

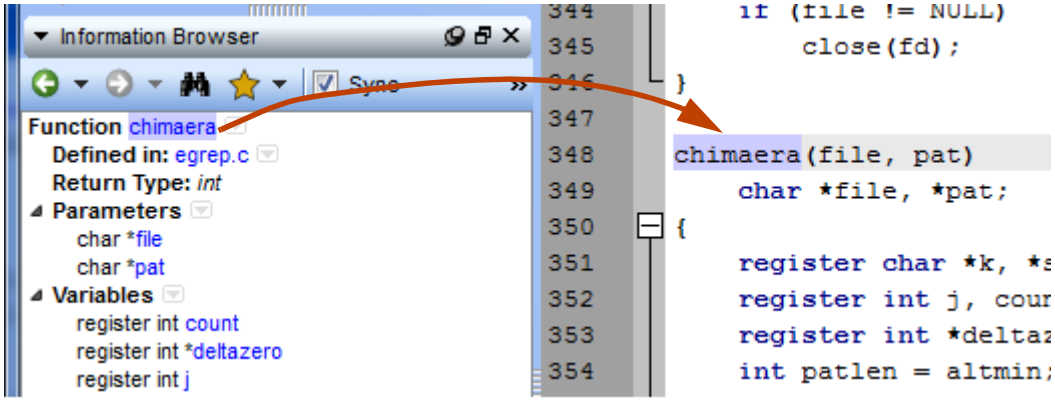
## Syncing the Information Browser

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

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


## Visiting Source Code

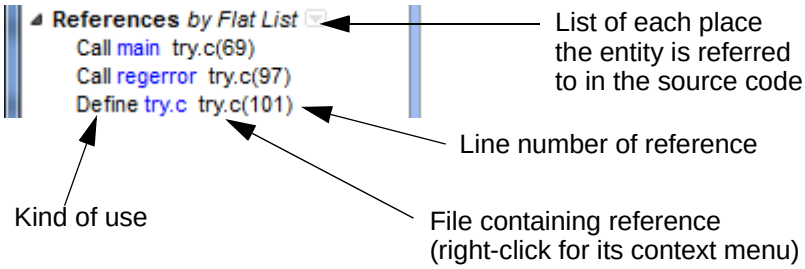
In general, if you double-click on an entity in an informational window (*Information Browser* or *Entity Filter*) the declaration of that entity will be loaded into the *Document Area*.




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

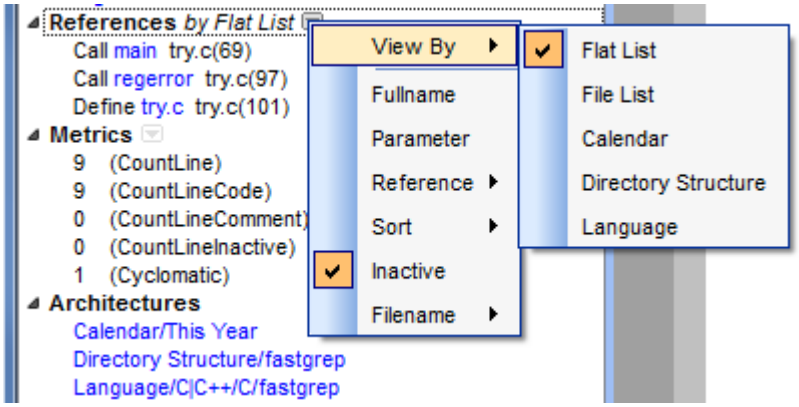
## Visiting References

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




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

Right-click on the “References” title for the node or 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.

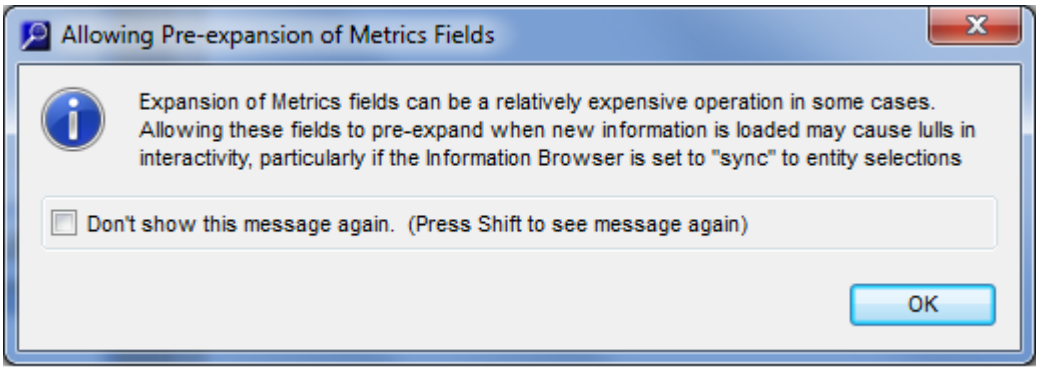



## Viewing Metrics

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

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

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




See *Metrics Reports on page 226* for details on metrics.

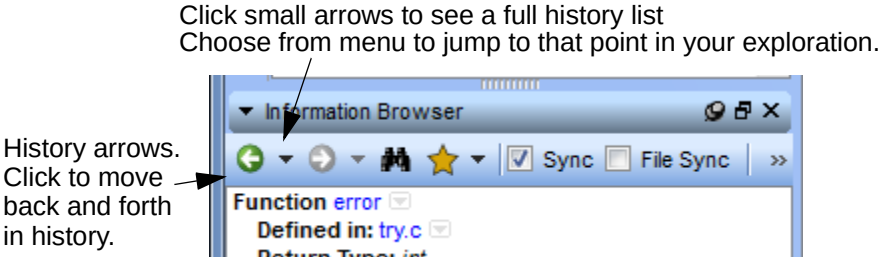
## Saving and Printing Information Browser Text

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 is has displayed. The *Information Browser* history can be found in the upper-left corner:

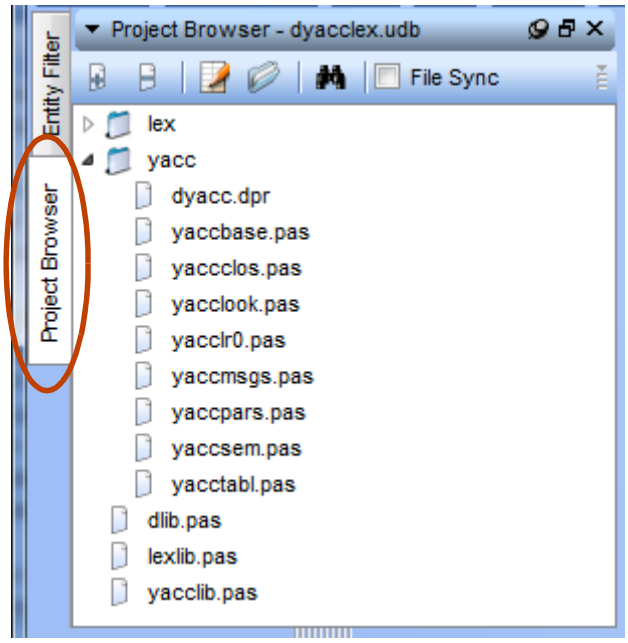



Use the right and left arrows to move back and forward in the history list. The down-arrows show the whole list.

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. The options with icons are also available in the toolbar for the Project Browser.

Add a file to the project

Remove selected file from project

Open file in Source Editor

Open file with default OS tool

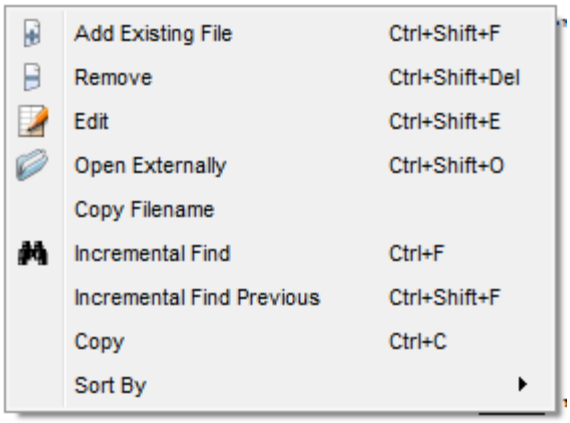
Copy filename to clipboard

Search for text in Project Browser

Search in reverse direction

Copy selected file to clipboard

Change sort order of files




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

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

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

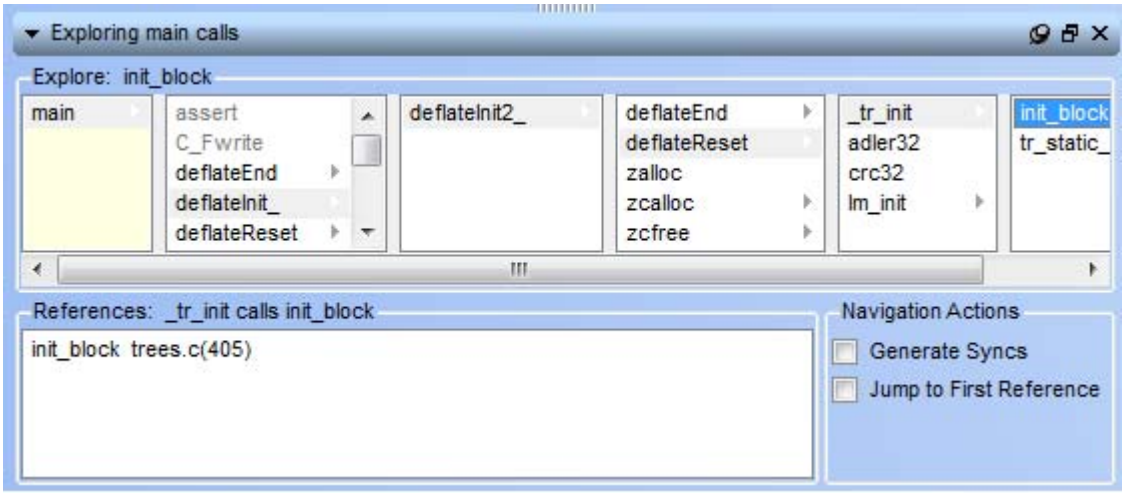
The **Open Externally** command opens an operating system dependent tool for the directory or file. For example, on Windows it opens a directory using the Windows Explorer. For a file it opens the default tool for the file extension.

The **Incremental Find** command opens a Find bar at the bottom of the Information Browser. Type text in the box and click an arrow to find an occurrence of the string in the Project Browser text. All text is searched, including files in folders that are currently closed. If you type a string that does not appear anywhere in the Project Browser, the field turns red. See page 133 for details on search options.

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

# 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.

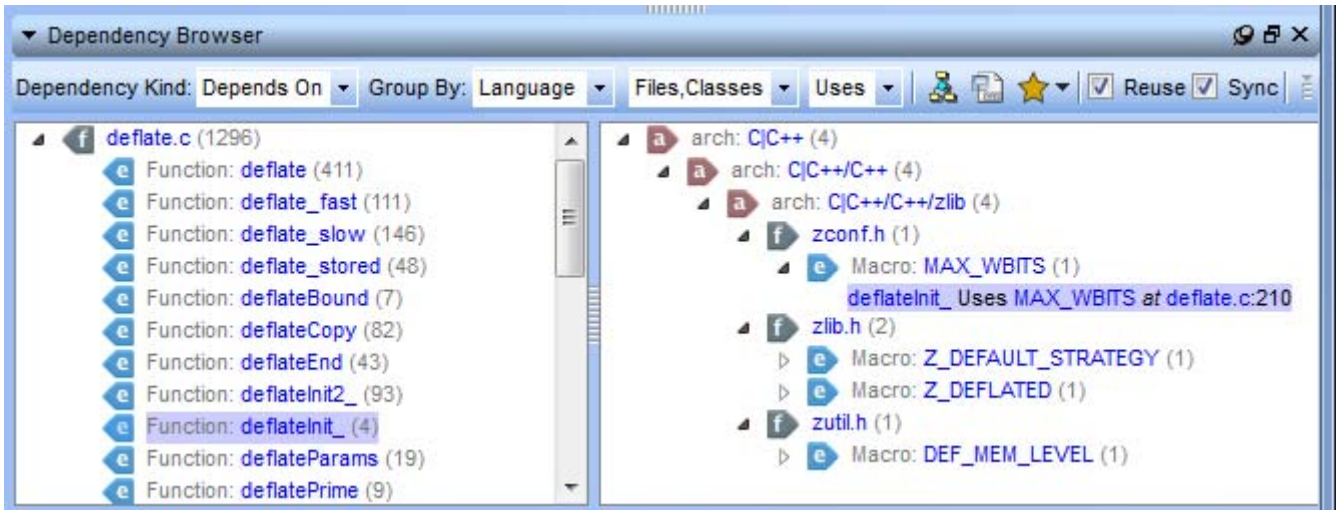
If you check the **Jump to First Reference** box, then the Source Editor automatically displays the initial reference to any entity you select in the Exploring window. Holding the Ctrl key down temporarily activates this behavior.

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

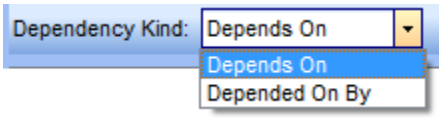
- **Show Long Name:** Shows the long name of each call.
- **Show Parameters:** Shows the parameter list for each call.

# Dependency Browser

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




To open the Dependency Browser, right-click on an architecture node, filename, class name, or package name anywhere in *Understand*, for example in the Entity Filter, Information Browser, or a graphical view. Choose **View Dependencies** from the context menu. Or, click the **View Dependencies** button in a dependency graph. You can also open the Dependency Browser by choosing **View > Dependency Browser**.

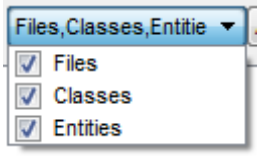



The left panel shows the item you selected and the items it contains. The right panel shows items that either depend on the item selected in the left panel or are dependent on that item, depending on your choice in the **Dependency Kind** field. For example, an item depends on another if it includes,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

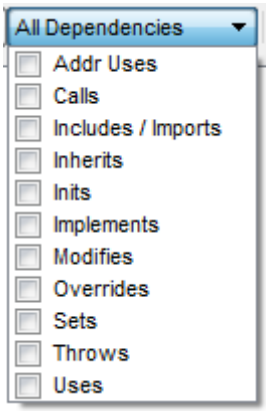
You can expand hierarchies in the left and right panels. For example, when you view dependencies for an architecture node, you can expand it to see lower-level architecture nodes, then files, then the entities in the files. Letters next to items identify whether they are architecture nodes (“a”), files (“f”), classes (“c”) or entities in files (“e”). You can also right-click on either panel and choose **Expand All** or **Collapse All**.

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

In the **Files, Classes, Entities** drop-down list, you can select whether to show each of these types of items in the right panel. If multiple items are checked, files are listed below the lowest-level architecture node that applies, classes are listed below files, and entities are listed below classes.




In the **All Dependencies** drop-down list, you can select types of dependencies to show. By default, all types are shown.




If you have the **Sync** box checked in the Dependency Browser and click on a relationship (connecting line or “edge”) in a dependency graph, the graph’s **Show** and **Group By** settings are copied to the Dependency Browser and the **Dependency Kind** is changed to Depends On.

Click the **Graph** icon to create a graphical view of the dependencies currently shown in the Dependency Browser. While the Dependency Browser shows one level of dependency, graphical views can show multiple levels. See page 196 for details.

Click the **Export** icon to export a comma-separated values (CSV) report of dependencies for the top item in the left panel of the Dependency Browser. You can also use the mouse to select items in the right panel of the Dependency Browser. Then right-click on a letter icon and choose **Copy** to place the currently selected text on your clipboard for pasting into other applications. See page 242 for other ways to export information about dependencies.

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

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

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

If you have a dependency graph open and the **Sync** box is checked, the Dependency Browser syncs to show any relationship you select in the dependency graph, and the two nodes connected by the relationship are highlighted in the Dependency Browser. In addition, **Show** and **Group By** settings from the dependency graph are synced with the Dependency Browser and the **Dependency Kind** is changed to Depends On.

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

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

- **Use Include/Import References.** By default, “includes” and “imports” are treated as dependencies. You may want to omit such relationships if they are required for building but are not logically dependent. This option can also be controlled in the Dependency category of the **Tools > Options** dialog (page 108).
- **Allow Pre-Expansion.** If you enable this option, nodes in the Dependency Browser are automatically shown as expanded. This option is on by default, and can also be controlled in the Dependency category of the **Tools > Options** dialog.

# Favorites

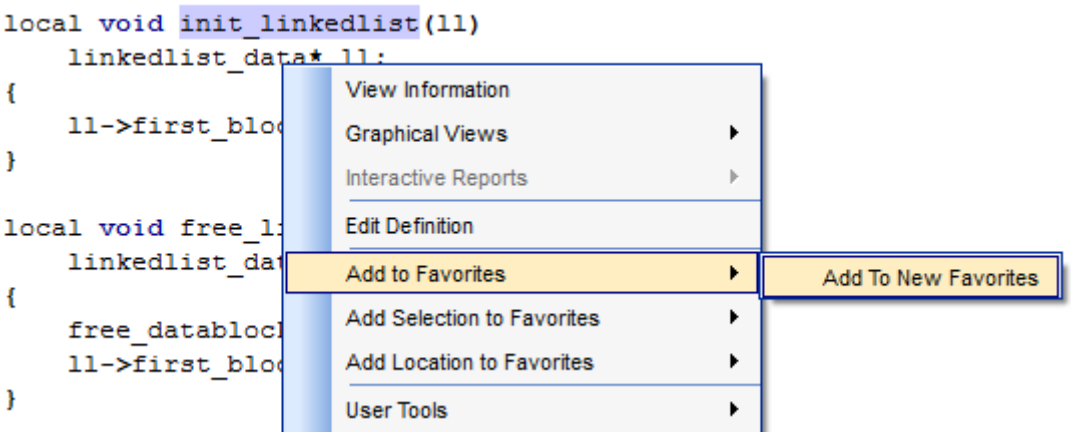
You can mark all kinds of things in Understand as “Favorites” so that you can quickly access them as you would web pages in a browser’s Favorites group. Your favorites can include entities, code locations, graphs, Information Browser displays, and dependencies. You can also store multiple plain text 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 184.

## Creating a Favorite Entity

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

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

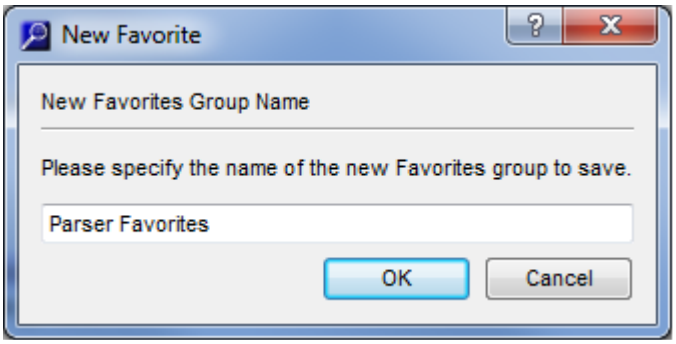



2 Choose **Add to Favorites > Add To New Favorites** (or an existing favorites group). This adds a link to the entity itself, even if the line number changes later. If you’ve already created a favorites group, you can select it from the submenu instead of using **Add To New Favorites**.

3 Alternately, if you right-click on a source file, you can choose **Add Location to Favorites** to add the line number in the file to a favorites group.

See *Creating a Plain Text Favorite* on page 145 for information about the **Add Selection to Favorites** command, which stores text for pasting from the clipboard.

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




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

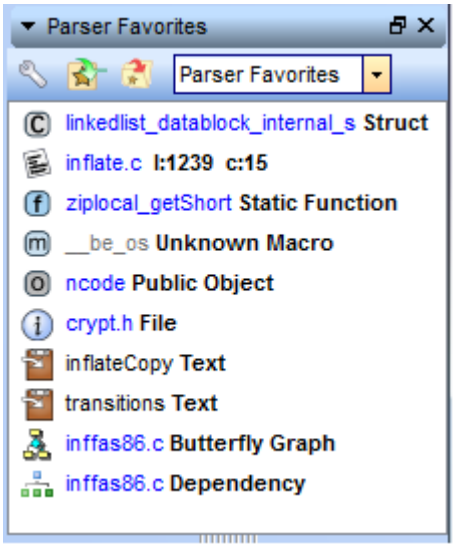
### Creating a Favorite View

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

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

### Using a Favorites Group

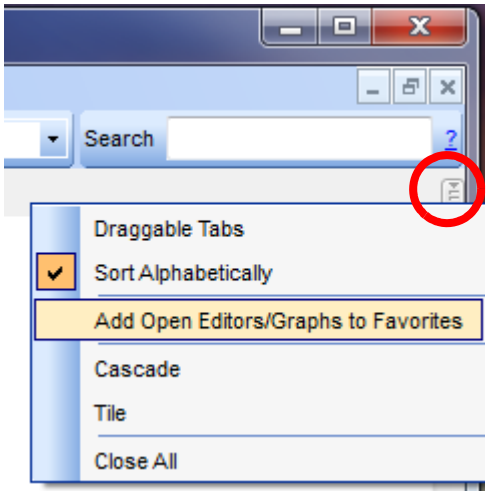
To open a Favorites group, choose **View > Favorites** and choose a group name from the menus. You see the favorites saved to that group.




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

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

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




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

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

Favorites with a file icon link to a file.

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

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

Favorites with a graph icon link to a graphical view.

Favorites with a clipboard icon store text that you can paste into source code. See *Creating a Plain Text Favorite* on page 145 to store text as a favorite. When you click on a text favorite, the text is placed in your clipboard and you can paste it into Source Editor windows or other applications.

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

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

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

Click this icon to create a text favorite. See *Creating a Plain Text Favorite* on page 145 for details.

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

Rename the current favorites group.

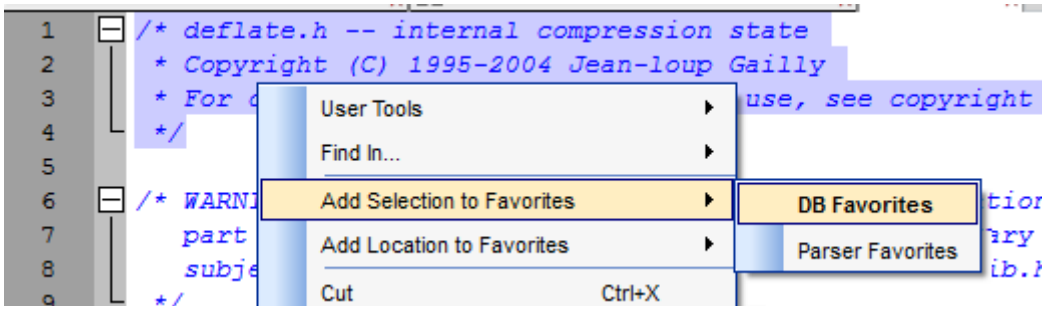
Delete the current group of favorites.

## Creating a Plain Text Favorite

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

To save text as a favorite, follow these steps:

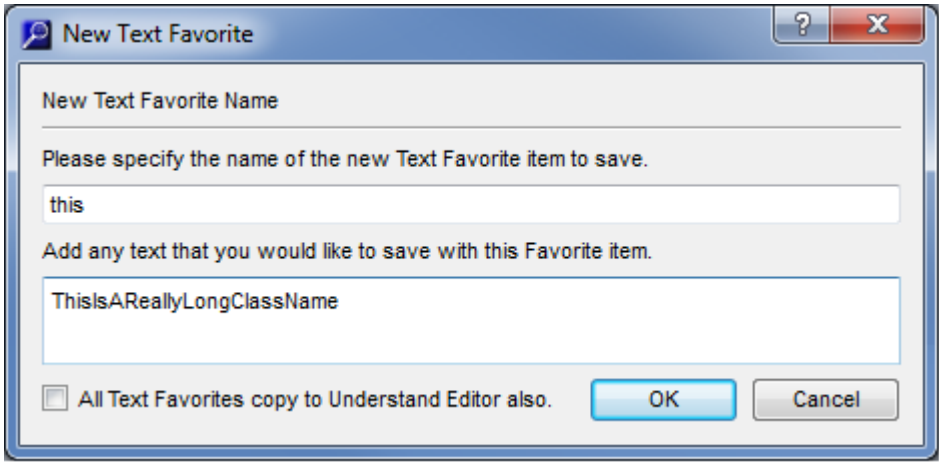
**1** Select text in a Source Editor and right-click.




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

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

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




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

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

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

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

# Chapter 5 Searching Your Source

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

This chapter contains the following sections:

# Searching: An Overview

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

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

| Search Command | See | Comments |
|---|---|---|
| Entity Locator | page 155 | project-wide, entity-based |
| Find in Files | page 150 | project-wide, text-based |
| Replace in Files | page 153 | project-wide text-based |
| Instant Search | page 148 | project-wide text-based |
| Find Next and Previous | page 163 | single file |
| Find & Replace | page 163 | single file |
| History | page 161 | project-wide |
| Bracket Matching | page 178 | single file |
| Favorites | page 142 | project-wide |
| Contextual Information Sidebar | page 164 | single file |
| Bookmarks | page 181 | project-wide |

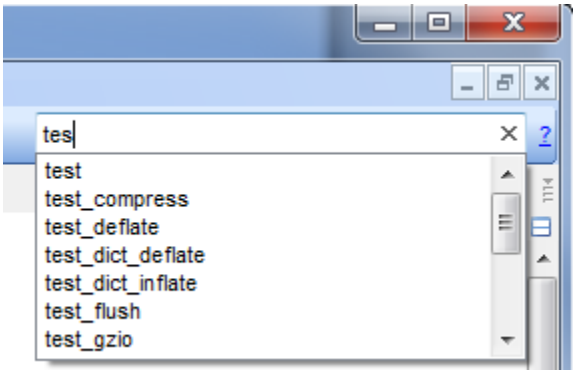
Each of these searching methods has advantages and disadvantages. Together they provide powerful ways to easily find what you need to find to better understand and modify your code.

See page 128 for a more complete list of the code exploration tools in *Understand*.

# Instant Search

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

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

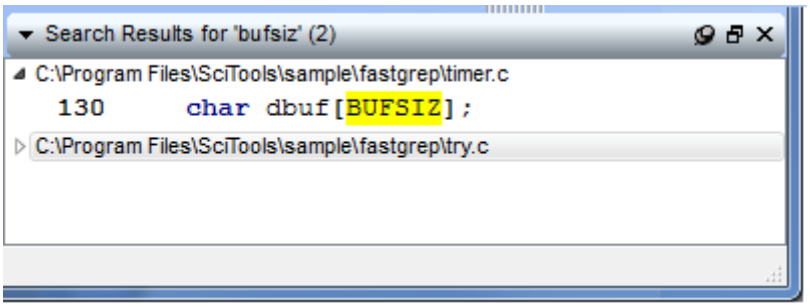



If you don't see this field, choose **View > Toolbars > Search** from the menus.

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

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

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




A number of more powerful search options are supported with Instant Search. The syntax used by this field is based on the syntax used by Apache Lucene, an open-source text search engine library. See lucene.apache.org/core/3_6_2/queryparsersyntax.html for syntax details.

The following list explains some of the syntax options available:

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

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

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

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

  ```
  string:test
  ```

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

  ```
  identifier:test
  ```

  The following search finds “test” only in comments:

  ```
  comment:test
  ```

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

  ```
  +delta for
  ```

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

  ```
  delta delta0 NOT delta2
  delta delta0 -delta2
  ```

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

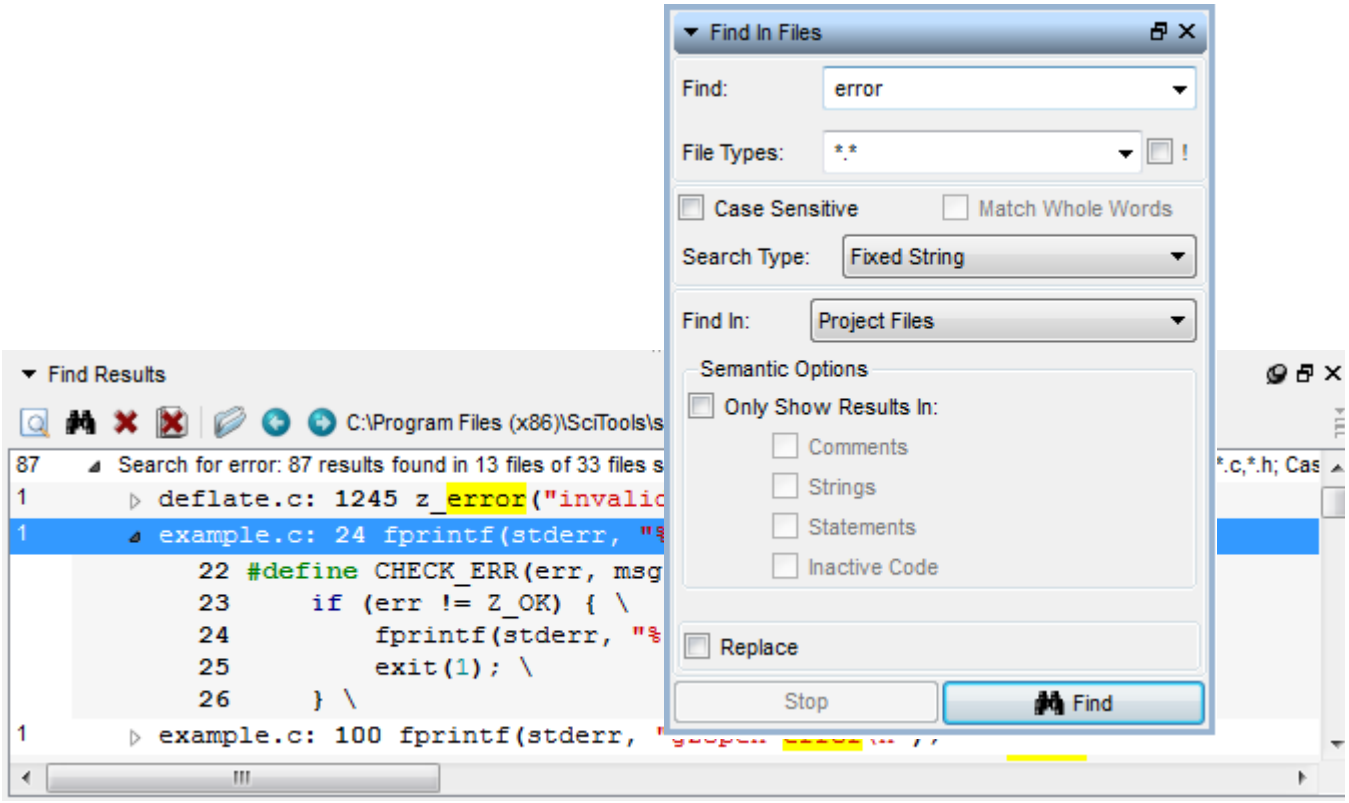
  ```
  (delta0 OR delta1) AND change
  ```

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

# Find in Files

You may search all project files or another selection of files for the occurrence of a text string or regular expression. Matches are shown in the *Find Results* window and can be visited in the source code by double-clicking on any line in the results. You can switch between the Find in Files and Replace in Files (see page 153) dialogs by checking the **Replace** box.

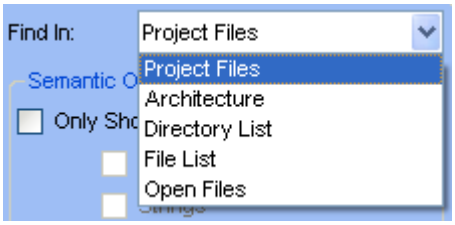
To open the Find in Files tool, choose **Search > Find in Files** from the menu bar, choose **Find in...** from any context menu, or press F5.




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

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

- **Case Sensitive:** Check this box for case-sensitive searching. The default is to ignore case.
- **Match Whole Words:** Check this box to match whole words only in regular expressions (“test” matches “test” but not “testing”). For fixed string and wildcard searches, word boundaries are ignored.
- **Search Type:** Choose whether to use Fixed String, Wildcard, or Regular Expression matching. See page 158 for details.
- **Find In:** Choose whether to search project files (either all files or just the open files), files in architecture nodes you select, files in directories you select, or files you select.

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

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

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

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

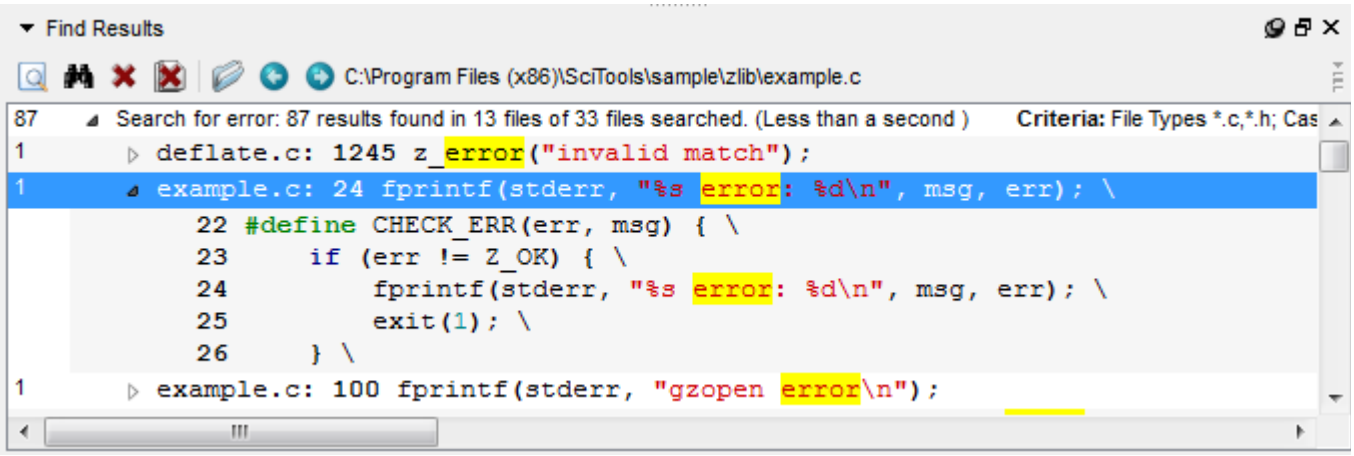
  If you select **Open Files**, all files that are currently open are searched. The **File Types** specification can be used to limit which open files are searched.

  When you right-click on an entity in source code or elsewhere, the **Find In** command lets you choose one of these options for the selected text string. The Find and Find In fields are filled in for you automatically.
- **Semantic Options:** If you choose to Find In “Project Files” or “Architecture”, you can check the **Only Show Results In** box to be able to control which matches are reported. Then you can check any combination of the **Comments**, **Strings**, **Statements**, and **Inactive Code** boxes to include those types of lines in the results. You must check at least one of these boxes if you check the **Only Show Results In** box.
- **Replace:** Switch to the Replace in Files (see page 153) dialog by checking this box.

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

## Find Results

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




You can view the source code for a match by double-clicking on a result. This opens the Source Editor and highlights the match. See *User Interface > Windows Category* on page 101 for ways to customize the Find Results display.

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

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

- Go to Find in Files dialog.
- Search within the currently selected set of results (using the same search bar described in *Searching the Information Browser* on page 133).
- Delete the current set of results.
- Delete all the results.
- Open the selected match in the Source Editor.
- Move to the previous or next match.
- Check the **Criteria** box to show the settings used to perform the search.

The drop-down icon in the upper-right corner of the Find Results area provides the following commands:

- By default, only the names of files are shown. To show full file paths, select “Long Names” from the **Display Files As** drop-down.
- Use the **Organize Results By** drop-down to change the organization of the most recent results. The choices are a flat list (the default), a file-based list, and hierarchies using the architectures.
- Choose **Show Criteria** to have the line that shows the number of results also list the search options that were used for each search.

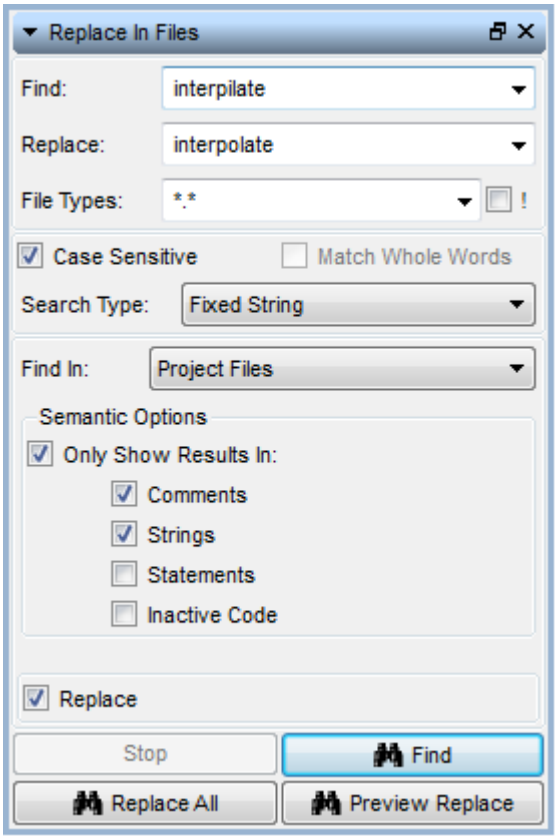
- Choose **Expand All by Default** to automatically expand all lines when new results are added.
- Choose **Clear Results Before Search** to automatically clear the results of the previous search whenever you run a new search.

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

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

## Replace in Files

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




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

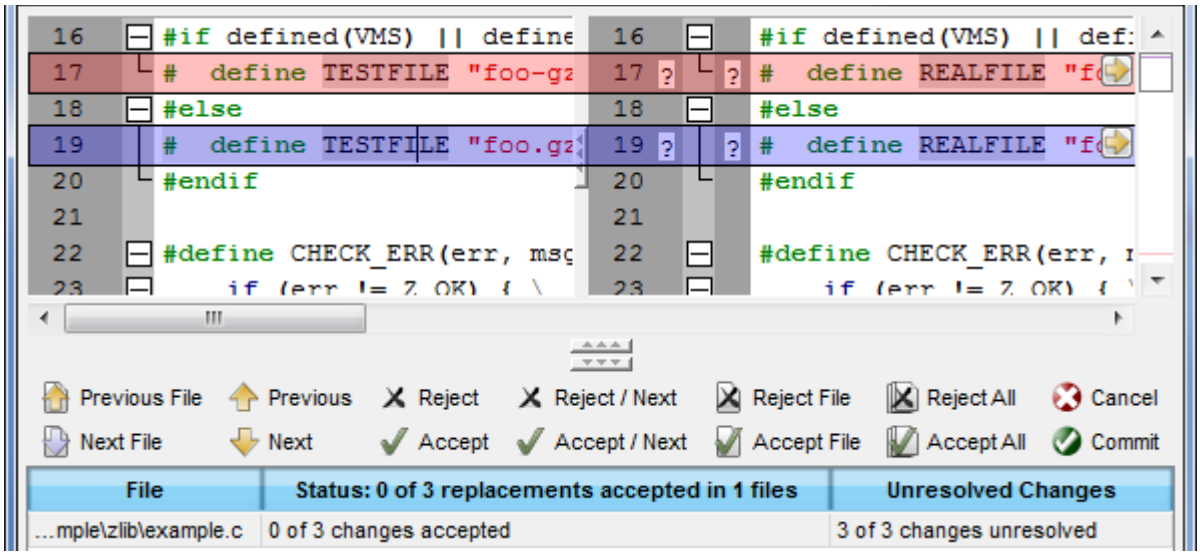
- There is a **Replace** field where you type the text you want to replace the matched string.
- In the **Search Type** field, you can select “Regular Expression - Multiline,” which lets your regular expressions match strings that cross line boundaries.

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

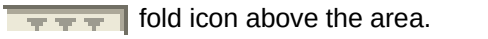
*Understand* checks for any unsaved source files. If there are unsaved files, you must click **Yes** to save all unsaved changes before making or previewing the changes.

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

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



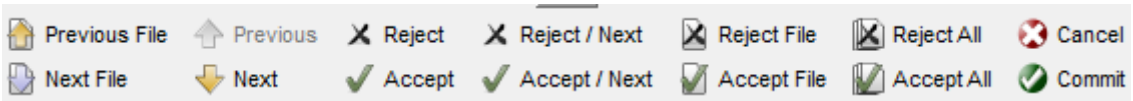

The top area shows the pre-change code on the left and the post-change code on the right. Replacements are in pink and the currently selected replacement is highlighted in blue. The left side has the **Hide Common Lines** option set so that most lines that will not be affected by the replacement are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

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

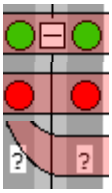
The lower area lists the files where replacements will be made and the number of replacements accepted and unresolved.

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

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




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



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

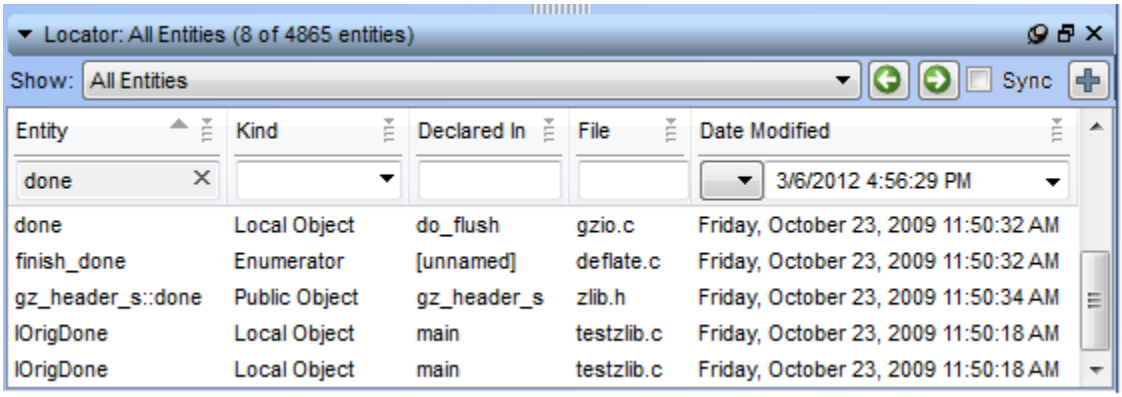
If you are sure you want to make all the changes, click **Accept All** and then click **Commit Changes** in the All Changes Resolved message window.

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

# Entity Locator

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

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




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

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.

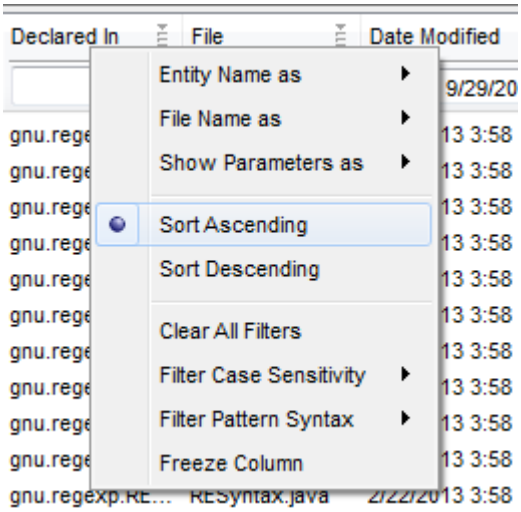
You can right-click on a column header and choose **Freeze Column** to move that column to the left and prevent it from being repositioned. By default, the File column is frozen and is the left column.

### Long versus Short Names

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

**Column Headers**

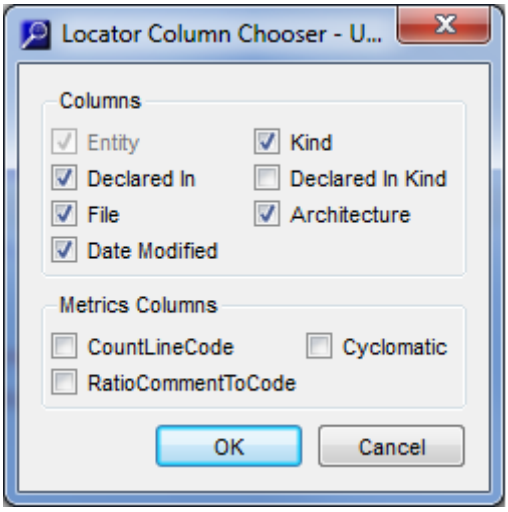
Column headers are tools in the *Entity Locator*. Left-click them to sort according to that column. Right-click a column or click the drop-down icon to see a menu that lets you control how entities are listed, sorted, and filtered.




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

**Choosing Columns**

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




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

## Filtering the List

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

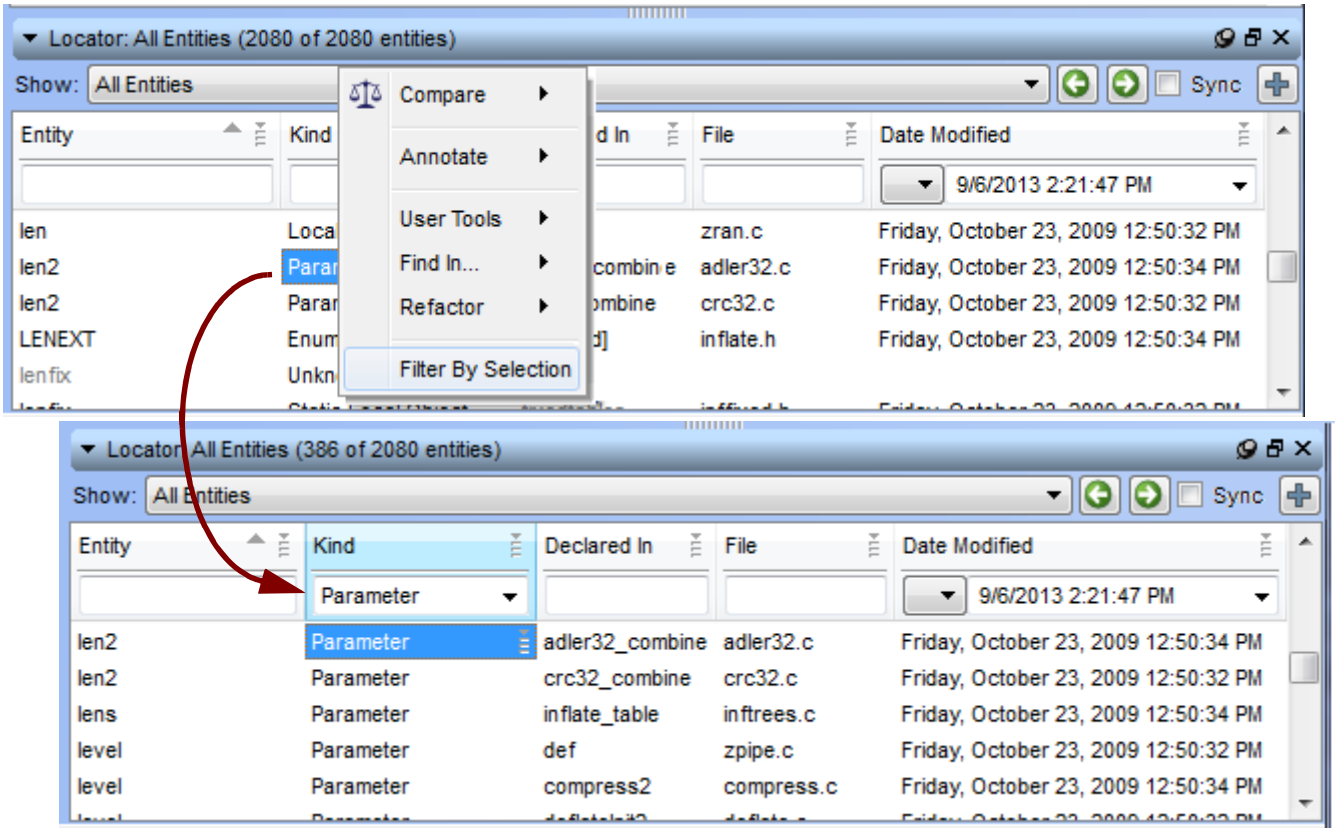
For example, you may filter by the *Kind* column selecting a kind from the drop-down list. You can also right-click on any item listed in the *Kind* column and select **Filter By Selection** from the menu. This filters the list of entities to contain only entities of the kind you selected. The title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

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

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

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

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




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

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

Right-click a column or click the drop-down icon to see the context menu for that column. You can choose for the filter case sensitivity to be **Case Sensitive** or **Case Insensitive** (the default). You can also 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.

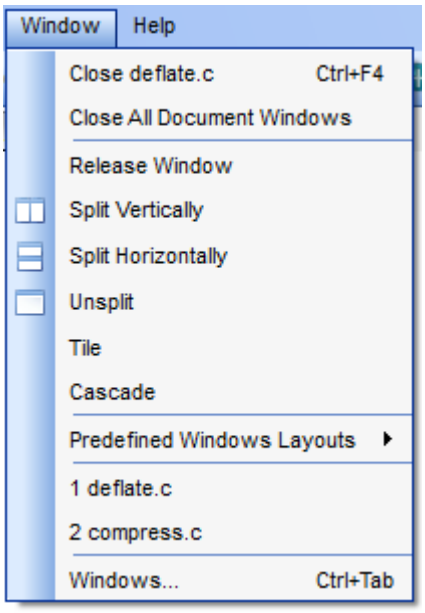
The following table lists some special characters used in regular expressions.

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

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

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

# Finding Windows




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

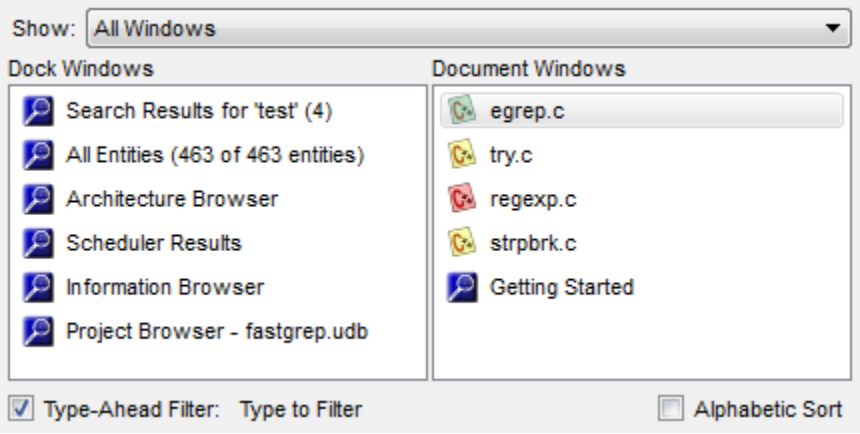
You can close the current document window by choosing **Window > Close <*current_window*>**. You can close all source files, graphical views, and other document windows by choosing **Window > Close All Document Windows**. If you have many windows open in the document area, you can right-click on the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

If you choose **Window > Release Window,** the tabbed area changes to a separate window that can be resized and moved around your screen. You can select **Capture Window** from the drop-down icon in the upper-right corner of a released window to replace the window within the main *Understand* window.

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

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

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




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

### Source Visiting History

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

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

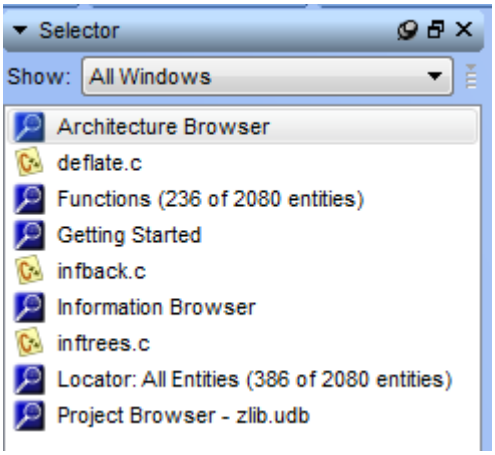
The source locations are stored as line numbers, not by entity name. If you want to save locations by entity rather than line number, see *Favorites* on page 142.

### 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. 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.

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




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

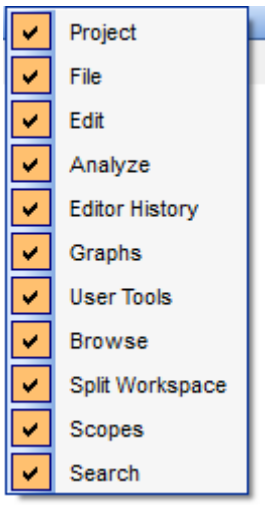
Using the Selector is a convenient way to perform actions—such as Close—on multiple windows by selecting multiple windows from the list, right-clicking, and choosing **Close Selected Window(s)** or **Close Unselected Window(s)**.

If you have created bookmarks in your source code (page 181), you can use the **Bookmarks** command in the View menu to open the list of bookmarks.

## Displaying Toolbars

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

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

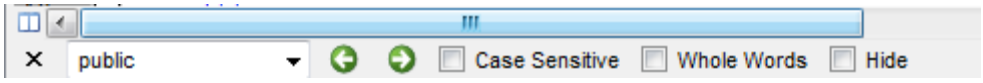


# Searching in a File

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

### Find Next and Previous

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




You can type a string in the field. As you type, matches for that string are highlighted in the Source Editor. The drop-down list contains recent search strings.

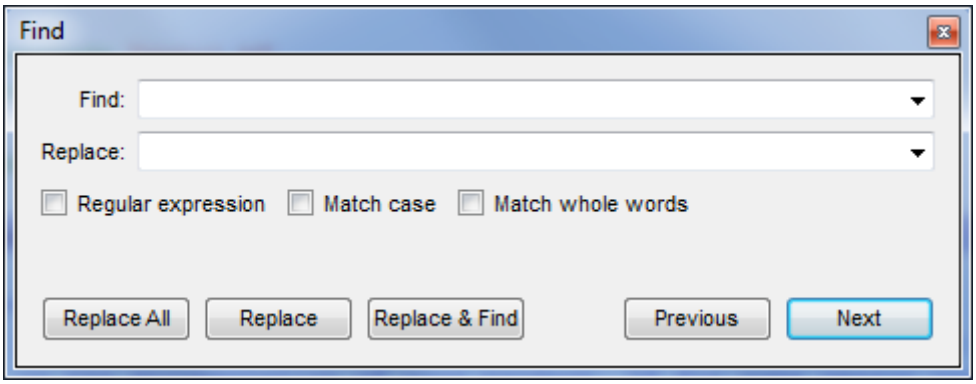
Click **Previous** or **Next** to move from match to match. You can check the **Case Sensitive** and **Whole Words** boxes to modify how the search is performed.

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

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

### Find & Replace

If you want to use Search-and-Replace or regular expressions for searching 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**, and **Match whole words** boxes to modify how the search is performed. If you check the **Regular expression** box, you can use Unix-style pattern matching. For a list of some of the capabilities of regular expressions, see page 157.

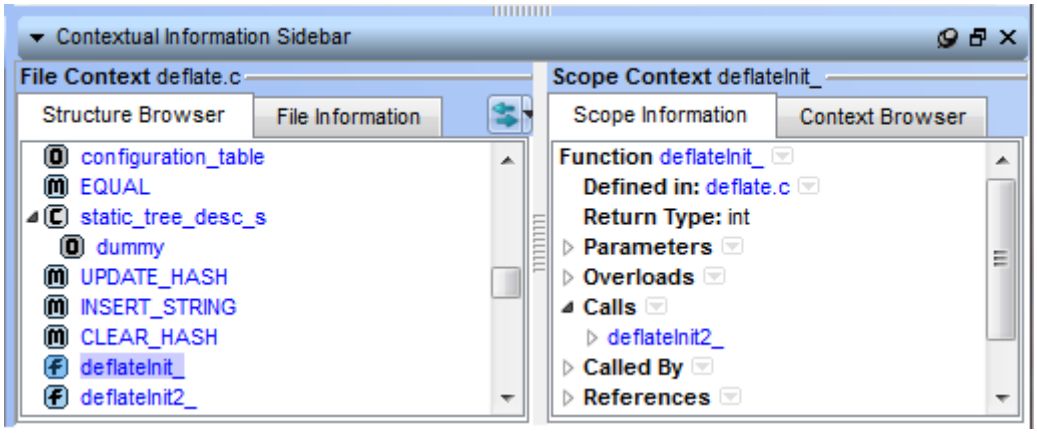
If you want to replace the string you are finding, type that in the **Replace** field.

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

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

## Contextual Information Sidebar

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




The CIS shows the structure and information for the currently active Source Editor. The tabs in the CIS provide the following information:

- **Structure Browser:** This is an expanded scope list for the current file. It lists the names of structures in the file. In addition to functions, it lists includes, macros, classes, and more. The icon next to the name indicates the type of entity. If you point your mouse cursor at an item, the hover text shows the entity type and name. Press Ctrl+F to search within this tab.
- **File Information:** This tab provides an Information Browser for the current file.
- **Scope Information:** This tab provides an Information Browser for the current entity—that is, the one highlighted in the Structure Browser tab.
- **Context Browser:** This tab shows the current entity's location in the hierarchy on the left and the entities it contains on the right.

The switch icon (Ctrl+,) to the right of the File Information tab changes the current file in the Source Editor and the CIS to a file in the same directory with the same name but a different file extension (the "companion file" if such a file exists). For example, the switch icon can toggle from a *.c or *.cpp file to a *.h file with the same name.

As always, right-clicking in any of these tabs provides links to more information about each entity.

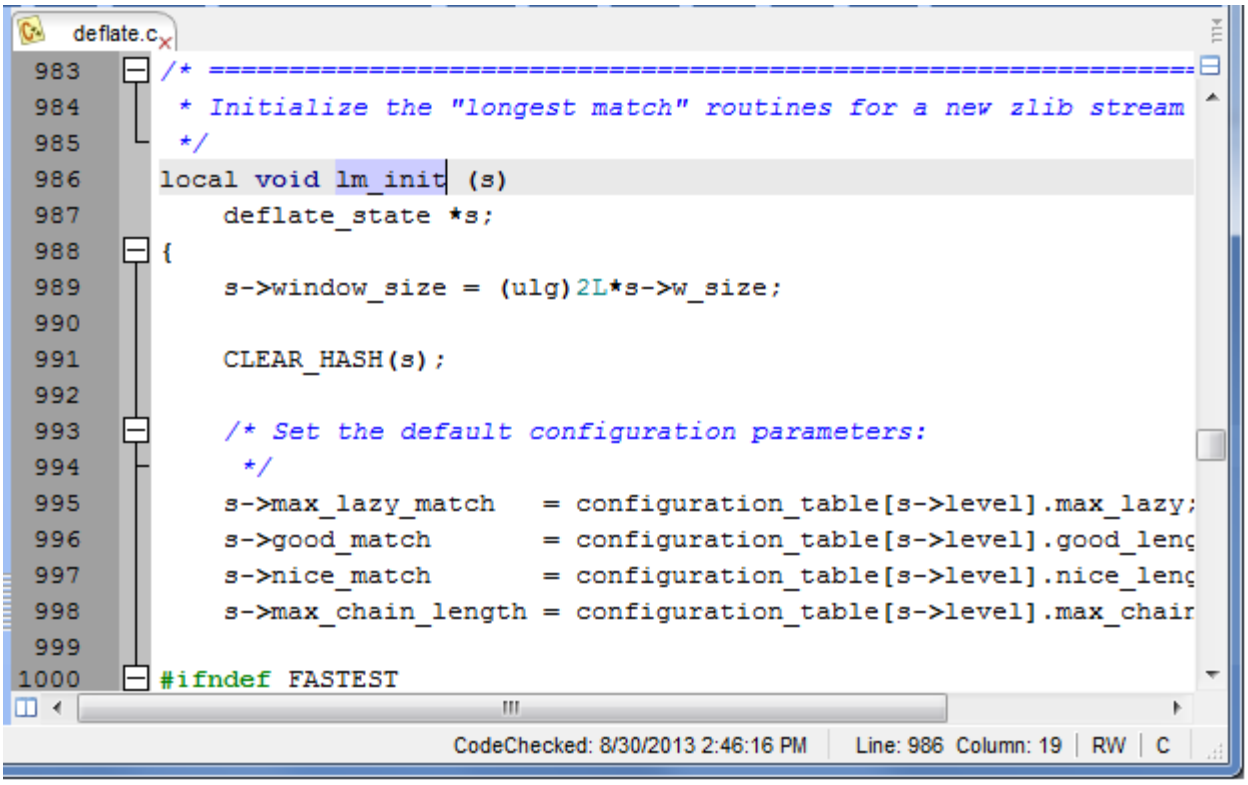
# Chapter 6 Editing Your Source

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

This chapter contains the following sections:

# Source Editor

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

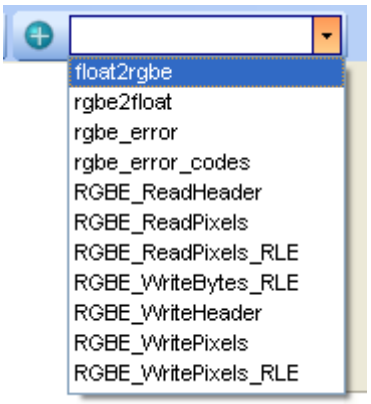



The line numbers and “fold” markings to expand/collapse blocks of code can be turned on and off in the **Editor** category of the Understand Options dialog you can open with the **Tools > Options** command (see page 109). The display font and a number of other items can also be changed in the **Editor** category. You can also enable bookmarks, indent guide marking, and a right margin marker (page guide) in that category of the dialog.

You can zoom in or out to make the text larger or smaller by choosing one of the **View > Zoom** menu options.

The **Editor > Styles** category of the Understand Options dialog (see page 115) lets you change the colors used for different types of source code. The **Key Bindings** category (see page 104) shows a list (and lets you modify the list) of keystrokes you can use in the Editor.

### Scope List

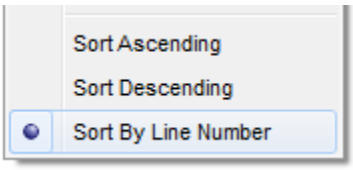



You can jump to a particular function, procedure, or other language-specific construct in the current source file by selecting from the scope drop-down list in the toolbar. The drop-down list shows all such constructs in the file the last time the project was analyzed.

You can click the **+** icon or choose **View > Scope List** to open the list in a Scope tab in the area where the Entity Filter is shown. This tab lists constructs in the current source file. This tab is useful for jumping around in large files.

The numbers next to each name in the Scope tab are the line numbers where each entity is declared in the file. Single-click on an item to view information about it in the Information Browser. Double-click on an item to jump to the location where that item is declared or created and to highlight all occurrences of that name in the current source file.

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 the *Contextual Information Sidebar* on page 164.

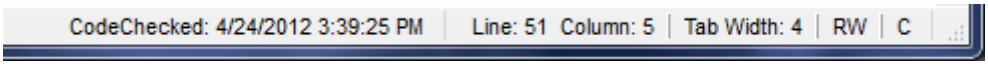
### Status Icons

Each file in a Source Editor has a status icon in its upper-left title bar. The letter in the icon indicates the type of file. The icon color indicates whether the file has been modified but not yet analyzed. An asterisk by the filename means the file has unsaved changes.

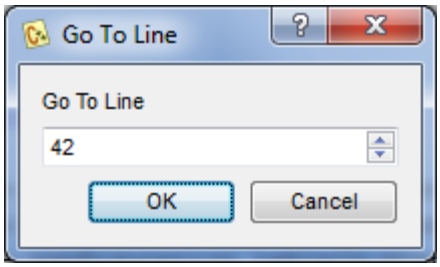
- Yellow icon = analyzed project file (has not been modified)
- Red icon = modified project file (needs to be analyzed)
- White icon = file not in the project

### Status Line

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



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




If you click the Tab Width, you can change the tab width for this file. See page 180.

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.

```
15  #define DO1(buf,i)  {adler += (buf)[i]; sum2 += adler;}
16  #define DO2(buf,i)  DO1(buf,i); DO1(buf,i+1);
17  #define DO4(buf,i)  DO2(buf,i); DO2(buf,i+2);
18  #define DO8(buf,i)  DO4(buf,i); DO4(buf,i+4);
19  #define DO16(buf)   DO8(buf,0); DO8(buf,8);
20
```

Once you select text, you can use the **Cut** and **Copy** commands in the **Edit** menu or the context menu. You may then paste the text into other applications as needed.

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

## Browse Mode

You can switch a Source Editor to “Browse” mode by clicking the **Browse** button in the main toolbar or choosing **View > Browse Mode** from the menus. When you are in Browse mode, the icon is highlighted.

When you are in Browse Mode, entities in the code act as links. An underline is shown when your mouse cursor moves to a link. Clicking a link moves you to the declaration of that entity and updates the Information Browser to show details about that entity.

If the declaration of an entity you click on is not found, a message is shown in the status bar and your computer beeps.

When you are in Browse Mode, you can still edit the file and the keyboard and right-click function the same as in regular mode. Only left-clicking the mouse is different.

You can temporarily enter Browse Mode by holding down the Ctrl key while using a Source Editor window. You can toggle Browse mode by pressing Ctrl+Alt+B.

See page 116 for settings to control the behavior of Browse Mode.

### Context Menu

The context menu in the Source Editor provides access to a number of exploration and editing features. Many of them let you find specific information about the entity you right-click on.

The following exploration features are typically included in the context menu (depending on where you click):

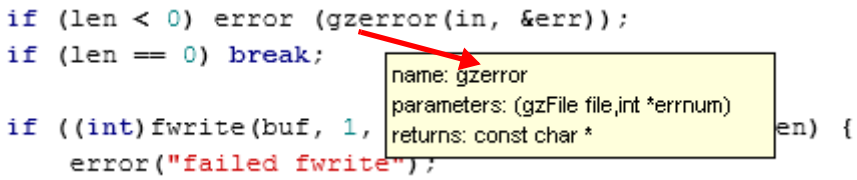
- View Information (see page 131)
- Graphical Views (see Chapter 10)
- Edit Source/Definition (see page 134)
- User Tools (see page 313)
- Explore (see page 139)
- Find in... (see page 150)
- Add Favorite (see page 142)
- Metrics Charts (see page 237)
- Metrics Browser (see page 233)

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

- Undo / Redo
- Cut / Copy / Paste (see page 168)
- Select All (see page 168)
- Jump to Matching Brace (see page 178)
- Select Block (see page 178)
- Hide/Show Inactive Lines (see page 178)
- Fold All (see page 178)
- Soft Wrap (see page 180)
- Comment Selection / Uncomment Selection (see page 179)
- Change Case (see page 179)
- Revert (see page 170)
- Add Bookmark (see page 181)

**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.




## Saving Source Code

If you have edited a source file, you can click , press Ctrl+S, or choose **File > Save** to save your changes.

You can choose **File > Save As** to save to a different file. If you save a project file to another filename, you will be asked whether you want to add the new file to the project.

If you have edited multiple source files, you can click or choose **File > Save All** to save changes to all modified files.

If you want to ignore changes you have made choose since the last save, right-click in a file and choose **Revert**.

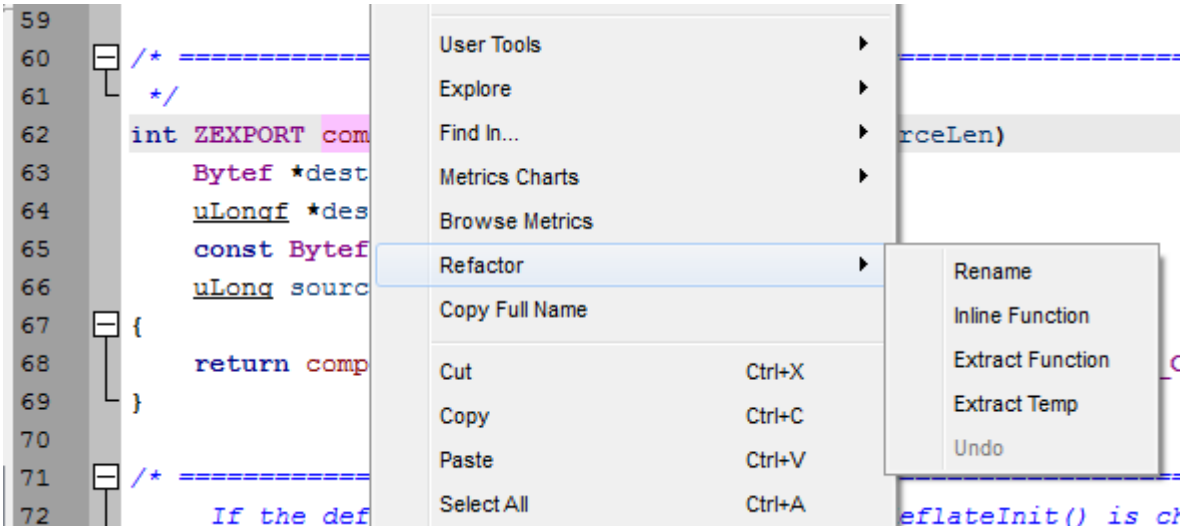
You can close the current source file by choosing **Window > Close <*current_file*>** from the menus. You can also middle-click on the tab above the source file area to close that tab (if your mouse has a middle button).

You can close all source files by choosing **Window > Close All Document Windows**. You can also right-click on the tab for the source file area and choose **Close**, **Close All**, **Close All But This**, or **Close All Tabs to the Right/Left**.

# Refactoring Tools

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

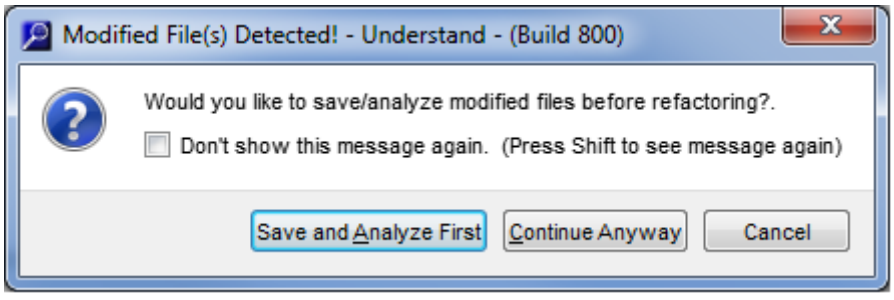
The refactoring tools allow you to preview the changes using a code comparison window. Refactoring changes can have significant effects on code, and should be reviewed before committing to make certain that the changes are correct.




The following refactoring tools are provided:

- **Rename:** page 172 (various languages)
- **Inline Function:** page 173 (C/C++)
- **Extract Function:** page 174 (C/C++)
- **Extract Temp:** page 176 (C/C++)

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



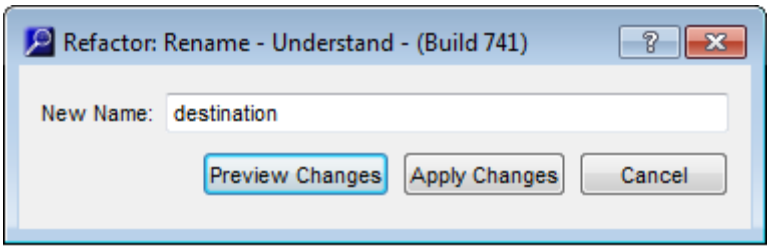

If you perform a **Refactor** operation and then decide that you did not want that change, right-click and choose **Refactor > Undo** from the context menu. In some cases, the refactoring operation cannot be undone because of subsequent changes.

## Renaming Entities

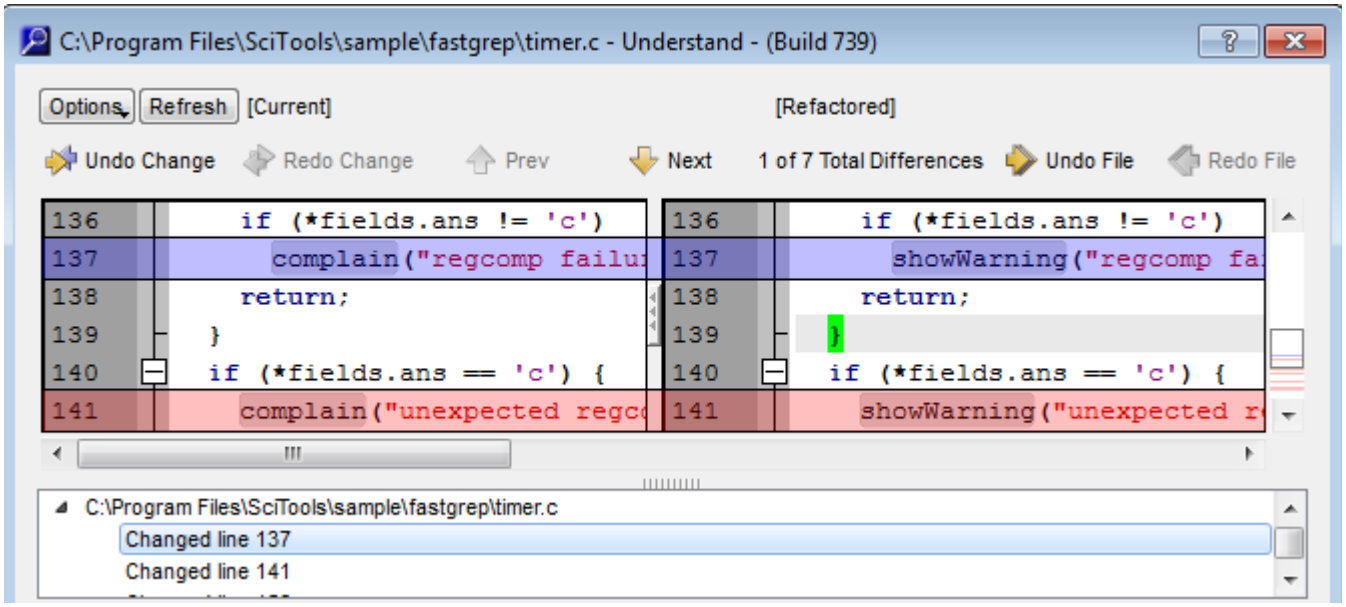
A command to rename entities is provided in order to support refactoring of your code in order to make your code more readable. It allows you to change the name of an entity throughout your project. The **Refactor > Rename** command is similar to **Replace in Files** (page 153). However, the difference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

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

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




4. A dialog opens that lets you examine all the instances where this entity name is used and how it will be changed. Code Comparison on page 301 describes the icons and drop-down menus in this dialog.




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

## Inlining Functions

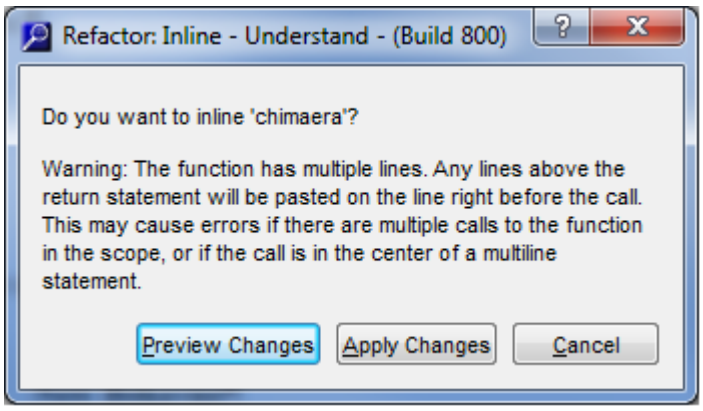
Inlining functions is a common optimization technique that places the code of a function at the location where it is called instead of in a separate function. In compiled languages, this can often be performed through compiler optimization, but inlining the source code may be useful for various reasons,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

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

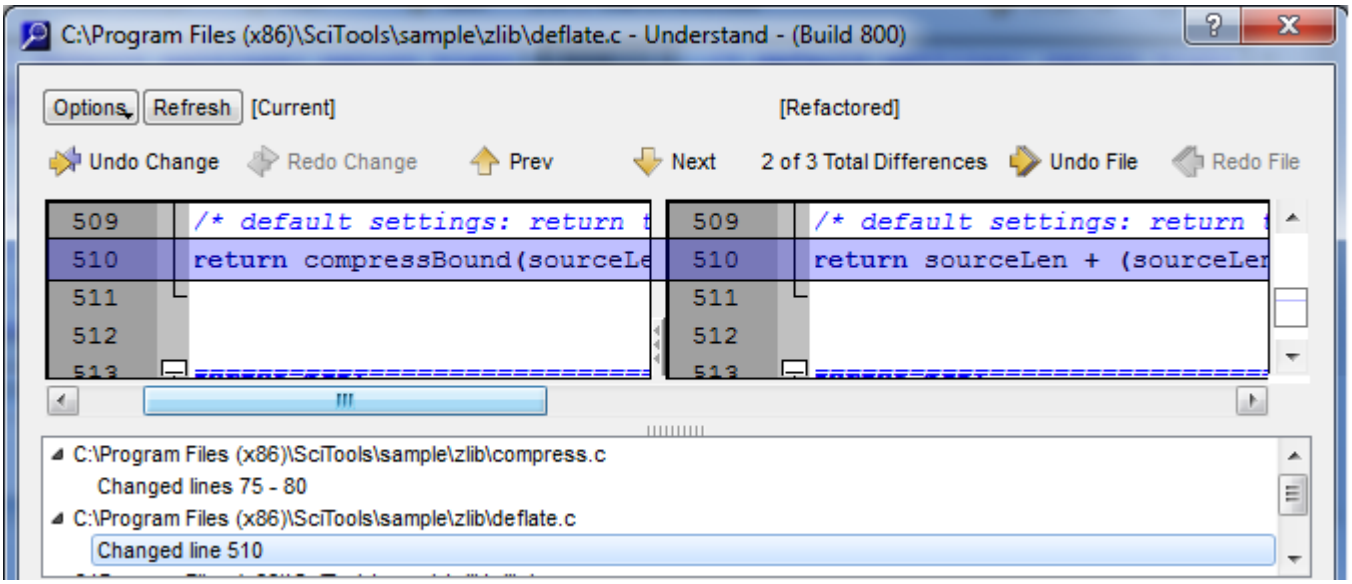
1 Highlight the name of the function you want to inline.

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

3 Click **Preview Changes**. If you are absolutely sure you want to perform the inlining operation, click **Apply Changes**.




4 A dialog opens that lets you examine all the changes that will occur. Code Comparison on page 301 describes the icons and drop-down menus in this dialog.



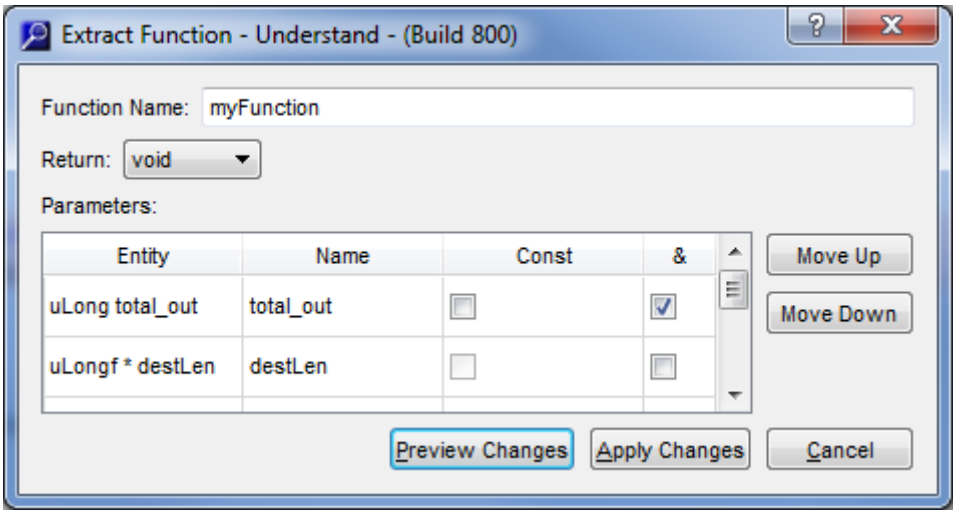

5 If you are sure you want to make all the changes, click **OK**.

## Extracting Functions

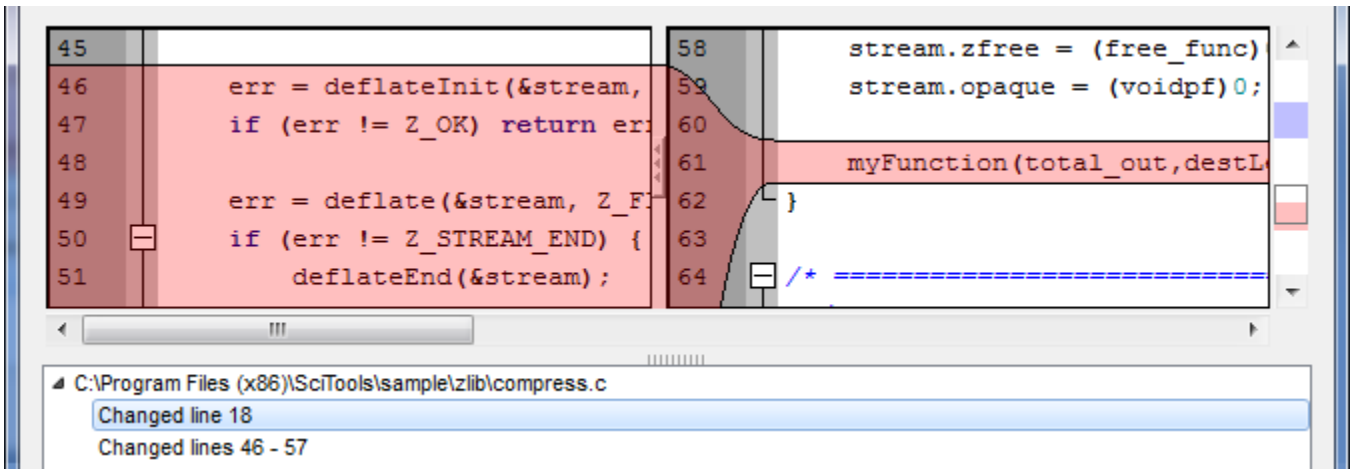
The opposite of function inlining is function extraction. You can extract some code to a separate function so that it can be called in several places and maintained in one place.

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

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




3. Type a name for the function to be extracted and called from this code location.
4. Select the value or variable to be returned by the function.
5. For the parameters to be passed to the function, you can use the Move Up and Move Down buttons to change the sequence and check the boxes to identify parameters to be passed as const values or by reference.
6. Click **Preview Changes**. If you are absolutely sure you want to perform the operation, click **Apply Changes**.
7. A dialog opens that lets you examine all the changes that will occur. Code Comparison on page 301 describes the icons and drop-down menus in this dialog.




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

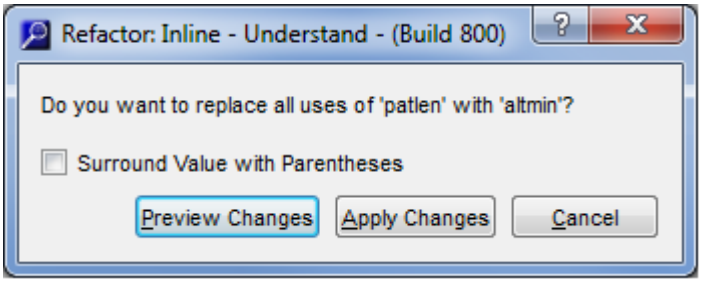
## Inline Temp

Inline temp refactoring can be used with a local or temporary variable that is initialized and never set after that. The inlining replaces used 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`:

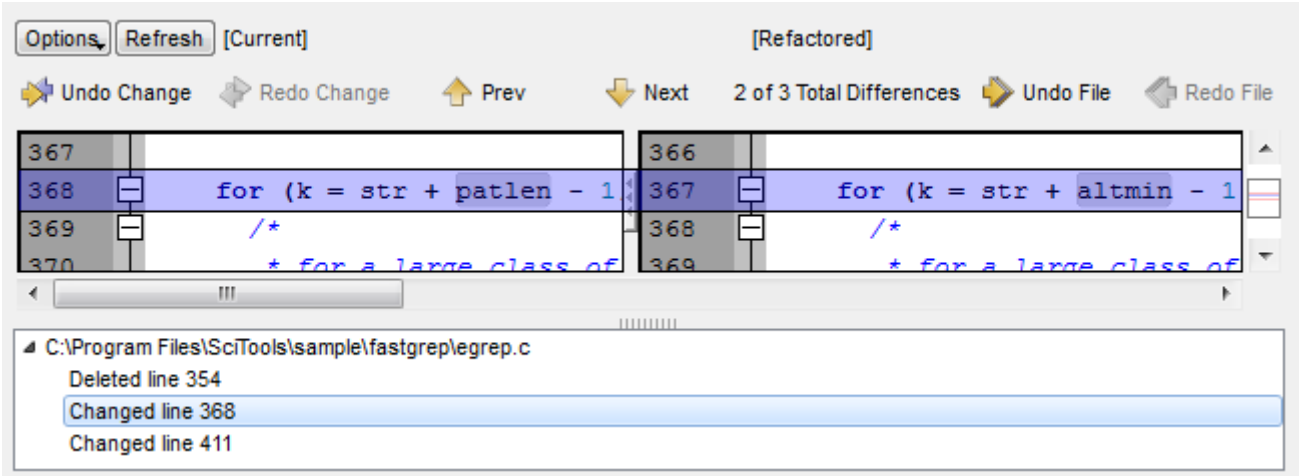
```
int patlen = altmin;

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 Code Comparison on page 301 describes the icons and drop-down menus in this dialog.



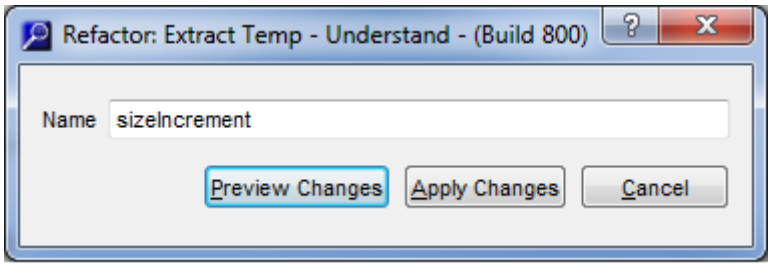

6. If you are sure you want to make all the changes, click **OK**.

### Extract Temp

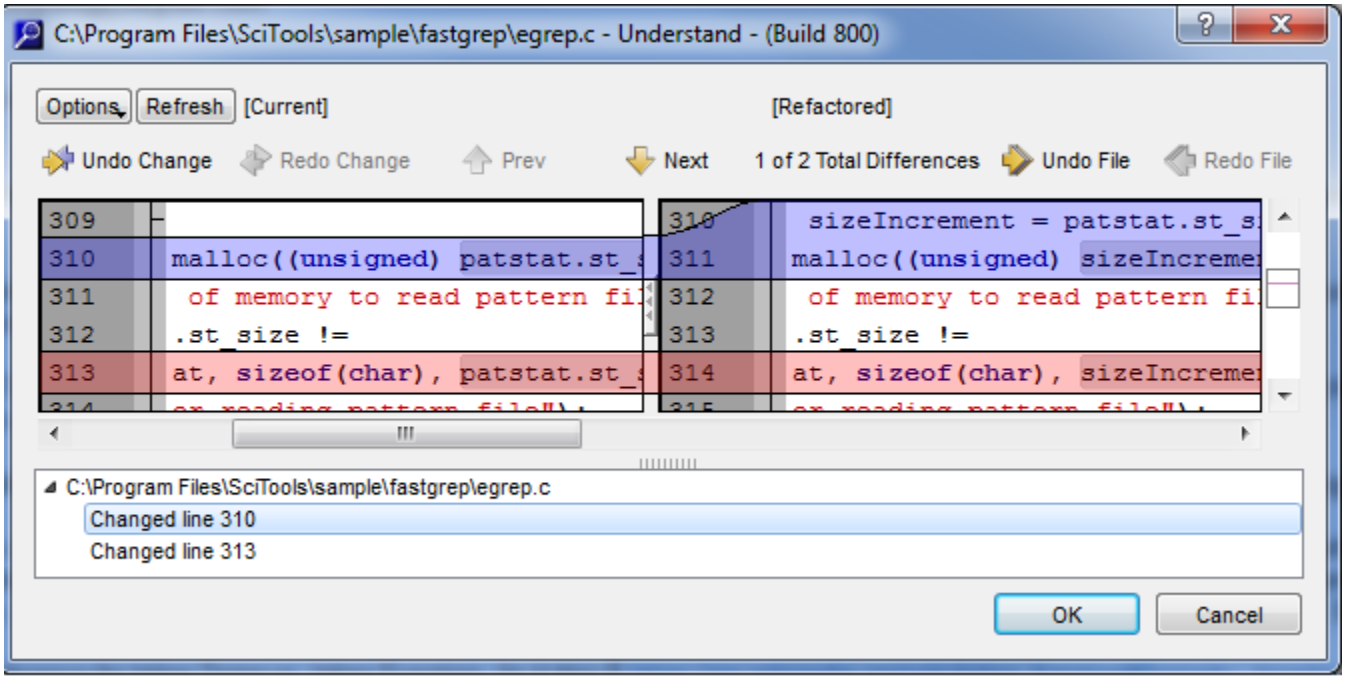
The opposite of inline temp is extract temp. If you have a complicated expression, you may want to assign a part of that expression to a local or temporary variable that can be reused within that function wherever the expression you select is used.

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

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




4. A dialog opens that lets you examine all the changes Code Comparison on page 301 describes the icons and drop-down menus in this dialog.




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

# Other Editing Features

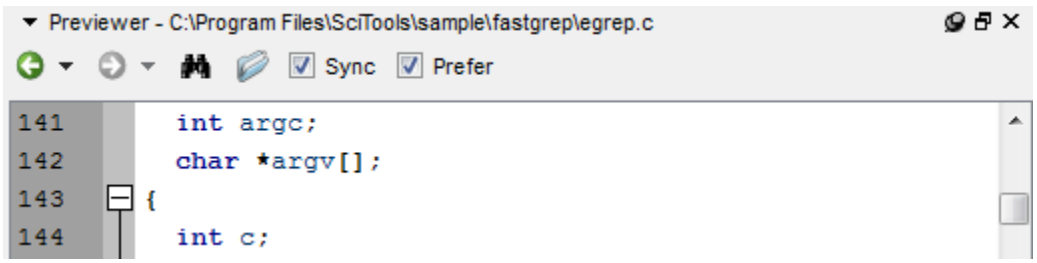
The Source Editor also provides several other options for displaying and editing files:



## Previewer

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

- You cannot edit the code in the Previewer window.
- **Sync checkbox;** If this box is checked, a single-click on an entity in another view displays the location where that entity is defined in the Previewer.
- **Prefer checkbox;** If this box is checked, a double-click on an entity in another view displays the location where that entity is defined in the Previewer. (Double-clicking on an entity in the Previewer always opens that entity's definition 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-click and **Jump to Matching Brace**) jumps the editor to the matching end or beginning brace. Ctrl+j isn't active unless your editing cursor is by a symbol that it can match. Another Ctrl+j takes you back where you started. You can also choose **Search > Go to Matching Brace** from the menus.

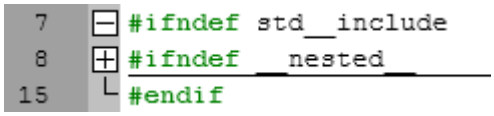
Pressing Ctrl+Shift+J (or right-click and **Select Block**) selects all the text from the bracket to its matching bracket.

Brackets without a match are highlighted in red when you move your cursor to them. Brackets with a match are highlighted in green.

When your cursor is on a preprocessor directive that has a match (for example, #ifdef and #endif), you can use Ctrl+j (or right-click and **Jump to Matching Directive**) to move your editing cursor to the match.

## Folding and Hiding

The - and + markings next to the line numbers allow you to "fold" the code to hide blocks such as functions, if statements, and other statements that have a beginning and end.




If you right-click on the code, you can choose **Fold All** to close all the open blocks. You can also fold and unfold source code by choosing **View > Fold All** from the menus.

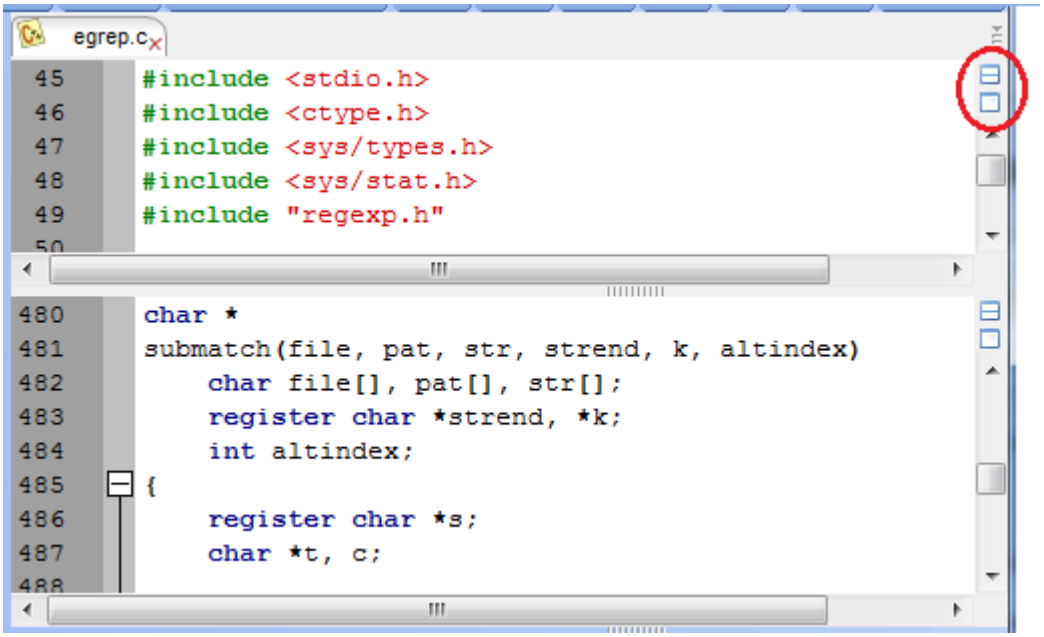
You can add explicit fold markers to code in languages where // is treated as the beginning of a comment. For example:

```
//{{
 /* code to hide when folded */
//}}
```

You can also choose **Hide Inactive Lines** to hide preprocessor lines that are not active because a preprocessor macro is not defined. Choose **Show Inactive Lines** to view all lines again. You can also toggle this setting by choosing **View > Hide Inactive Lines** from the menus.

### Splitting the Editor Window

You can click the **Split** icon (circled below) to divide the source editor into two or more separately scrollable panes. Click one of the **Join** icons to merge two panes.




### Commenting and Uncommenting

You can comment code that you have selected by right-clicking and choosing **Comment Selection**. To remove the comment characters, right-click and choose **Uncomment Selection**. You can do the same thing using the **Edit > Comment Selection** and **Edit > Uncomment Selection** commands in the menus.

Note that nested comments within the selection are not analyzed.

### Changing Case

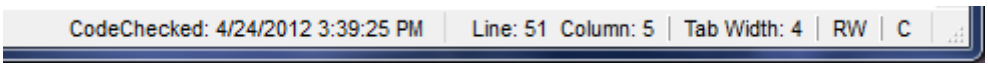
You can change the case of selected text in the Source Editor. Follow these steps:

1. Select a word or words in the source code.
2. Choose **Edit > Change Case** from the menus, or right-click and choose **Change Case** from the context menu.
3. Choose the type of case you want to apply to the selection. The choices are as follows:

| Choice | Default Keystroke | Original | Result |
|---|---|---|---|
| Lowercase | Ctrl+U | Test_me please | test_me please |
| Uppercase | Ctrl+Shift+U | Test_me please | TEST_ME PLEASE |
| Invert Case | Ctrl+Shift+I | Test_me please | tEST_ME PLEASE |
| Capitalize | Ctrl+Alt+U | Test_me pleaSe | Test_me PleaSe |

**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 109).




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 111).

**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**. You can also change the wrapping mode by choosing **View > Soft Wrap** from the menus. The wrapping is for display only; no actual line breaks are added to your source file.

See *Editor > Advanced Category* on page 111 to change the wrap mode for source code printing.

**Insert and Overtype Modes**

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

**Sorting Lines Alphabetically**

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

**Keyboard Commands**

To see a list of keystrokes that work in the Source Editor, choose **Tools > Options** and go to the **Key Bindings** category. For example, Ctrl+Alt+K cuts the text from the cursor position to the end of the line. And, Ctrl+T transposes the line at the cursor position with the line above it.

Another way to see a list of key bindings is to choose **Help > Key Bindings**. Search for the line that says “Editor” (around line 110) to get to the beginning of the keystrokes for the Source Editor windows.

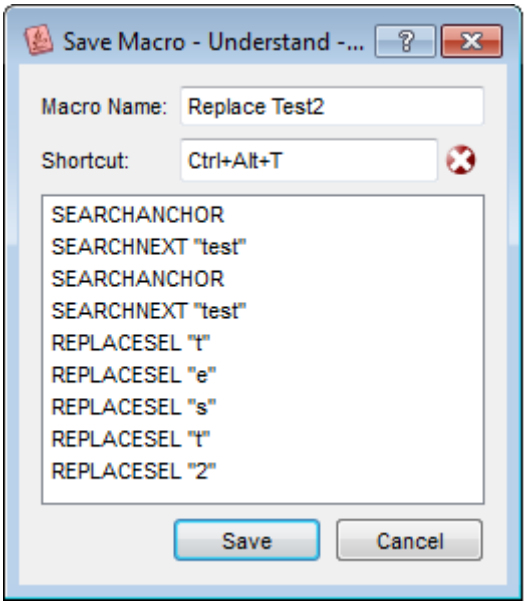
**Recording, Playing, and Saving Macros**

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 114 for details.

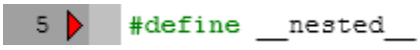
## Creating and Opening Files

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**. For example, the companion file of encrypt.c is encrypt.h.

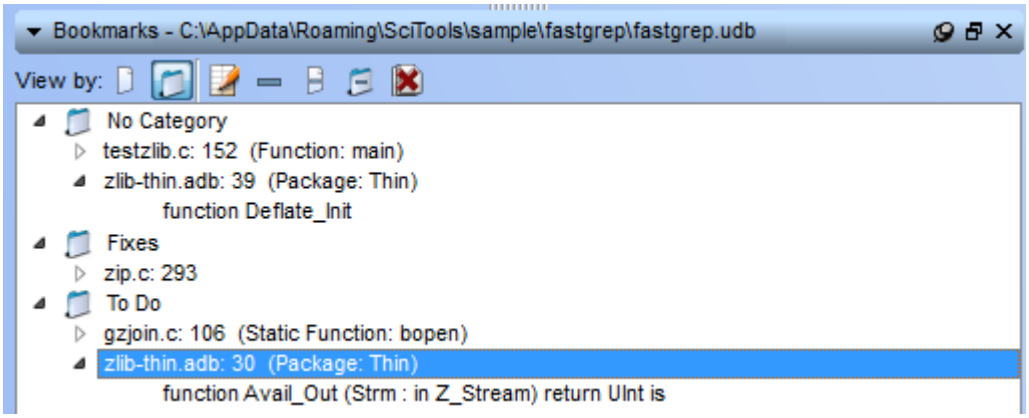
## Bookmarking

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.




In a file with multiple bookmarks, you can right-click and choose **Previous Bookmark** or **Next Bookmark** to quickly move between places in a file. These commands are also available under **Edit > Bookmarks** in the menus.

You can open a Bookmarks area to view a list of all your bookmarks in all your files by choosing **View > Bookmarks** from the menus.

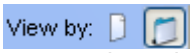



If you point to bookmarked code in the Bookmarks area, the 5 lines of code surrounding the bookmarked line are shown in the hover text.

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 the first line of code inside a function.

Bookmarks and Favorites (page 142) are stored as part of the project. If you want to mark code locations on a cross-project basis, see *Annotations* on page 184.

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



You can use the **View by** icons to switch between a file-based and a category-based view. The file-base view lets you expand filenames to see the bookmarks in that file. The category-based view lets you assign bookmarks to categories you create.


Select a bookmark and click this icon to change the category the bookmark is in. To create a new category, type the name and click **OK**. To use an existing category, select it from the list.


Select a bookmark and click this icon to delete that bookmark.


Select a bookmark and click this icon to mark it as a temporary bookmark to be deleted 24 hours after marking it as temporary.


Select a file in the file-based view and click this icon to delete all the bookmarks in this file. You can also select a bookmark and click this icon to delete all the bookmarks in the file that contains the selected bookmark.


Select a category in the category-based view and click this icon to delete all the bookmarks in the category. The category itself is not deleted.

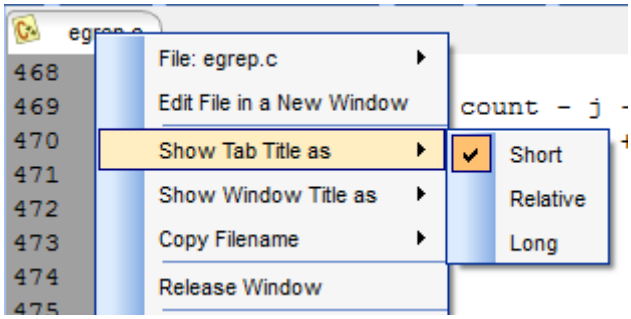

Click this icon to delete all your bookmarks.

The drop-down icon in the upper-right corner of the Bookmarks area provides the following commands:

- **Copy to Clipboard on Double Click:** By default, 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 to change a tab to a window within the *Understand* window.

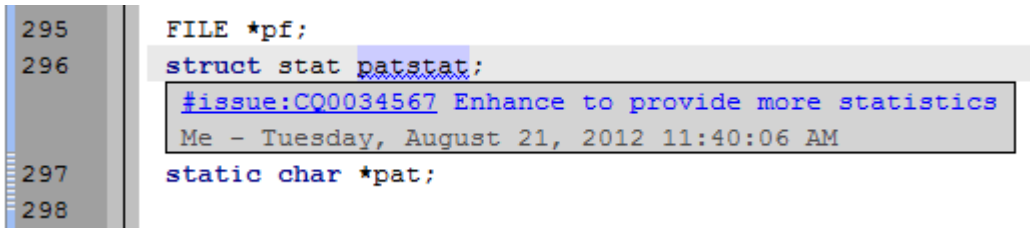
## Changing the Source Code Font Size

You can change the default display font and font size in the **Editor** category of the Options dialog that you open with the **Tools > Options** command (see page 109).

In addition, you can change the display size of the font for an individual source code window by choosing options from the **View > Zoom** submenu. **View > Zoom > Zoom In** makes the font size larger. **View > Zoom > Zoom Out** makes the font size smaller. **View > Zoom > Reset Zoom** changes the font size back to the default.

# Annotations

Annotations let you add comments or notes about entities without changing the source code directly. You can view the annotations inline, following the definition of the entity to which they are attached. They can also be seen in hover text wherever the annotated entity is used, including in graphs and in the Information Browser.

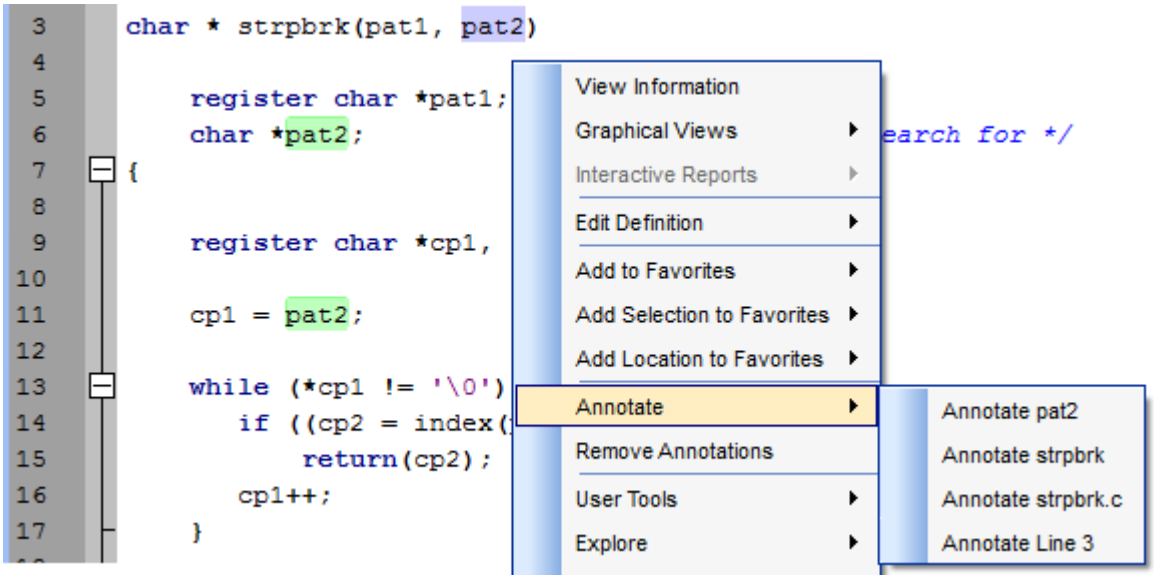



Each annotation can be “tagged” with a key value pair. Such tagging is useful for organizing your notes using keywords, author names, or any other identifier you want to use.

## Adding an Annotation

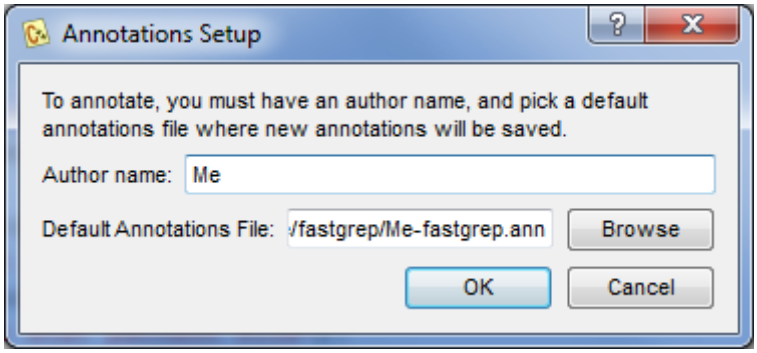
To add an annotation, follow these steps:

1. Highlight an entity, such as a variable or function name, anywhere in *Understand*. For example, you can select an entity in source code, in the Information Browser, or in the Entity Filter.
2. Right-click on the entity and choose **Annotate** and the entity or line number you want to annotate from the context menu. (Line numbers cannot be annotated in a file that has not been saved. Annotations to a line number are updated if possible when the line number changes.)

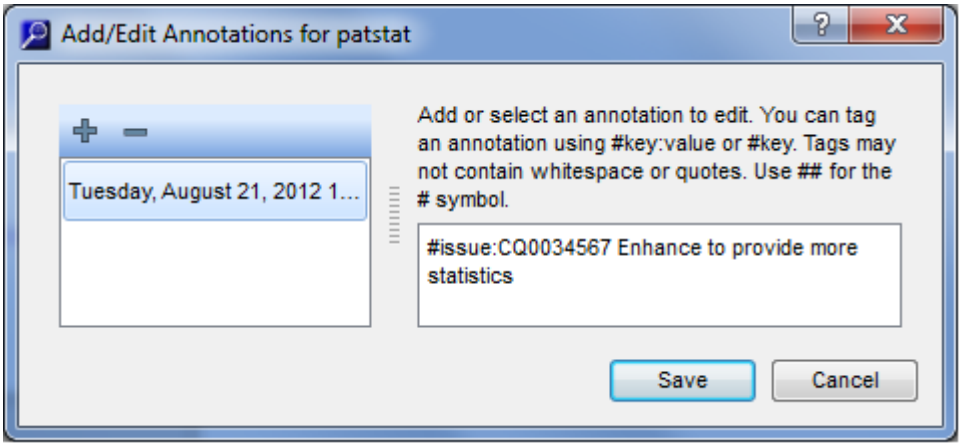



3. If this is the first time you are adding an annotation, you will see the Annotations Setup dialog. Type your name in the **Author name** field. The annotations are stored

in a *.ann file within the project. The **Default Annotations File** is stored in the project directory. The filename includes your name and the name of the project. (See page 186 to learn about managing annotation files.)




4 In the Add/Edit Annotations dialog, type your annotation comment in the right box. If you want to tag your comment so that it will be easy to search for, begin the text with a #key or #key:value tag. For example, you can use #reminder:CodeReview to flag items that should be reviewed. Or, you could use #errorchecks to flag items that need to have their status tested. The author name you entered is automatically associated with your annotations, so you don't need to include your name in a #key:value pair. You can type multiple keys in a single annotation, and keys can occur anywhere within the annotation text. If you want to use a # sign in the annotation text without having it treated as a key, type ##.




5 If you want to create multiple annotations for the same item, click the "+" icon. The current date and time are added to the left box. You can select annotations using this timestamp when you want to edit annotations.

6 Click **Save**. Your annotation appears in the source code where the selected entity is defined. (See page 186 to control how annotations are displayed.)

You can also add an annotation by clicking on an entity and choosing **Annotations > Annotate > Annotate <entity>** from the menus. Options are shown to annotate the current file, the current entity, and any entities that contain the current entity (such as a function that contains the selected variable).

### Editing an Annotation

To edit an existing annotation, follow these steps:

1. Highlight an entity that has an annotation anywhere in *Understand*.
2. Right-click on the entity and choose **Annotate** and the entity you want to annotate from the context menu.
3. In the left box, select the timestamp for the annotation you want to edit.
4. Modify the annotation text in the right box.
5. Click **Save** to store your changes.

If you edit an annotation that was originally created by someone else, that other person remains the author of the annotation. The timestamp for the annotation is updated to the last time it was edited.

### Deleting an Annotation

To delete an existing annotation, follow these steps:

1. Highlight an entity that has an annotation anywhere in *Understand*.
2. Right-click on the entity and choose **Annotate** and the entity whose annotation you want to delete from the context menu.
3. In the left box, select the timestamp for the annotation you want to edit.
4. Click the “-” (minus) icon above the list of timestamps.
5. Click **Save** to finish deleting the annotation.

### Managing Annotation Files and Display

You can choose **Annotations > Annotation Options** from the menus, and then set the following options:

- Your name or username to identify the original author of your annotations.
- The annotation files to look in for this project, and which file is the default for annotations you add. For example, you can have *Understand* display annotations from separate files for everyone working on this project.
- The foreground and background colors to use when displaying annotations from each of the files.
- How to display annotations: inline, as hover text, and with an indicator.

See *Annotations* on page 59 for details about setting these options.

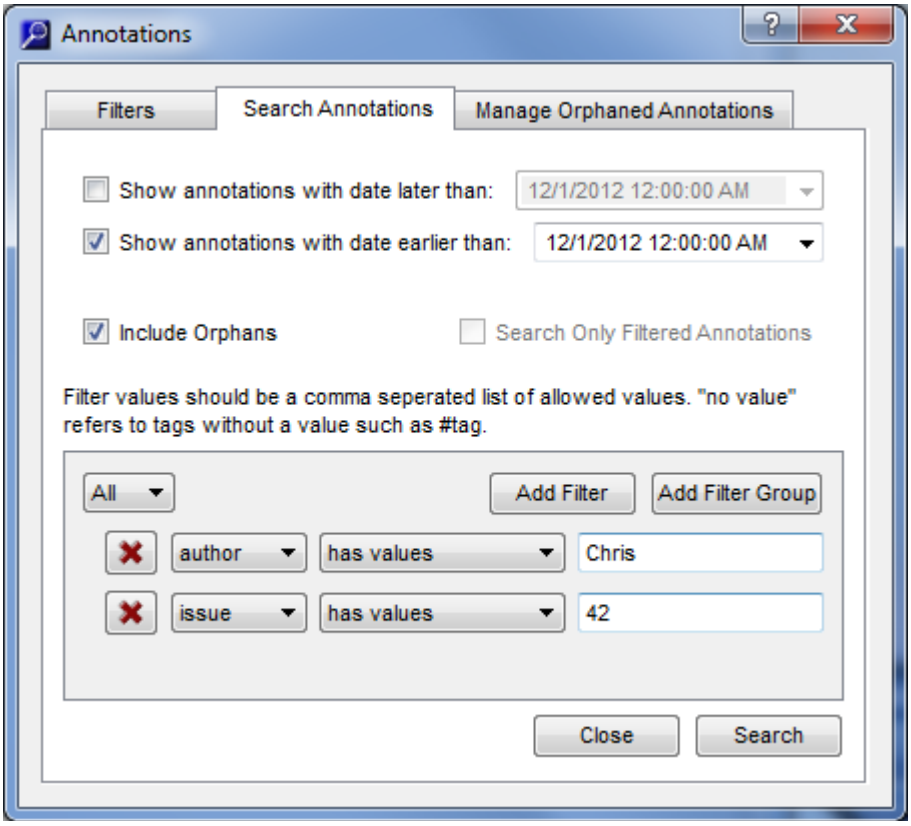
If other developers are also annotating code using *Understand*, choose **Annotations > Refresh Annotations** from the menus when you want to get the latest annotations they have added.

## Searching Annotations

You can search for annotations based on the key:value pairs, the author, and the timestamp. To search annotations, follow these steps:

1 Choose **Annotations > Search Annotations** from the menus.

2 Specify any of the following search parameters you want to use.




- **Date range:** Check one or both date range boxes if you want to find annotations edited after a certain date and time and/or annotations last edited before a certain date and time.
- **Include Orphans:** Check this box if you want the search to also find annotations that are linked to entities that have been deleted. See page 189 for more about orphaned annotations.
- **Search Only Filtered Annotations:** Check this box if you want to limit the search to annotations that match the current filters. See page 188 for more about filtering annotations.
- **Filter values:** You can set up one or more filters for the search based on the author and any #key:value pairs in the annotations. The “has values” and “doesn’t have value” options let you type a value to match or exclude for a #key:value pair. Exact matches for the author name and key values must be used; partial matches and wildcards are not supported. The “any value” option matches any annotation that has that #key, no matter what the value. The “no value” option matches annotations that have that #key, but no #key:value pair. Use **Add Filter** to create another filter, and choose **All** or **Any** to determine how matching is performed.

You can even use **Add Filter Group** to create nested levels of filters that have different settings for **All** and **Any**.

3 Click **Search**. The results are shown in the Annotation Search Results area in the main *Understand* window.

4 Expand the search results, and double-click on an item to go to the location where that annotation appears in the code. (That is, the location where the entity associated with the annotation is defined.)

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

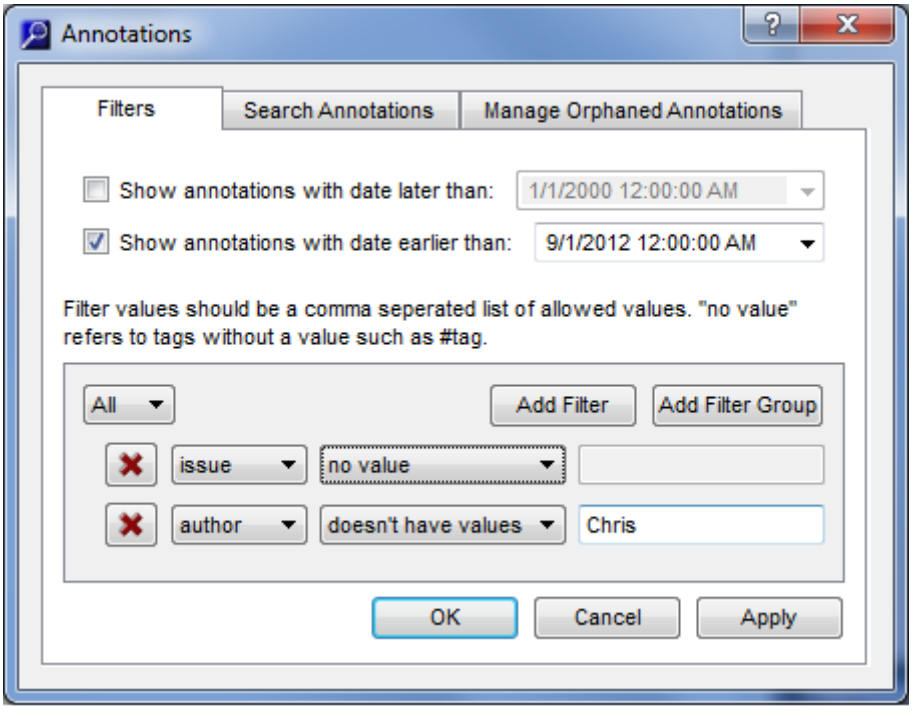
If other developers are also annotating code using *Understand*, choose **Annotations > Refresh Annotations** from the menus when you want to get the latest annotations they have added.

## Filtering Annotations

You can filter annotations based on the key:value pairs, the author, and the timestamp. To filter annotations, follow these steps:

1 Choose **Annotations > Filter Annotations** from the menus.

2 Specify any of the following filters you want to use.




- **Date range:** Check one or both date range boxes if you want to find annotations edited after a certain date and time and/or annotations last edited before a certain date and time.

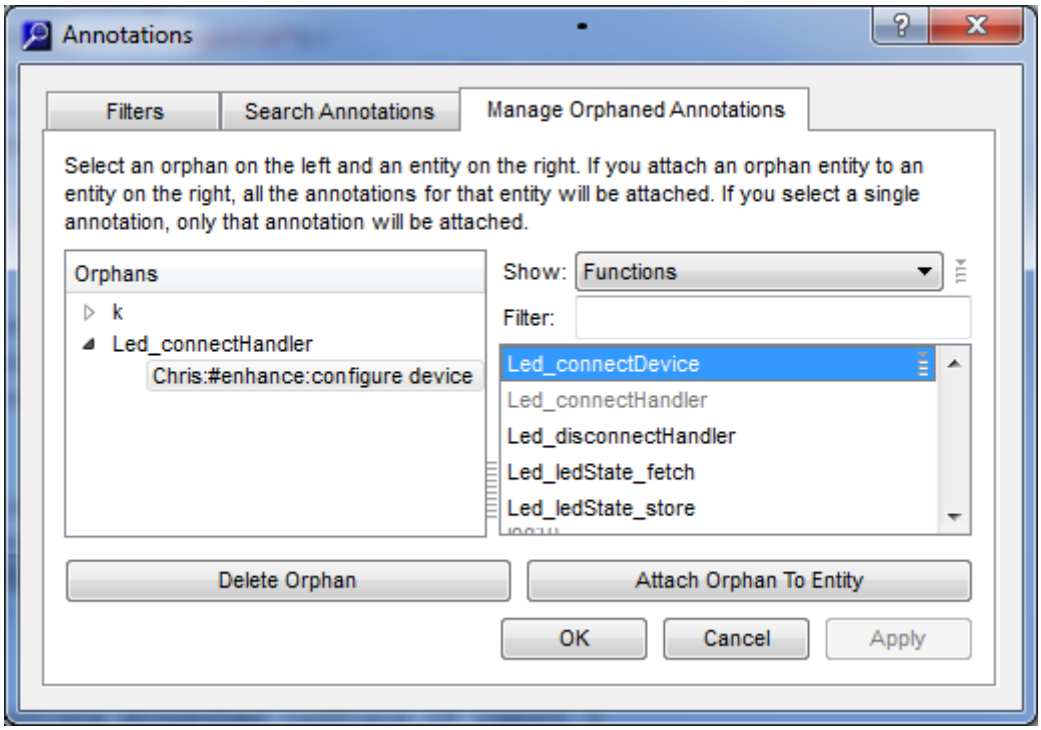
- **Filter values:** You can set up one or more filters for the search based on the author and any #key:value pairs in the annotations. See page 187 for details on using these fields.

3 Click **OK** or **Apply**. The filters you specify are applied to the annotations shown throughout *Understand*.

## Managing Orphaned Annotations

If you create an annotation, and later delete the entity with which it was associated, that annotation becomes an “orphan” when you re-analyze the project. Orphan annotations aren’t shown in the code anywhere. You can manage orphan annotations by choosing whether to delete or re-attach them. To manage orphan annotations, follow these steps:

1 Choose **Annotations > Manage Orphaned Annotations** from the menus.




2 Expand an orphan in the list on the left to see the annotation text.

3 If you want to delete the selected annotation, click **Delete Orphan**.

4 If you want to attach the selected annotation to a different entity, select an entity from the list on the right. (You can shorten the list by selecting a type of entity from the **Show** drop-down.)

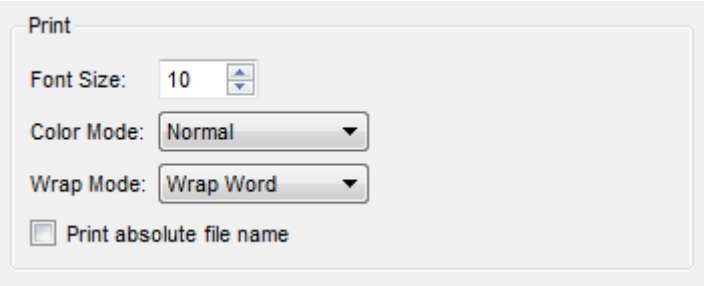
5 Click **Attach Orphan To Entity** to connect the selected orphan to the selected entity. The annotation will be shown in the code where the new entity is defined.

6 Click **OK**.

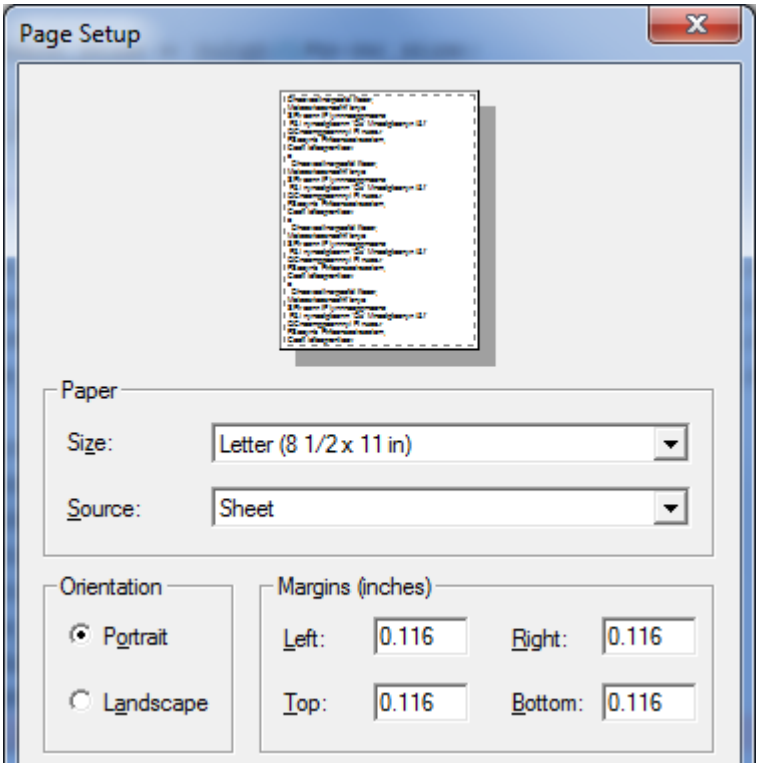
# Printing Source Views

The menu option **File > Print** opens the standard print dialog for your operating system so you can print the currently viewed source file. The output shows 66 lines per page.

By default, files are printed in the font and color shown on the screen when you choose the **File > Print** menu option. You can customize code printing in the Options dialog. To open this dialog, choose **Tools > Options**. Expand the **Editor** category, and select the **Advanced** category. Options to control how code is printed are in the Print area. See *Editor > Advanced Category* on page 111 for details about these fields.




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



# Chapter 7 Architecting Your Codebase

This chapter explains the architecture features provided by *Understand* and explains how you can use them to analyze your code.

This chapter contains the following sections:

# About Architectures

An architecture is an abstract hierarchy layered onto a body of source code. For example, a staff architecture could have nodes for each engineer working on a particular project. The nodes would contain a list of source code files belonging to or to be modified by that engineer. Dependencies and interactions could then be derived from that architecture.

Architectures allow you to name regions of a software project or ways of looking at software hierarchically. An architecture creates a hierarchy of source code units (entities). You can use the provided architectures or create your own.

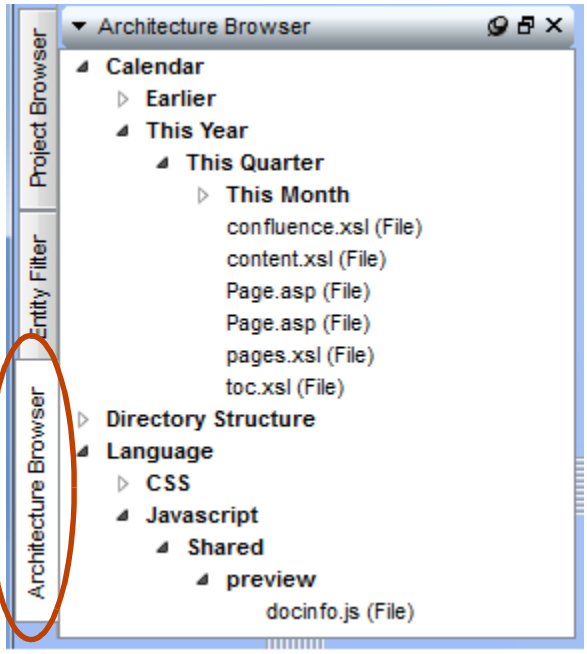
Architectures need not reference every source entity in the database; that is, they can define a subset of the entities. Also, architectures can contain a particular entity more than once. (Technically, that is, the architecture's flattened expansion need not maintain the set property.)

You can combine architectures successively to create novel filters for your entities.

From a more technical perspective, simple set algebra is used to combine and transform architecture hierarchies. The result of the filter is a list of entities. This result list can be viewed as a flat list or in terms of another architecture. The filter definition can be saved as a dynamic architecture. A dynamic filter architecture is updated as the contents of the database change and it can be used to reconstitute the filter at a later date.

# Using the Architecture Browser

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




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

This Architectures area is similar to the Filters area. When you click on an item, information about it is automatically shown in the Information Browser (as long as the “Sync” box is checked in the Information Browser).

## Exploring Architectures

To explore the existing architectures, click the “+” signs to expand the hierarchy. Entities, such as files, functions, and variables are shown in the hierarchies.

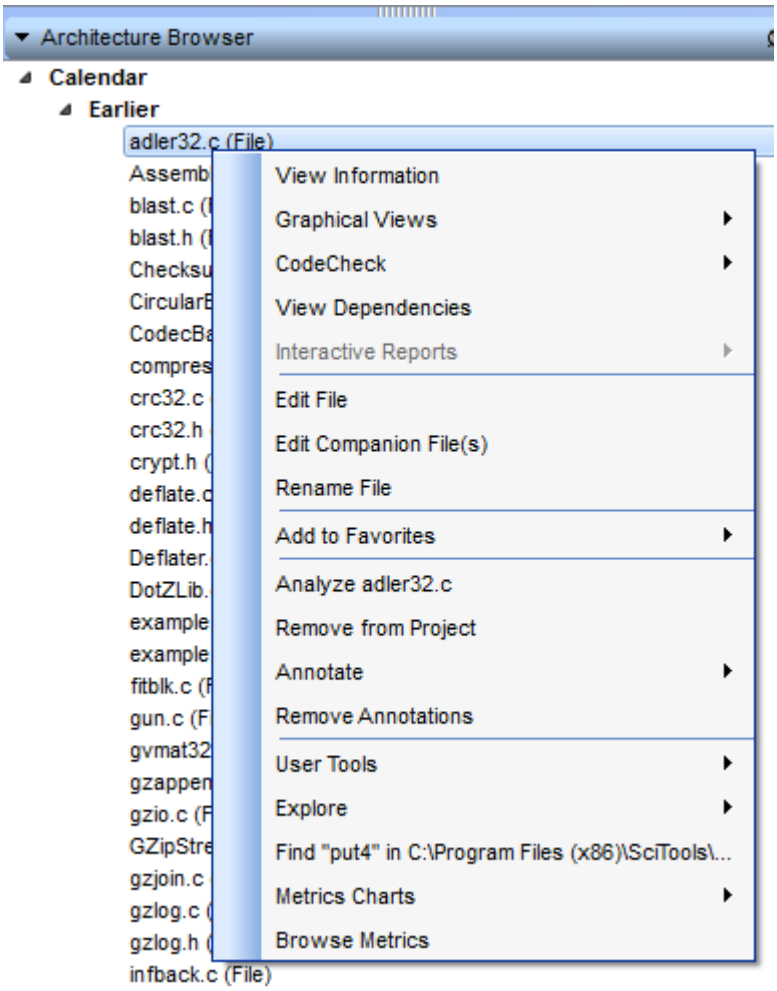
*Understand* provides some “auto-architectures” that are built in:

- **Directory Structure:** Lists the project files in their normal file hierarchy—showing directories and their subdirectories.
- **Calendar:** Lists files in the project according to their last change date. A hierarchy of dates is shown that progresses from This Year, This Quarter, This Month, and This Week to Yesterday and Today.
- **Language:** Lists files first by their source code language and then by their location in the directory structure. (This architecture exists only if your project contains multiple languages.)
- **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.)

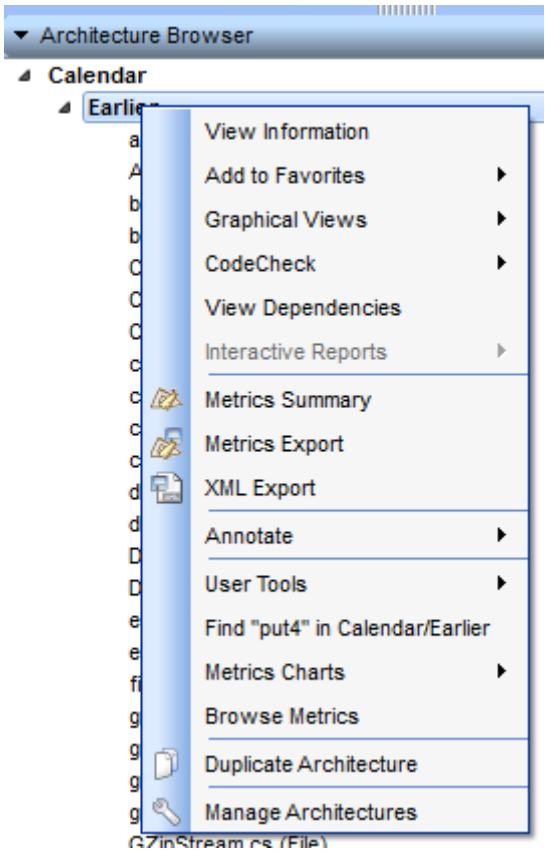
The auto-architectures are updated only when the project is analyzed. So, if your source code is actively being modified and you have not analyzed it recently, architectures—especially the Calendar architecture—could be out-of-date.

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**




Notice that the context menu for an architecture node (such as a filesystem directory or "This Quarter") contains some extra items not available in other context menus:

- **Graphical Views > Graph Architecture:** Creates a graph of the architecture hierarchy from this point down. You are asked whether you want to include entities in the graph or just the architecture nodes. See page 196.
- **Graphical Views > Dependency Graphs:** Shows the dependencies between architecture nodes. See page 196.
- **Metrics Summary:** Provides metrics for the entities within the selected node. The metrics are based on entities in the current node, but not those in sub-nodes lower in the hierarchy. See page 200.

- **Metrics Export:** Creates a CSV output of the metrics from the Metrics Summary. See page 200.
- **XML Export:** Creates an XML export listing the architecture nodes and entities from the selected point down in the hierarchy. See page 206.
- **Edit Architecture:** Opens the Architecture Builder for the selected architecture if it is one you created. You cannot edit the auto-architectures provided with *Understand*. See page 204.
- **Rename Architecture:** Opens a Rename Architecture window that lets you rename the selected architecture or node if it is one you created. You cannot rename the auto-architectures provided with *Understand*. See page 202.
- **Duplicate Architecture:** Opens a Duplicate Architecture window that lets you type a name for a duplicate copy of the selected architecture. See page 202.
- **Manage Architectures:** Opens the Architect Manager window. See page 201.

# Viewing Architecture Dependency Graphs

You can generate graphs that show the hierarchy of an architecture. You can save these graphs as PNG, JPEG, SVG, Visio XML, and DOT files.

***Note:*** Dependency graphs are also available for classes and packages.

To create a graph, follow these steps:

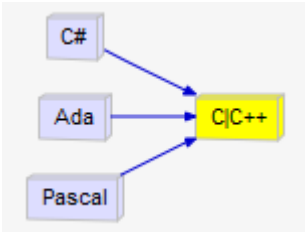
1. Select the highest-level architecture node you want to graph. You can graph the entire hierarchy or just a sub-hierarchy.
2. Right-click on the node and choose **Graphical Views** from the context menu. Depending on the node you select, the submenu allows you to choose **Graph Architecture**, **Depends On**, **Depended On By**, **Butterfly-Dependency Graph**, and **Internal Dependencies**. When you have selected an architecture node, the same list of graphical views is available by choosing **Graphs > Graphs for *<current_entity>*** from the menus.

To open the Internal Dependencies graph for an entire architecture, choose from the **Graphs > Dependency Graphs** menu.

Architecture dependency graphs have the same toolbar as other types of graphical views. See page 250 for details about using the icons in the graphical view toolbar.


To save a graph as a JPG, PNG, or SVG file, see page 276. To save a graph to a Visio file, see page 276.

Dependency and relationship graphs provide an additional Graph Customizer toolbar that you can use to modify the graph display. This toolbar lets you control expansion, highlighting, and arrows on a per-node basis. It also lets you undo and redo your changes, and save and load graph customizations. For example, this is the default Depended On By graph for the C|C++ node in the multi-language zlib sample project.




Nodes that are drawn as 3D boxes (like those in this figure) can be expanded to shown the nodes they contain by double-clicking on them. You can keep expanding nodes until you get to the file level.

You can hover your mouse cursor over a line that connects two boxes to see which items are connected by the relationship and to highlight the line. (See page 118 if highlighting is disabled.)

Click the **View Dependencies** button or right-click on a dependency graph and choose **View Dependencies** to open the Dependency Browser (page 140) for the selected node or relationship. 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. In addition, **Show** and **Group By** settings from the dependency graph are synced with the Dependency Browser and the **Dependency Kind** is changed to Depends On.

You can right-click on the gray background of a dependency graph (outside any colored boxes) to control whether long, short, or relative names are displayed for architecture node names and filenames. In addition, you can enable or disable the reference count numbers that show how many times a particular dependency occurs. See Controlling Cluster Graph Layout on page 272 for details about the context menu options within architecture dependency graphs.

The context menu when you right-click on a node in a dependency graph offers commands similar to those you see when right-clicking on an entity or architecture node name elsewhere in *Understand*.

The context menu when you right-click on an edge (arrow) in a dependency graph provides a list of the references that constitute the edge. Choose an item from the list to visit the source code for this relationship. You can limit the length of this list as described for the **Tools > Options** dialog on page 118.

You can customize the display colors, shapes, and arrows used in cluster graphs in the Graphs category of the **Tools > Options** dialog (page 118).

### Graph Customizer Toolbar

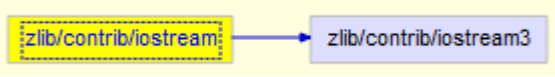
The toolbar icons in the Graph Customizer pane perform the following actions:

- **Save icon.** Prompts you for a name for the current settings. Settings apply only to the specific graph type and root node in this view. If you have already saved settings for this graph type/root node combination, you can select a set you want to update from the context menu. Otherwise, type a name for your current settings and click **Save**.
- **Load icon.** Prompts you to select a named group of graph settings that you want to open in the current window. The list shows only settings saved for this graph type/root node combination. To see the full list of saved settings, choose **Graphs > Dependency Graphs > Load Saved Dependency Graph**.
- **Undo icon.** Undo your last change.
- **Redo icon.** Redo the last change you undid.
- **Restore Defaults.** Restores graph to the settings it had when you opened it.
- **Selected Node tools.** Use this drop-down menu to control the display of the currently selected nodes and edges. The icon for this toolbar item changes to reflect the most recently used command. The commands are active depending on whether you have selected nodes, whether any selected nodes have edges coming in or out, and whether any selected nodes have children.

  You can select one or more nodes in a dependency graph by using your mouse to drag a rectangle over the nodes you want to select. Or, hold down the Ctrl key while clicking on multiple nodes you want to select.

  - **Show Selected Node 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 Child Edges Going 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 Child Edges Coming 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.
- **Highlight Edges Going Out.** Causes the selected node to be highlighted in yellow. Arrows pointing from this node to other nodes become darker. Nodes to which they point are highlighted in light blue. Internal Dependency graphs let you highlight such edges; other types of dependency graphs let you show or hide such edges.




- **Highlight Edges Coming In.** Causes the selected node to be highlighted in yellow. Any arrows that point to this node from other nodes become darker, and nodes which point to this node are highlighted in light blue. Internal Dependency graphs only let you highlight such edges; other types of dependency graphs let you show or hide such edges.


- **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 the Show All Hidden Nodes button.)

• **Global Node tools.** Use this drop-down menu to show and hide various nodes and edges. The icon for this toolbar item changes to reflect the most recently used command.

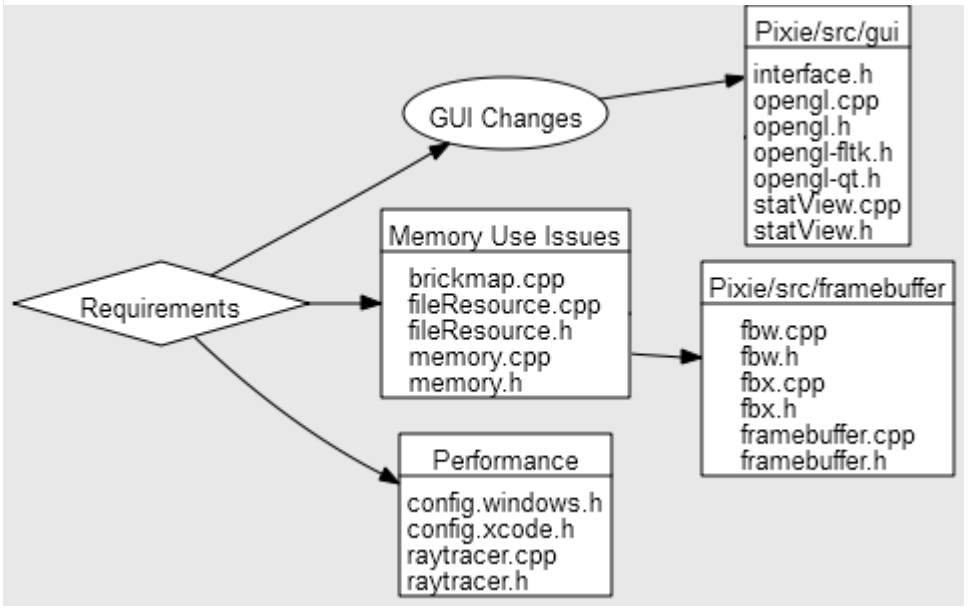
- **Hide Nodes With No Highlighted Edges.** This field is available only for Internal Dependency graphs, and you can use it only if you have turned on highlighting of "edges." If you check this box, all nodes that do not have a highlighted arrow pointing to it or away from it are hidden, and the graph is reorganized as needed to omit those nodes.
- **Hide Unhighlighted Edges.** This field is available only for Internal Dependency graphs, and you can use it only if you have turned on highlighting of "edges," which are the connections between nodes. If you check this box, all arrows that are not highlighted are hidden, and the graph is reorganized as needed to omit those non-highlighted relationships.
- **Clear All Highlighted Edges.** This button is available only for Internal Dependency graphs. If you click this button, all node and "edge" highlighting is removed.

- **Show All Hidden Nodes.** If you click this button, any nodes that have been hidden using the “Hide Selected Nodes” button are restored. This button does not expand any nodes that have been contracted to hide child nodes. If you have hidden any nodes, you can select entities from the **Hidden Nodes** drop-down list to redisplay those nodes.
- **Dependencies 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. If you have the **Sync** box checked in the Dependency Browser, the current Show settings in the dependency graph are copied to the Dependency Browser when you click on a relationship in the graph.
- **Open Dependency Browser.** This button opens the Dependency Browser (page 140) for the most recently selected node or relationship in the graph. Syncing between the dependency graph and the Dependency Browser is turned on by default when you open the Dependency Browser this way.

### Graph Architecture View

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 248).

This type of graph does not provide a Graph Customizer panel, but you can right-click on the graph to modify the display. For example, in the following Architecture Graph, **Include Entity Lists** was off by default but was turned on by right-clicking.

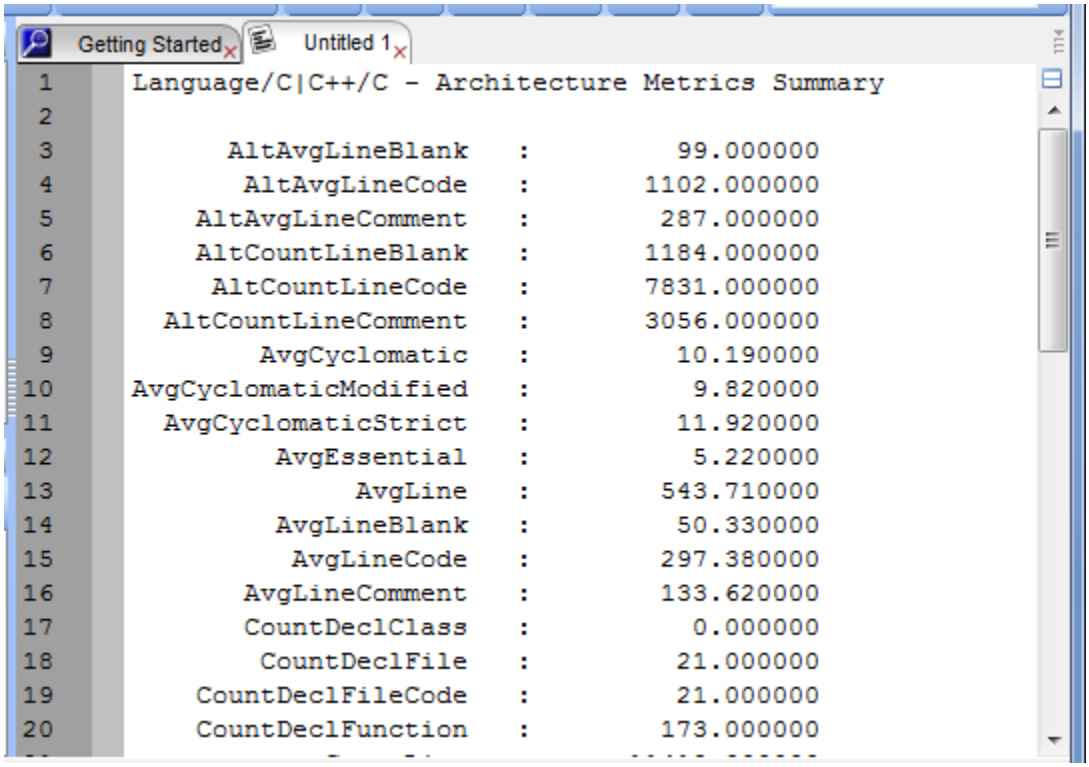


# Viewing Architecture Metrics

You can generate metrics information about an architecture or a subset of an architecture. The metrics information can be either a text summary or a comma-separated list for use in spreadsheets.

To create a metrics summary, follow these steps:

1. Select the highest-level node of the architecture for which you want metrics.
2. Right-click on the node and choose **Metrics Summary** from the context menu.
3. You see an Architecture Metrics Summary window.



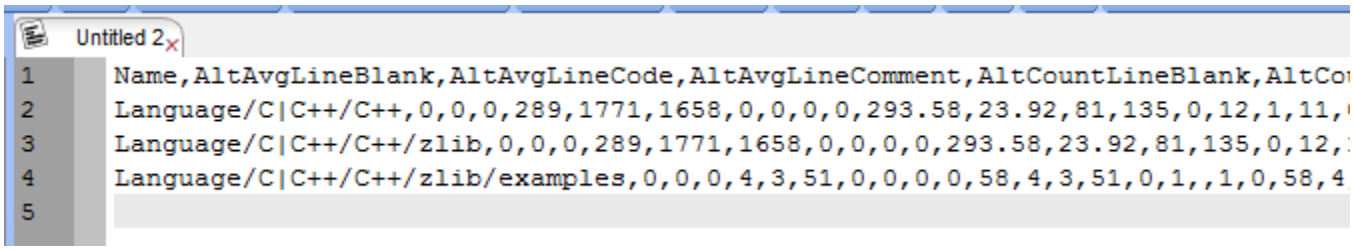

4. When you close the window, you are asked whether you want to save the file. If you click **Save**, you can save the summary as text.

To create a metrics export file, follow these steps:

1 Select the highest-level node of the architecture for which you want metrics.

2 Right-click on the node and choose **Metrics Export** from the context menu.

3 You see a comma-separated values file. The heading label for each column is in the first row. Each node in the architecture hierarchy has a separate row with metrics for that node's contents.




4 When you close the window, you are asked whether you want to save the file. If you click **Save**, you can save the data as a .CSV file.

# Managing Architectures

To open the Architect Manager window, choose **Project > Architectures > Manage Architectures** from the main menu bar in *Understand*. The window lists the auto-architectures on the right and custom architectures you have created on the left.


The checkboxes allow you to control whether custom and auto architectures are shown in the Architectures area. Removing the checkmark next to an architecture can improve performance, especially for large projects. So, you might want to disable/hide architectures you never or rarely use.

You can use the icons at the top of this area or right-click on an architecture to perform the following actions:

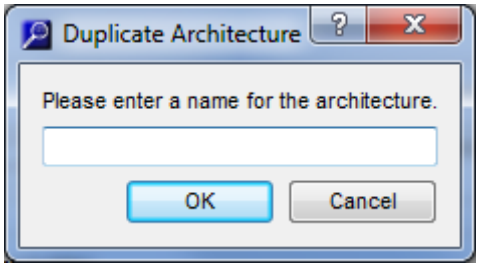
- **Create a new architecture:** See page 202.
- **Edit architecture:** Predefined and custom architectures only. See page 204.
- **Rename architecture:** Predefined and custom architectures only. See page 202.

- **Duplicate architecture:** See page 202.
- **Delete architecture:** Predefined and custom architectures only.
- **Import architecture from XML:** See page 206.
- **Export architecture to XML:** See page 206.

## Creating an Architecture

There are several ways to create a new architecture:

- To create an architecture from scratch, choose **Project > Architectures > New Architecture** from the menus or click the icon in the Architect Manager. Use the Architecture Wizard to create the architecture as described in *Using the Architecture Wizard* on page 203.
- To duplicate an existing architecture (which you can then modify), select an architecture and click the icon in the Architect Manager window. Or, right-click an existing architecture node in the Architecture Browser and choose **Duplicate Architecture** from the context menu to create an architecture from that node and lower in the hierarchy.

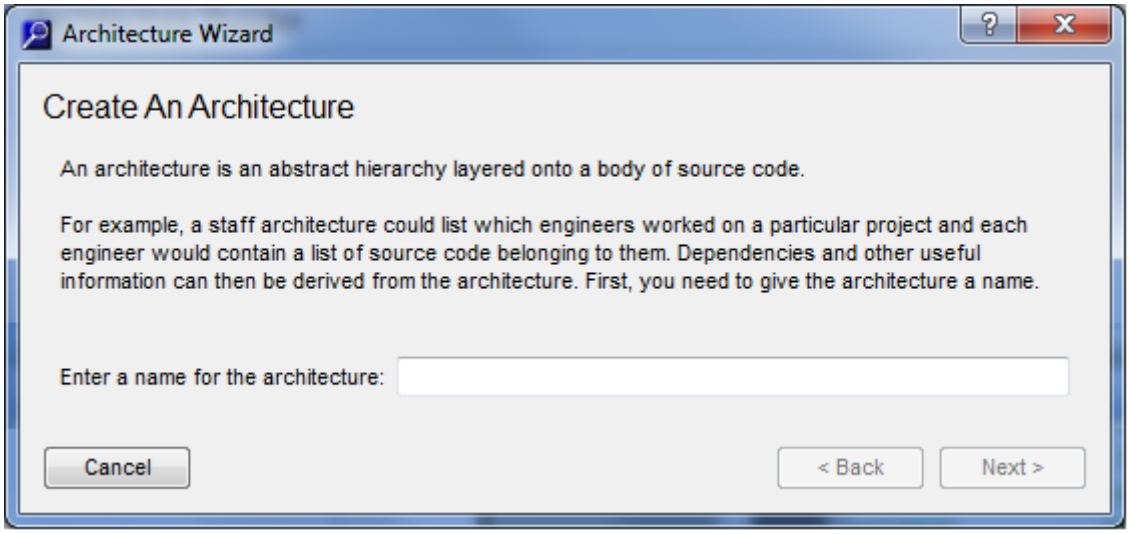



You can rename an architecture you have created by selecting an architecture and clicking the icon in the Architect Manager window. Or, right-click on an existing custom architecture and choose **Rename Architecture** from the context menu.

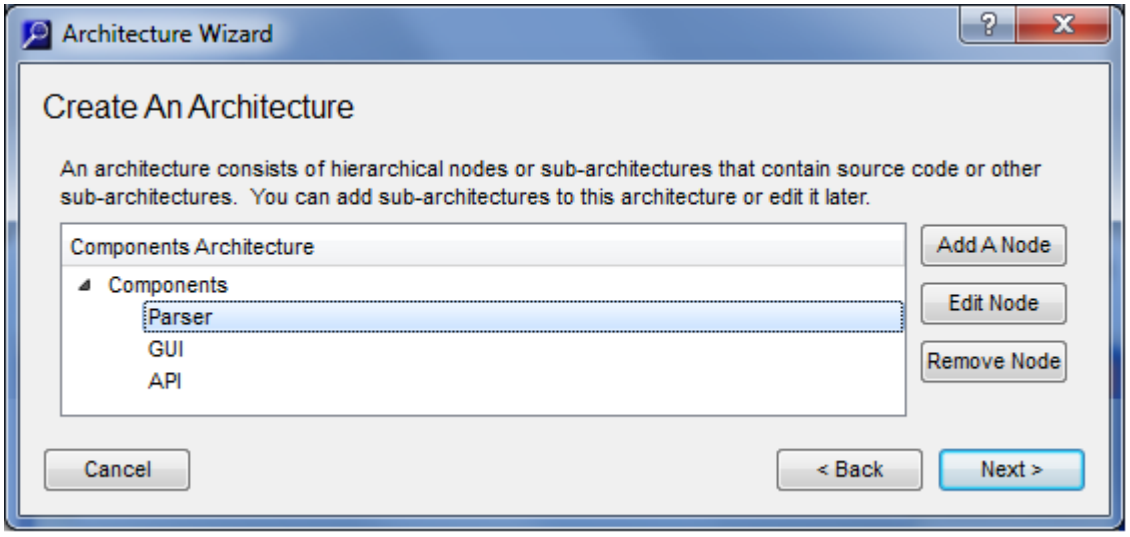
## Using the Architecture Wizard

When you open the Architecture Wizard by choosing **Project > Architectures > New Architecture** from the menus or clicking the icon in the Architect Manager window, you see a page that asks for the name of your architecture.

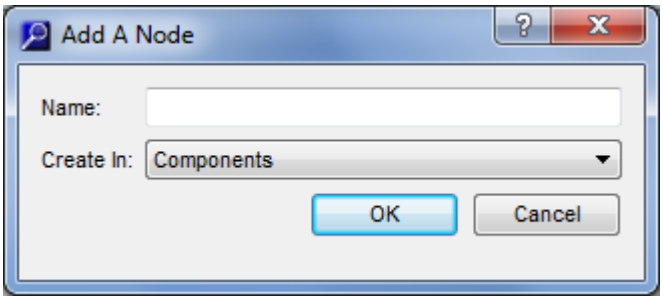
Type a name for the architecture. This name should be fairly short so it can be shown in architecture trees.




Then click **Next** to see the page that lets you add and edit architecture nodes. This is the hierarchy to which entities will be assigned in a later page of the wizard



Click **Add a Node** and type the **Name** of the node you want to add. The default location is within the node you had selected in the Architecture Wizard, but you can select another location in the **Create In** field. Then click **OK**.

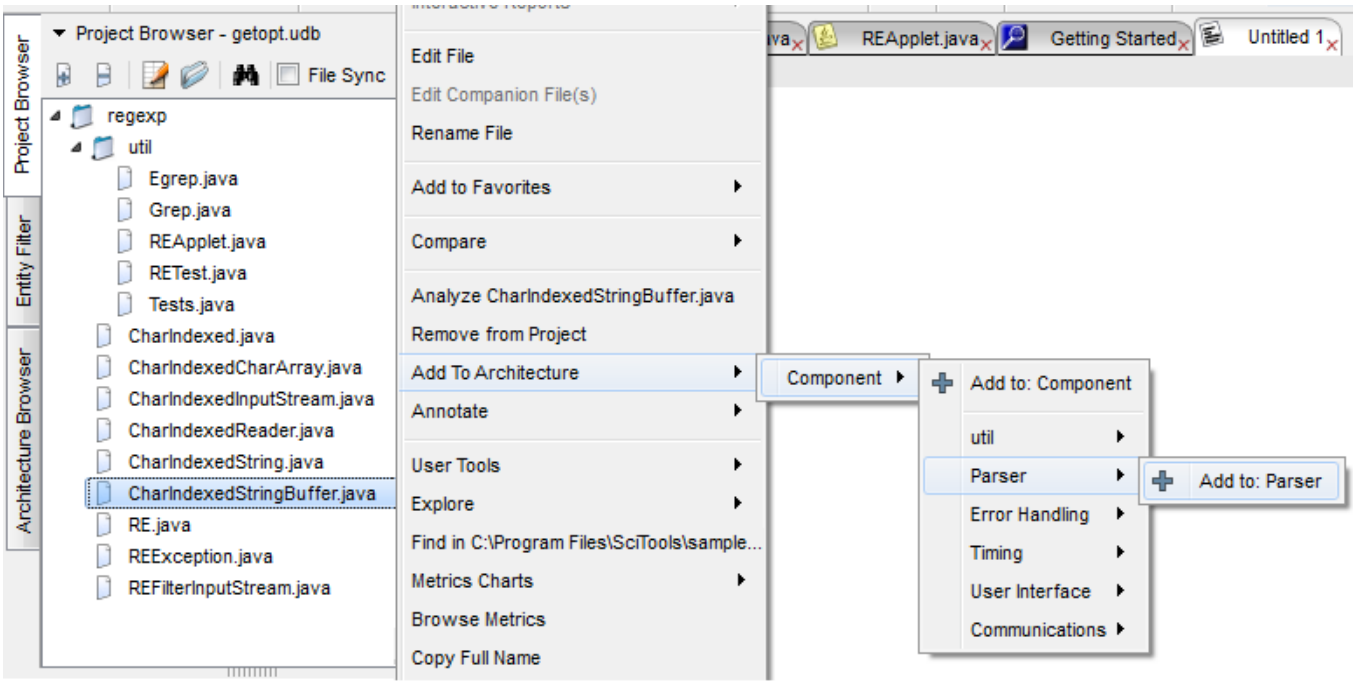



You can modify nodes you have created by selecting a node and clicking **Edit Node**. You can delete the selected node by clicking **Remove Node**.

The next window presents an animation that shows how to use the Architecture Builder to add entities to the nodes you have created. When you have finished watching the animation, click **Finish**. This opens the Architecture Builder shown in the animation. Your architecture nodes are shown on the right. See *Editing an Architecture* on page 204 for details on adding entities to each node.

## Editing an Architecture

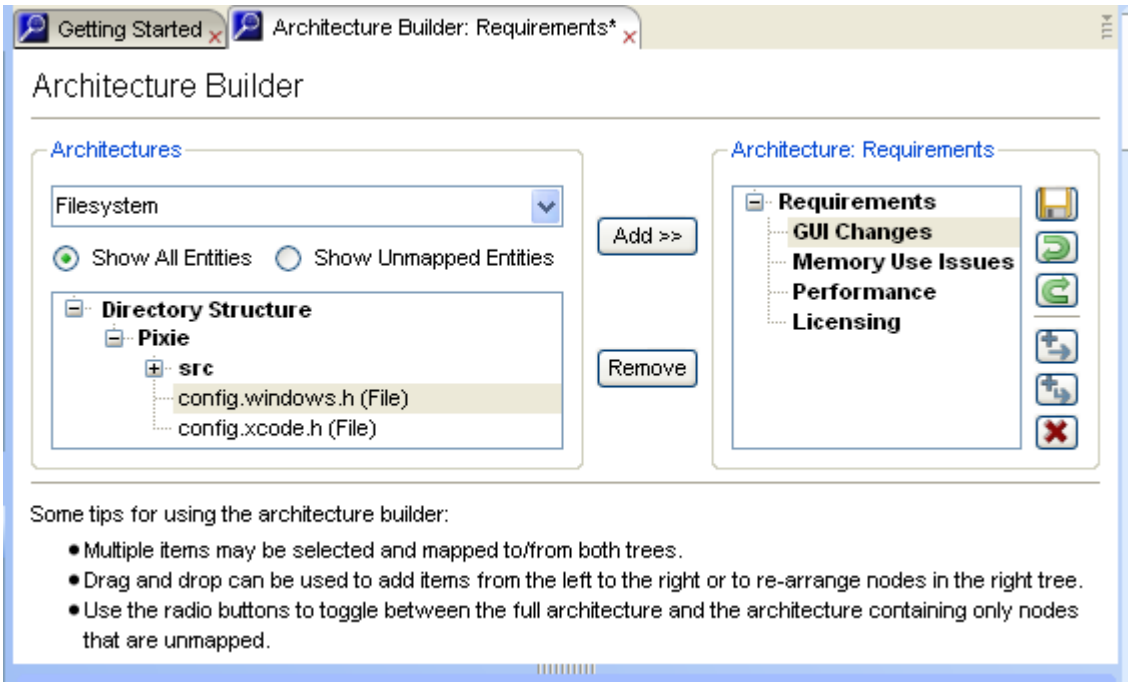
You can quickly add a file to an existing custom architecture by right-clicking on a file in the Project Browser and choosing **Add to Architecture** and the architecture node within which you want to place the file.



To make changes to an architecture beyond adding files, select that architecture and click the icon in the Architect Manager window. Or, right-click on an existing architecture and choose **Edit Architecture** from the context menu. Both actions open the Architecture Builder.

You cannot edit the Auto Architectures provided with *Understand*. However, you can use the icon in the Architect Manager window to create a duplicate architecture of one of the Auto Architectures. Then, you can edit the duplicate architecture.

This dialog allows you to add nodes to architectures. You create an architecture structure on the right-hand side and map entities into the architecture from the left-hand side.




To create and edit nodes in the Architecture Builder, follow these steps:

- Double-click the name of any node on the right side of the Architecture Builder, and rename that node by typing. (Or you can select a node and press Enter.)
- Move one or more nodes by dragging them to the node you want to be the parent node. Within a node, the children are sorted alphabetically.
- Click the icon to create a new node at the same level as the selected node. Click the icon to create a new node as the child of the selected node.
- Click the icon to delete the selected node.
- Click the icon to undo your last change. Click the icon to redo you last undo.

To map files to nodes in the Architecture Builder, follow these steps:

**1** On the left side of the Architecture Builder, select an existing architecture from the drop-down list that will allow you to easily find the files you want. The default is the Directory Structure architecture.

2 You can choose whether to show all entries in the architecture or just the unmapped entries. For example, if you want to map all the entries into your new architecture, you might want to select **Show Unmapped Entries** so that you can see which files you haven't mapped yet.

3 In the left architecture hierarchy, select one or more files or architecture nodes.

4 In the right architecture hierarchy, select the node you want to contain your selection.

5 Click the **Add** button or drag your selection to the right side.

6 When you finish editing your custom architecture, click **Save**.

You can use the **Remove** button to delete files and nodes from the architecture you are editing.

As always, you can right-click on any node or file to use its context menu to get information.

You can save your edits to the architecture at any point by clicking the  icon. Then, you can continue editing. If you close the Architecture Builder without saving changes, you will be asked if you want to save your changes.

## Using XML to Manage Architectures

You can use XML as a way to share architectures between one *Understand* database and another.

In addition to using XML to share architectures, you can use XML export/import to quickly create architectures that are a simple subset of another architecture by selecting a lower node in the hierarchy.

### Exporting Architectures to XML

To create an XML file for an architecture, follow these steps:

1 Select the highest-level node of the architecture that you want to export. All of the hierarchy below the node you select will be represented in the XML file.

2 Click the  icon in the Architect Manager window. Or, right-click on the node you selected and choose **XML Export** from the context menu.

3 You see an XML file that contains <arch> and <set> tags for architecture nodes.

4 When you close the XML window, you are asked if you want to save the file. If you click **Save**, the default filename is the name of the node you selected.

### Importing XML Architectures

To import an XML file for an architecture, follow these steps:

1 Click the  icon in the Architect Manager window.

2 In the Choose XML File to Import Architecture dialog, select an XML file that matches the tag format used by *Understand* to describe architectures. For example, you can choose XML files created by *Understand*. Click **Open**.

3 The architecture described by the XML file is added to your list of architectures.

# Chapter 8 Using Reports

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

This chapter contains the following sections:

# Configuring Reports

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

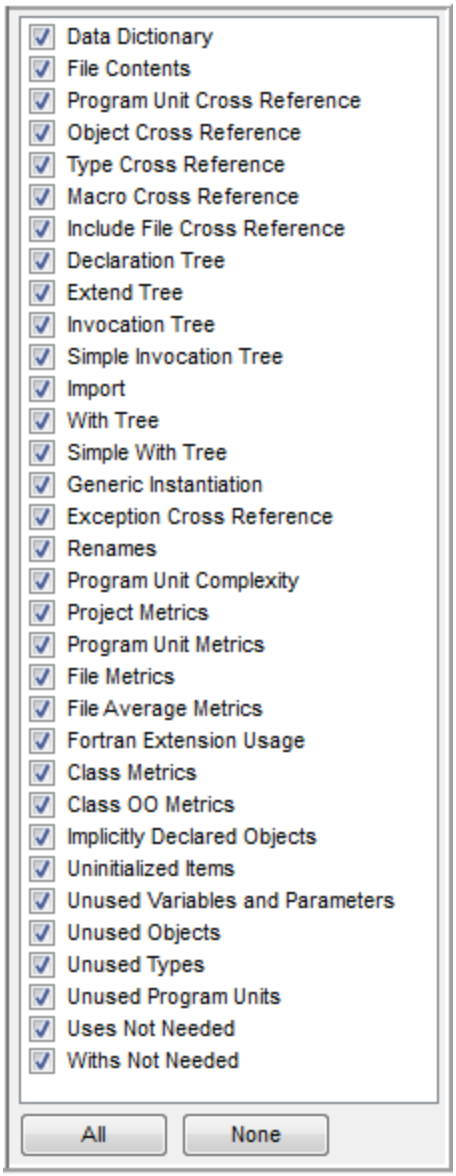
To configure how reports will be generated, choose **Reports > Configure Reports**. This opens the Project Configuration dialog with the **Reports > Output** category selected. From there, you can also configure the **Reports > Options** and **Reports > Selected** categories.

See page 54 for details on the **Reports > Output** category. In general, you can configure the following:

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

For details on the **Reports > Options** category, see page 55.

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




The specific reports available depend upon the source languages used in your project.

See *An Overview of Report Categories* on page 212 for descriptions of the types of reports you can generate.

### Customizing Report Colors

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

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

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

If you modify the stylesheet and want to use if 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 211.

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

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

***Note:*** HTML, text, and project metrics reports may also be generated with the “und” command line program. Refer to Chapter 14 for details.

# Viewing Reports

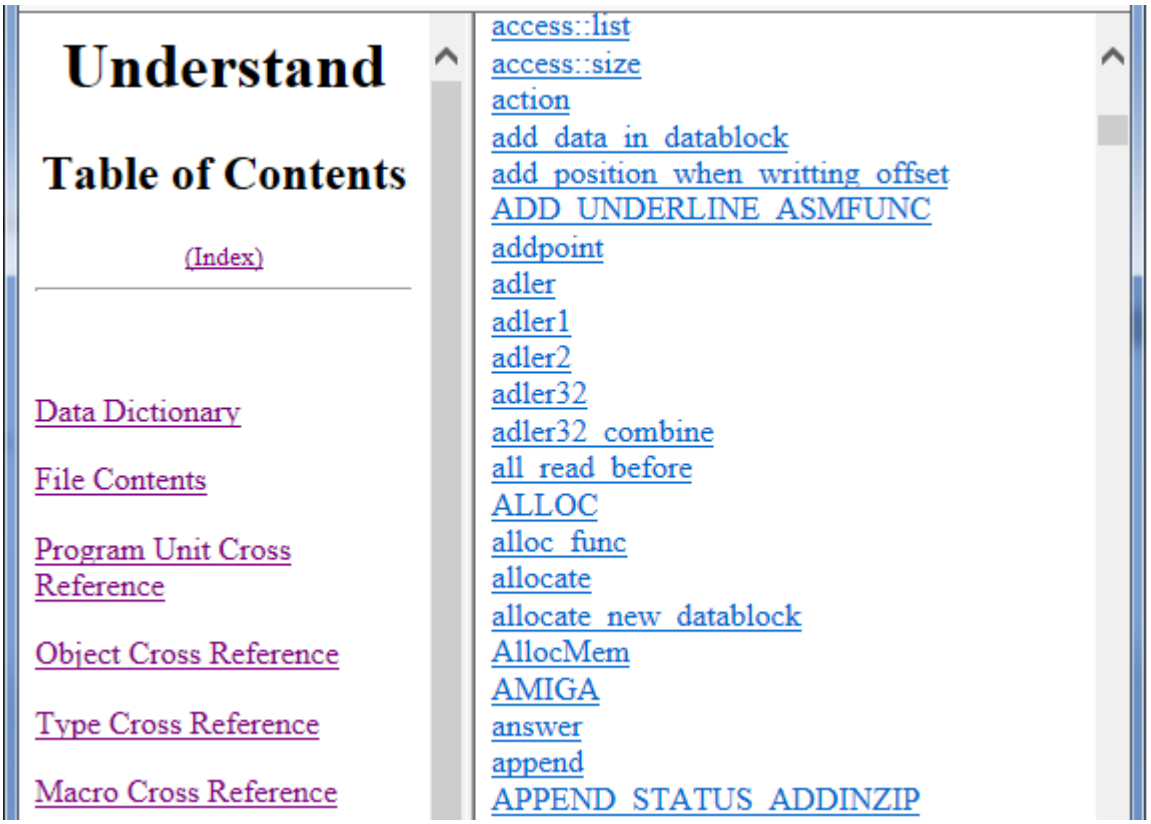
To view generated reports, choose **Reports > View Reports**. Then choose the **HTML** or **Text** option.

File names of reports generated vary based on the type and format of the report generated.

- For text files, a single text file containing all selected reports may be generated or separate files for each type of report may be generated. A single text file is named <*project_name*>.txt. For separate text files, the file name is the type of report.
- For HTML reports, you can generate either a single HTML files for each report type, or smaller files divided either alphabetically by entity name or in groups of N number of entities. An index file is also generated that contains links to all the other HTML reports generated. The main window page is named index.html.

For HTML reports, a single index file contains an alphabetic list of all entities found in all other generated HTML reports. The entities listed in the index have hyperlinks to the Data Dictionary report for that entity. The entity index file is named entity_index.html and can be accessed from the “index” link on the main HTML page.

The following figure shows an example of the entity index.



# An Overview of Report Categories

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

- **Cross-Reference** reports show information similar to that in the *Information Browser*, except that all entities are shown together in alphabetic order. See *Cross-Reference Reports* on page 214.
- **Structure** reports show the structure of the analyzed program. See *Structure Reports* on page 219.
- **Quality** reports show areas where code might need to be examined. See *Quality Reports* on page 222.
- **Metrics** reports show basic metrics such as the number of lines of code and comments. See *Metrics Reports* on page 226.

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 | *Data Dictionary Report* on page 214 |
| Cross-Reference | *File Contents Report* on page 215 |
| Cross-Reference | *Program Unit Cross-Reference Report* on page 215 |
| Cross-Reference | *Object Cross-Reference Report* on page 216 |
| Cross-Reference | *Type Cross-Reference Report* on page 216 |
| Cross-Reference | *Macro Cross-Reference* on page 217 |
| Cross-Reference | *Include File Cross-Reference* on page 217 |
| Cross-Reference | *Exception Cross-Reference Report* on page 218 |
| Structure | *Declaration Tree* on page 219 |
| Structure | *Extend Tree* on page 220 |
| Structure | *Invocation Tree Report* on page 220 |
| Structure | *Simple Invocation Tree Report* on page 220 |
| Structure | *Import Report* on page 221 |
| Structure | *With Tree Report* on page 221 |
| Structure | *Simple With Tree Report* on page 221 |
| Structure | *Generic Instantiation Report* on page 221 |
| Structure | *Renames Report* on page 221 |
| Quality | *Program Unit Complexity Report* on page 222 |
| Quality | *Uninitialized Items* on page 224 |
| Quality | *Unused Objects and Functions* on page 224 |
| Quality | *Unused Objects Report* on page 224 |
| Quality | *Unused Types Report* on page 225 |
| Quality | *Unused Program Units Report* on page 225 |
| Quality | *Uses Not Needed Report* on page 225 |
| Quality | *Withs Not Needed Report* on page 225 |

| Report Type | Report Name and Page |
|---|---|
| Quality | *Implicitly Declared Objects Report* on page 223 |
| Quality | *Fortran Extension Usage Report* on page 223 |
| Metrics | *Project Metrics Report* on page 226 |
| Metrics | *Program Unit Metrics Report* on page 228 |
| Metrics | *File Metrics Report* on page 228 |
| Metrics | *File Average Metrics Report* on page 229 |
| Metrics | *Class Metrics Report* on page 227 |
| Metrics | *Class OO Metrics Report* on page 227 |

## Augment with the PERL or C API

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

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

For details on the PERL interface choose **Help > PERL API Documentation**. Also visit the blog and 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 API. For information about creating plugins, please contact support@scitools.com. The SciTools Support page at scitools.com/support and the SciTools blog at scitools.com/blog also contain messages concerning plugins.

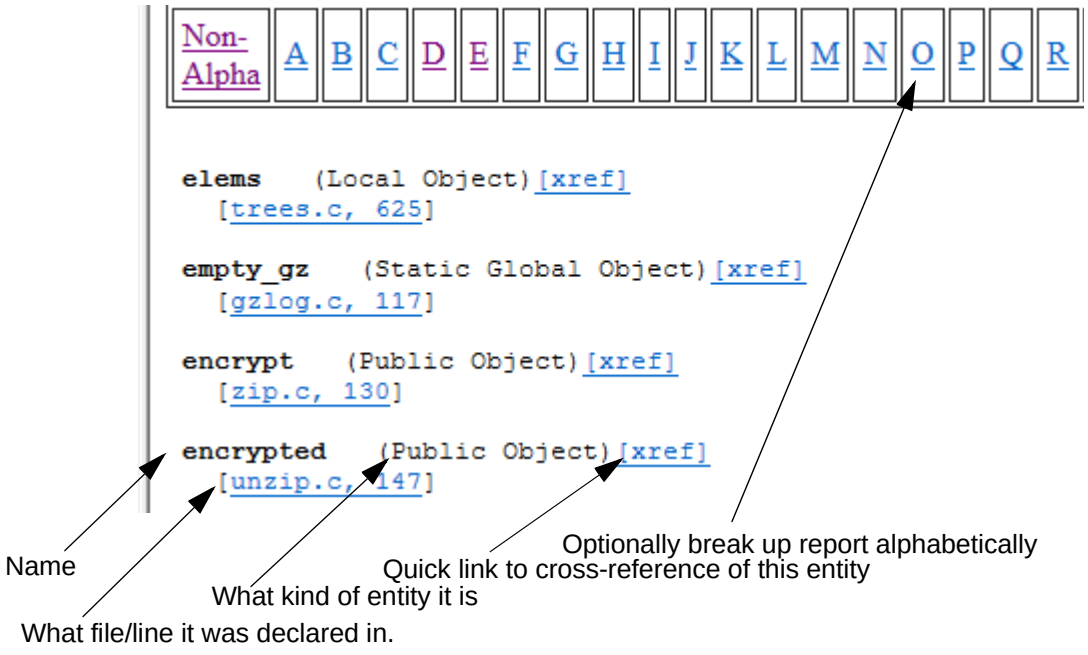
# Cross-Reference Reports

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

| Report Name |
|---|
| *Data Dictionary Report* on page 214 |
| *Program Unit Cross-Reference Report* on page 215 |
| *File Contents Report* on page 215 |
| *Object Cross-Reference Report* on page 216 |
| *Type Cross-Reference Report* on page 216 |
| *Class and Interface Cross-Reference* on page 216 |
| *Macro Cross-Reference* on page 217 |
| *Include File Cross-Reference* on page 217 |
| *Exception Cross-Reference Report* on page 218 |

## Data Dictionary Report

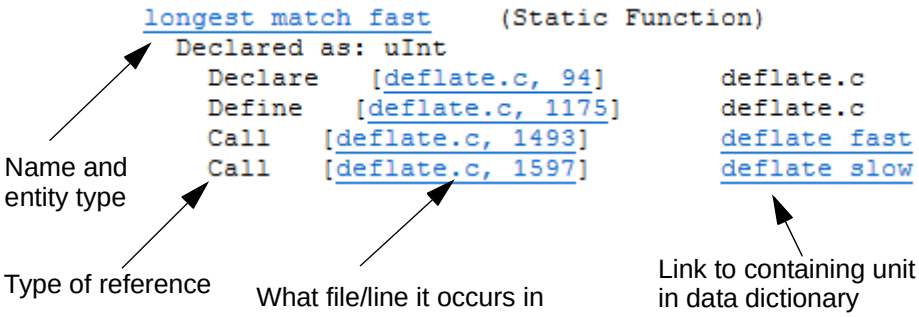
The *Data Dictionary Report* lists all entities alphabetically. Each listing shows the entity name, what kind of entity it is (for example, macro, type, variable, function, include, file, or procedure), along with links to the location where each is declared in the source code.



### 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.

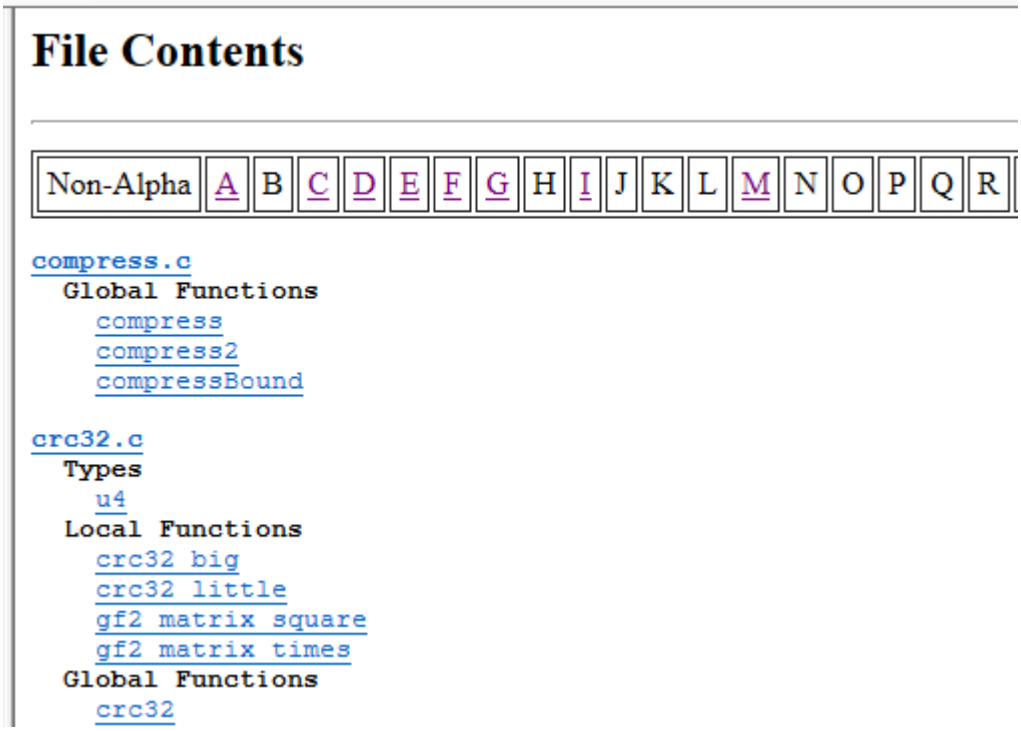
The HTML version offers hyperlinks to the Data Dictionary report entry and to the source code where each reference occurs.




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.

```
a     (Parameter)  Declared as: unsigned
    Define    [gzappend.c, 100]       gcd
    Use     [gzappend.c, 104]         gcd
    Modify    [gzappend.c, 109]       gcd
    Return    [gzappend.c, 117]       gcd

access::have     (Public Object)  Declared as: :
    Define    [zran.c, 71]            access
    Set     [zran.c, 102]             addpoint
    Use     [zran.c, 106]             addpoint
    Modify    [zran.c, 125]           addpoint
```

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.

```
gz_header     (Typedef)
  Declared as: struct gz_header_s
    Define    [zlib.h, 124]           zlib.h
    Type    [zlib.h, 126]             gz_headerp

gz_header_s     (Struct)
    Define    [zlib.h, 109]           zlib.h
    Type    [zlib.h, 109]             gz_header
```

### Class and Interface Cross-Reference

The Class and Interface Cross-Reference Report lists all declared classes and interfaces in alphabetic order, along with their declaration and usage information. The HTML version of the report includes hyperlinks to the data dictionary report entries, as well as the source code where the reference occurs.

```
Error     (Unknown Class)
  Declared as:
    Create    [REMatch.java, 62]      REMatch.clone
    Create    [Grep.java, 273]        Grep.processStream

Event     (Unknown Class)
  Declared as:
    Typed    [REApplet.java, 198]     e

Exception     (Unknown Class)
  Declared as:
    Extend    [REException.java, 38]  regexp.REException
    Typed    [Grep.java, 236]         ex
```

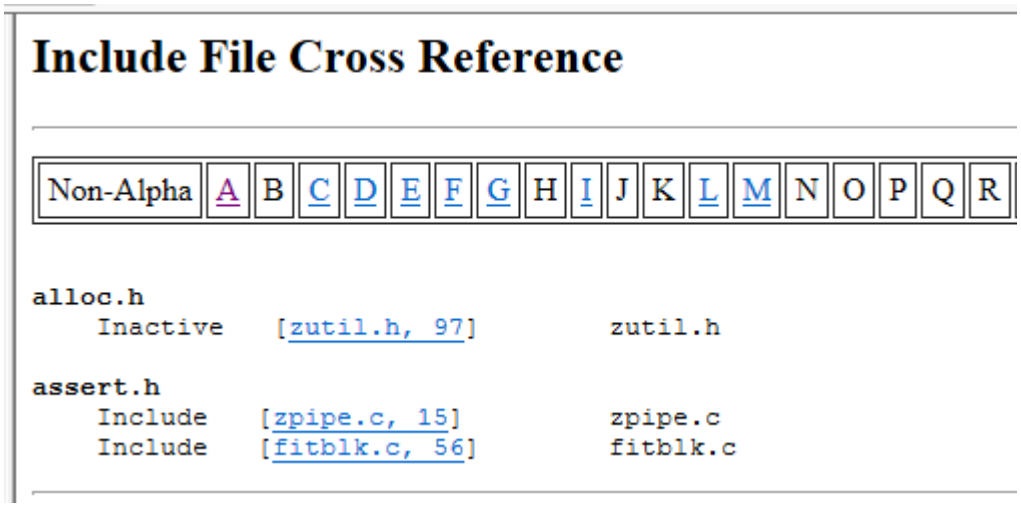
### Macro Cross-Reference

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

```
PIECE
  Declared as: 16384
    Define    [gun.c, 73]            gun.c
    Use     [gun.c, 96]              in
    Use     [gun.c, 138]             out
```

### Include File Cross-Reference

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



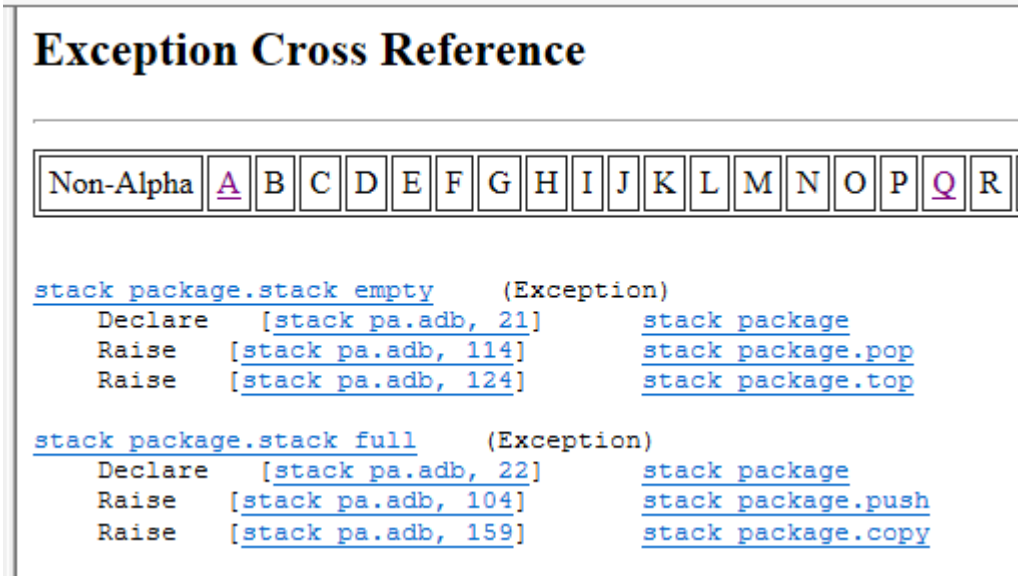
**Include File Cross Reference**

Non-Alpha | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R

```
alloc.h
    Inactive    [zutil.h, 97]        zutil.h

assert.h
    Include    [zpipe.c, 15]         zpipe.c
    Include    [fitblk.c, 56]        fitblk.c
```

### Exception Cross-Reference Report

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



Exception Cross Reference

Non-Alpha | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R

stack package.stack empty (Exception)
Declare [stack pa.adb, 21] stack package
Raise [stack pa.adb, 114] stack package.pop
Raise [stack pa.adb, 124] stack package.top

stack package.stack full (Exception)
Declare [stack pa.adb, 22] stack package
Raise [stack pa.adb, 104] stack package.push
Raise [stack pa.adb, 159] stack package.copy

# 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.



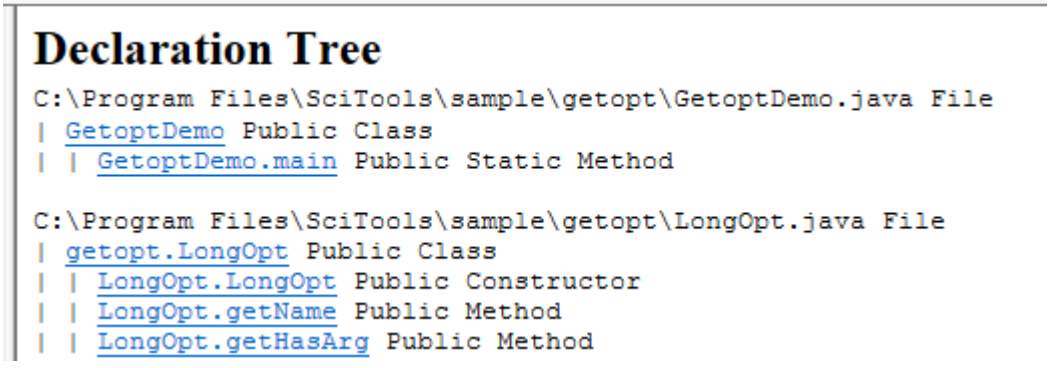
## Declaration Tree

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.



**Declaration Tree**

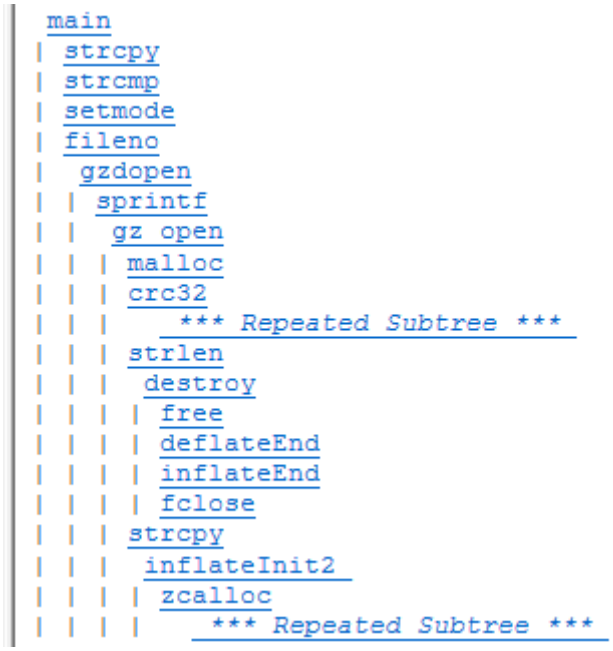
```
C:\Program Files\SciTools\sample\getopt\GetoptDemo.java File
| GetoptDemo Public Class
| | GetoptDemo.main Public Static Method

C:\Program Files\SciTools\sample\getopt\LongOpt.java File
| getopt.LongOpt Public Class
| | LongOpt.LongOpt Public Constructor
| | LongOpt.getName Public Method
| | LongOpt.getHasArg Public 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.




**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”.

**With Tree Report**

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

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

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

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

**Simple With Tree Report**

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

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

**Generic Instantiation Report**

This report lists each package that was created through instantiation.

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

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

**Renames Report**

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

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

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

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

**Import Report**

The *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 |
|---|
| *Program Unit Complexity Report* on page 222 |
| *Fortran Extension Usage Report* on page 223 |
| *Implicitly Declared Objects Report* on page 223 |
| *Uninitialized Items* on page 224 |
| *Unused Objects and Functions* on page 224 |
| *Unused Objects Report* on page 224 |
| *Unused Types Report* on page 225 |
| *Unused Program Units Report* on page 225 |
| *Uses Not Needed Report* on page 225 |
| *Withs Not Needed Report* on page 225 |

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.

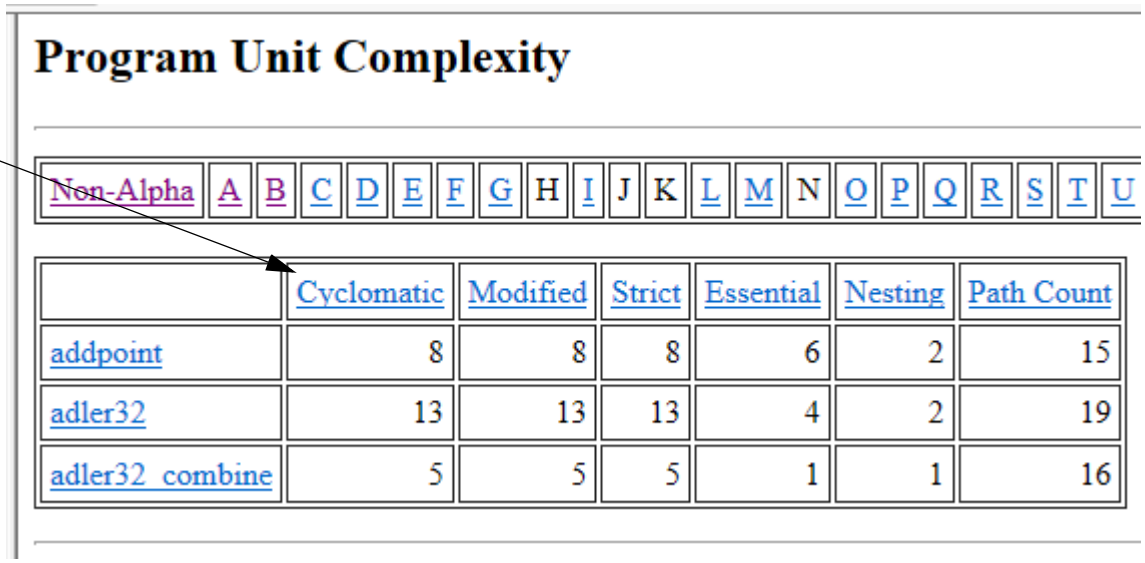
The Cyclomatic complexity is the number of independent paths through a module. The higher this metric the more likely a program unit is to be difficult to test and maintain without error.

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



Program Unit Complexity

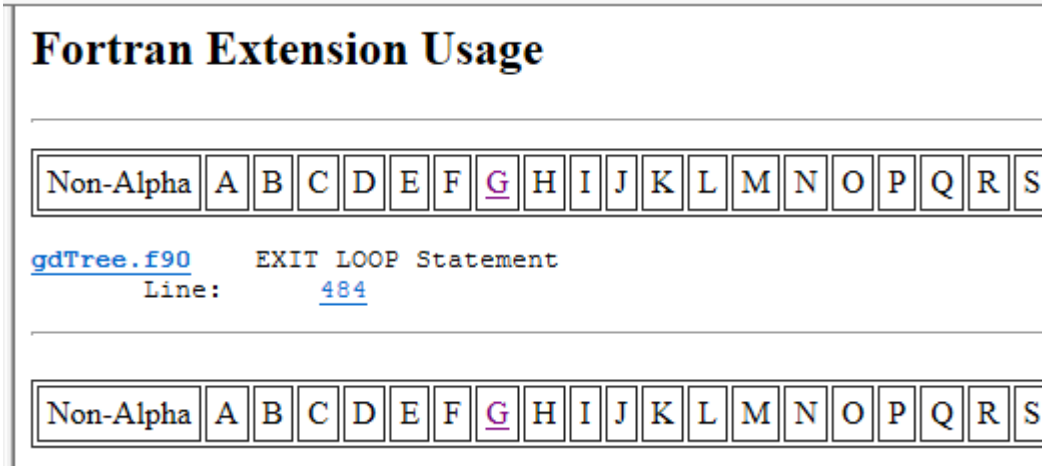
Non-Alpha | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U

| | Cyclomatic | Modified | Strict | Essential | Nesting | Path Count |
|---|---|---|---|---|---|---|
| addpoint | 8 | 8 | 8 | 6 | 2 | 15 |
| adler32 | 13 | 13 | 13 | 4 | 2 | 19 |
| adler32_combine | 5 | 5 | 5 | 1 | 1 | 16 |

**Fortran Extension Usage Report**

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

The following is a snippet from a sample Fortran Extension Usage report:



Fortran Extension Usage

Non-Alpha | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S

gdTree.f90 EXIT LOOP Statement
Line: 484

Non-Alpha | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S

**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**

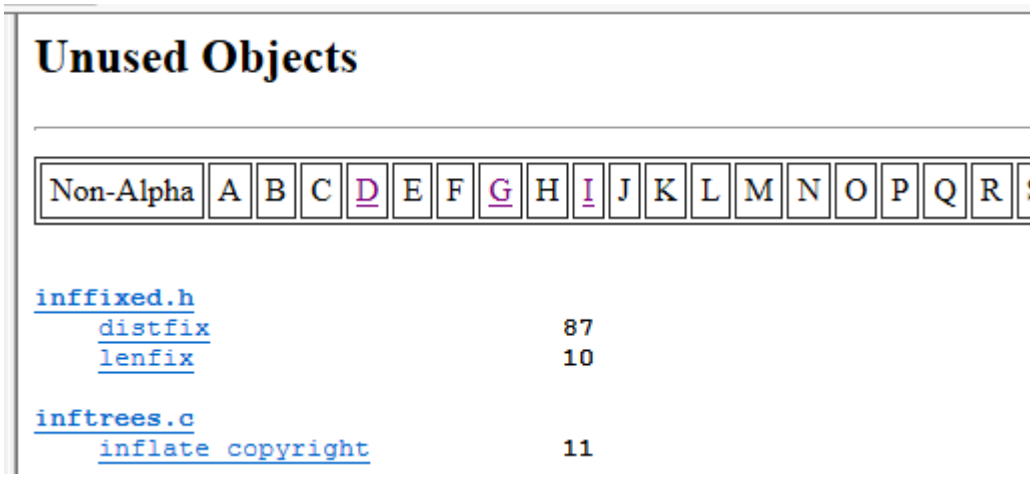
The *Uninitialized Items* report lists items such as variables that are not initialized in the code. The report is organized by file. Each uninitialized item within the file is listed by name along with the line number on which the item is declared. The HTML version offers hyperlinks to the location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

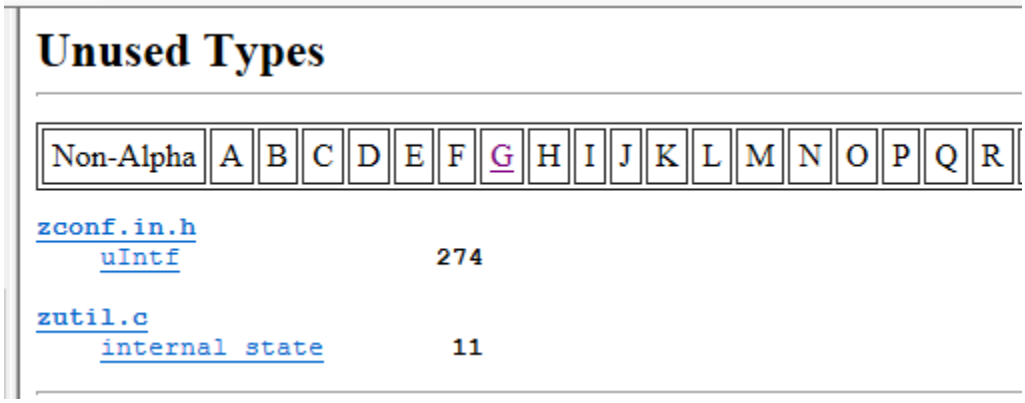
**Unused Objects Report**

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



**Unused Types Report**

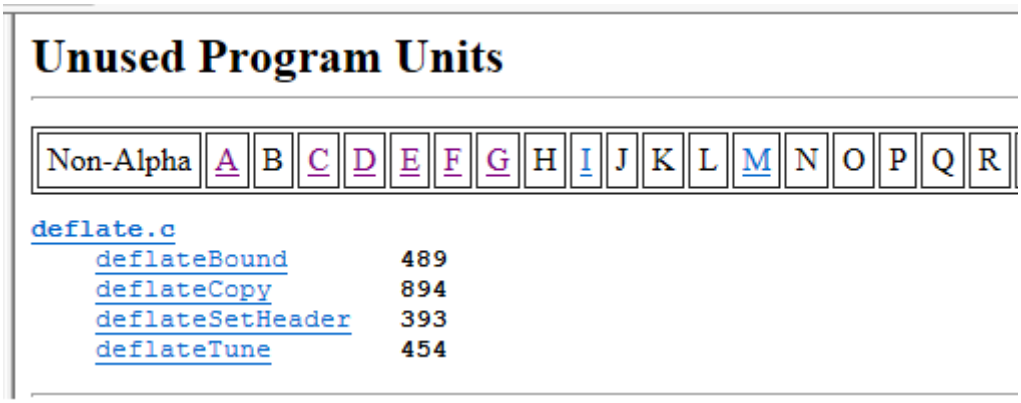
The *Unused Types Report* lists types that are declared but never used. The HTML version has links to the function's Data Dictionary report entry and the source where the type is declared.




**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.




**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 231 for other ways to gather metrics.

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

| Report Name and Page |
|---|
| *Project Metrics Report* on page 226 |
| *Class Metrics Report* on page 227 |
| *Class OO Metrics Report* on page 227 |
| *Program Unit Metrics Report* on page 228 |
| *File Metrics Report* on page 228 |
| *File Average Metrics Report* on page 229 |

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 include: the total number of files, the total number of program units, and the total number of lines of source code.

**Project Metrics**

| | |
|---|---|
| **Classes**: | 0 |
| **Files**: | 33 |
| **Program Units**: | 173 |
| **Lines**: | 14941 |
| **Lines Blank**: | 1344 |
| **Lines Code**: | 7217 |
| **Lines Comment**: | 4426 |
| **Lines Inactive**: | 1623 |
| **Executable Statements**: | 5152 |
| **Declarative Statements:**: | 1074 |
| **Ratio Comment/Code**: | 0.61 |

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

**Class Metrics Report**

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

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

## Class Metrics

| Class | Lines | Lines Blank | Lines Code | Lines Comment | Average Lines | Average Lines Comment | Average Complexity | Maximum Complexity | Ratio Comment/Code |
|---|---|---|---|---|---|---|---|---|---|
| GetoptDemo | 85 | 2 | 70 | 15 | 79 | 14 | 16 | 16 | 0.21% |
| gnu.getopt.Getopt | 960 | 125 | 506 | 337 | 43 | 5 | 6 | 45 | 0.67% |
| gnu.getopt.LongOpt | 153 | 25 | 48 | 81 | 6 | 0 | 1 | 2 | 1.69% |

**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.

## Program Unit Metrics

| | Lines | Comments | Blanks | Code | Lines-exe | Lines-dec | Stmt-exe | Stmt-dec | Ratio Comment/Code |
|---|---|---|---|---|---|---|---|---|---|
| uncompress | 36 | 1 | 6 | 29 | 19 | 7 | 22 | 2 | 0.03 |
| updatewindow | 49 | 3 | 4 | 42 | 29 | 6 | 29 | 2 | 0.07 |

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 Metrics

| | Lines | Comments | Blanks | Code | Lines-exe | Lines-dec | Stmt-exe | Stmt-dec | Ratio Comment/Code | Units |
|---|---|---|---|---|---|---|---|---|---|---|
| deflate.c | 1736 | 456 | 181 | 916 | 617 | 181 | 598 | 109 | 0.50 | 22 |
| deflate.h | 331 | 170 | 61 | 88 | 0 | 88 | 0 | 81 | 1.93 | 0 |

## 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.

# Chapter 9 Using Metrics

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

This chapter contains the following sections:

# About Metrics

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

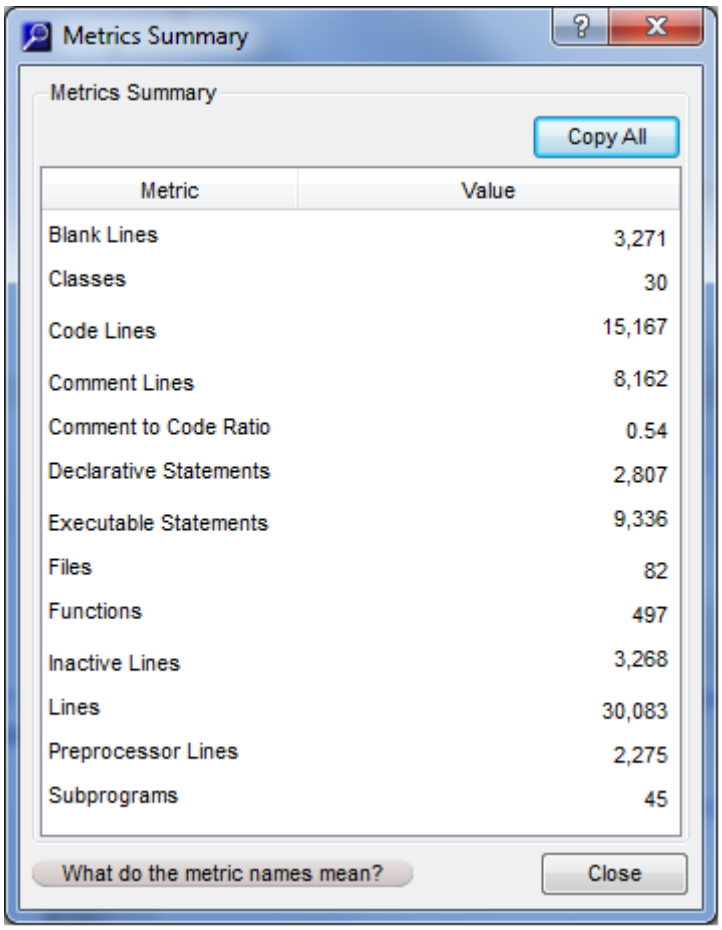
- **Information Browser:** The Information Browser tree has a **Metrics** node. You can expand this branch to show a few metrics for the current entity. See page 135.
- **Metrics Summary:** You can choose **Metrics > Metrics Summary** from the menus to see a short list of metrics for the entire project. See page 232.
- **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 233.
- **Export to HTML:** You can click this button in the Project Metrics Browser to export the full list of metrics for all architecture nodes and files. See page 234.
- **Export to CSV:** You can choose **Metrics > Export Metrics** from the menus to create a text file of all the project metrics in comma-delimited format. See page 235. (You can schedule this export to occur regularly; see page 50.)
- **Configure Metric Charts:** You can choose **Metrics > Configure Metric Charts** from the menus to open a dialog that lets you display graphs of volume and complexity metrics on an architecture basis. See page 237.
- **Reports:** When you create reports by choosing **Project > Project Reports**, some of the reports provide metrics. See page 226.
- **PERL/C API:** A more advanced way to get existing metrics and calculate new metrics is with the PERL and C API. These provide full access to the *Understand* database. Choose **Help > PERL API Documentation** for more information. See page 213.

*Understand* provides a large number of metrics you can generate about your code. The complete list of metrics available in *Understand* changes frequently—more frequently than this manual is reprinted.

A complete and accurate list of metrics is always available on our website at scitools.com/support/metrics_list/. The “What do the metric names mean?” buttons in metrics dialogs links to this page.

# Metrics Summary

Choose **Metrics > Metrics Summary** from the menus to see a short list of metrics for the entire project.

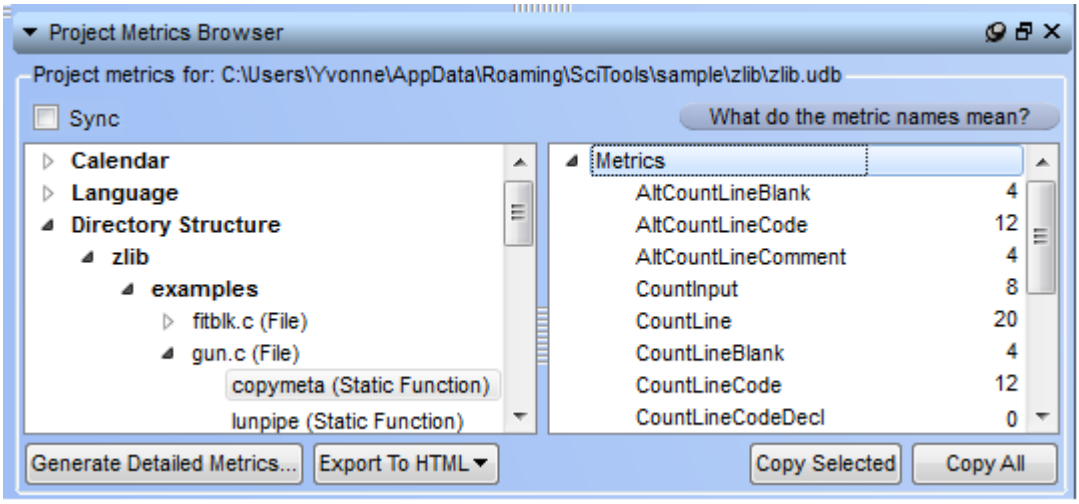



If your cursor hovers over a metric, you will see a short description of that metric. A complete and accurate list of metrics is always available on our website at scitools.com/support/metrics_list/. The “What do the metric names mean?” buttons in metrics dialogs link to this page.

# Metrics Browser

To open the Project Metrics Browser, choose **Metrics > Browse Metrics** from the menus. You can also open the Metrics Browser with a particular architecture node, file, or entity selected by right-clicking on the name of an item in *Understand* and selecting **Browse Metrics**.

You can browse architectures in your project and select any architecture node, file, or entity. The list on the right shows code size and complexity metrics for the selected item. If your cursor hovers over a metric, you will see a short description of that metric. For a full list of metrics with descriptions, see scitools.com/support/metrics_list/. The "What do the metric names mean?" button links to that page.

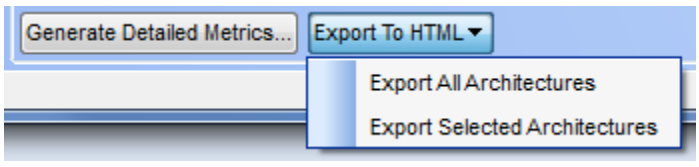



If you check the **Sync** checkbox, clicking on an entity elsewhere in *Understand* causes the Metrics Browser to display the metrics for that entity.

Double-click a file or entity to open the Source Editor for that item. Right-click to see the standard informational menu choices.

You can select rows on the right and click **Copy Selected** or press Ctrl+C to copy those lines to the clipboard. Click **Copy All** to copy the full list of metrics for the selected directory or file.

Click **Export to HTML** to generate reports as described on page 234. You can choose to **Export All Architectures** and all of their nodes or to **Export Selected Architectures** and nodes that you have selected using the Ctrl or Shift key while clicking on the nodes you want in the metrics export. Click **Generate Detailed Metrics** to open the Export Metrics dialog and generate a text file in comma-delimited format as described on page 235.



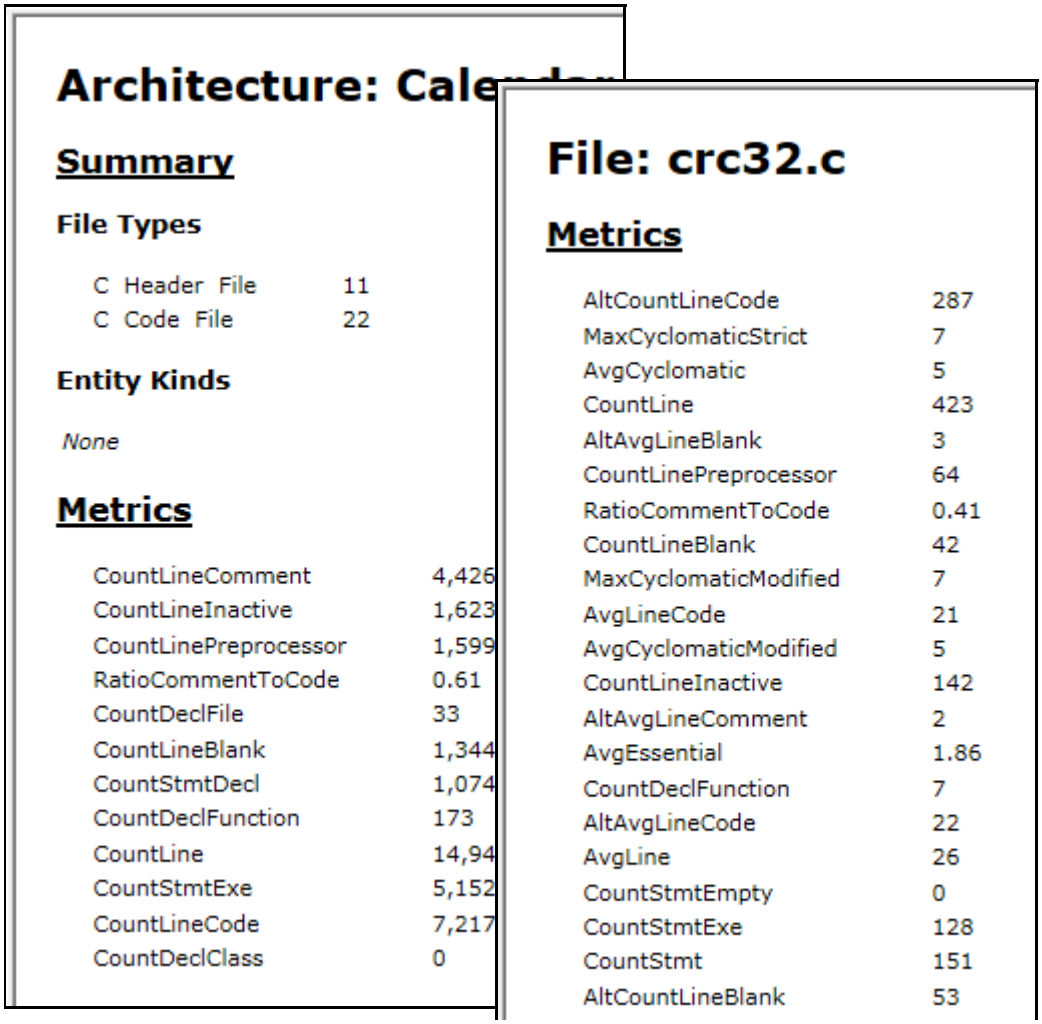
# Exporting Metrics to HTML

You can click the **Export to HTML** button in the Project Metrics Browser (page 233) to export the full list of metrics for all entities and architecture nodes. When you click this button, you see a Browse for Folder dialog appropriate to your operating system.

Choose or create the folder where you want the metrics files to be created. The files are actually stored in a folder called “pixie_proj_Metrics” below the folder you select.

If the directory already exists, you are asked if the files should be overwritten. If you answer “No”, a number is appended to the old directory name to it to save it as a backup.

If the report is generated successfully, you are asked if you want to view the report. Click **Yes** to open the top-level page, index.html.



Architecture: Cale

Summary

File Types

| | |
|---|---|
| C Header File | 11 |
| C Code File | 22 |

Entity Kinds

*None*

Metrics

| | |
|---|---|
| CountLineComment | 4,426 |
| CountLineInactive | 1,623 |
| CountLinePreprocessor | 1,599 |
| RatioCommentToCode | 0.61 |
| CountDeclFile | 33 |
| CountLineBlank | 1,344 |
| CountStmtDecl | 1,074 |
| CountDeclFunction | 173 |
| CountLine | 14,94 |
| CountStmtExe | 5,152 |
| CountLineCode | 7,217 |
| CountDeclClass | 0 |

File: crc32.c

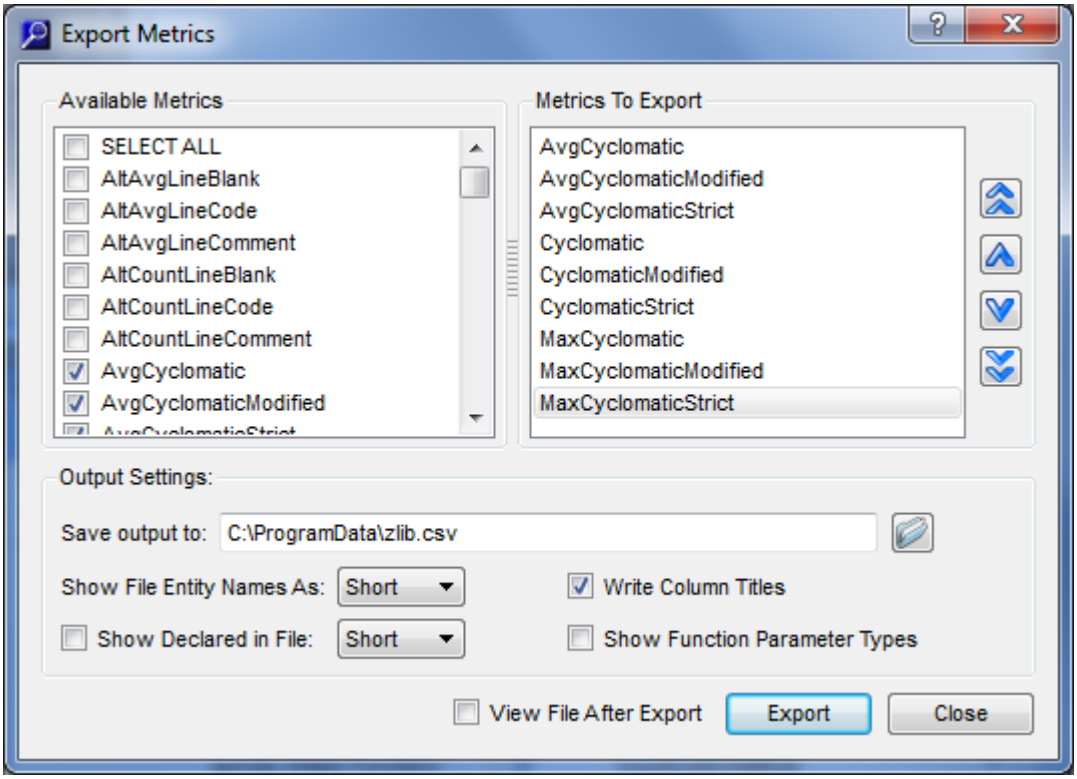
Metrics

| | |
|---|---|
| AltCountLineCode | 287 |
| MaxCyclomaticStrict | 7 |
| AvgCyclomatic | 5 |
| CountLine | 423 |
| AltAvgLineBlank | 3 |
| CountLinePreprocessor | 64 |
| RatioCommentToCode | 0.41 |
| CountLineBlank | 42 |
| MaxCyclomaticModified | 7 |
| AvgLineCode | 21 |
| AvgCyclomaticModified | 5 |
| CountLineInactive | 142 |
| AltAvgLineComment | 2 |
| AvgEssential | 1.86 |
| CountDeclFunction | 7 |
| AltAvgLineCode | 22 |
| AvgLine | 26 |
| CountStmtEmpty | 0 |
| CountStmtExe | 128 |
| CountStmt | 151 |
| AltCountLineBlank | 53 |

The HTML-based report lets you select any architecture node, file, function, or other entity type that has metrics in the left pane. The right pane shows metrics available for that item.

# Exporting Metrics to a CSV File

You can save metric information to a comma-delimited text file by choosing **Metrics > Export Metrics** from the menus or clicking the **Generate Detailed Metrics** button in the Project Metrics Browser. You can use the generated file in Excel and other spreadsheet programs. The Export Metrics dialog looks like this:




The defaults in this dialog come from the Project Configuration dialog in the **Metrics > Options** category (page 52) and the **Metrics > Selected** category (page 53).

You can override the defaults using the following fields:

- **Available Metrics:** Check the boxes next to metrics you want to include in the output. Check the “SELECT ALL” box to select all metrics. Uncheck the “SELECT ALL” box to unselect all metrics. 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.

After setting options, click **Export** to export the .CSV file. If you check the **View File After Export** box before exporting the file, the CSV file is opened with the default application for working with CSV files. This is likely to be a spreadsheet application.

If the output file already exists, you are asked if the files should be overwritten. If you answer “No”, you can change the output filename and click **Export** again.

You can schedule this metrics to be automatically exported to a CSV file on a regular basis. See page 50 for details.

See *Exporting Dependency Metrics* on page 242 for more types of metrics you can export to a CSV file.

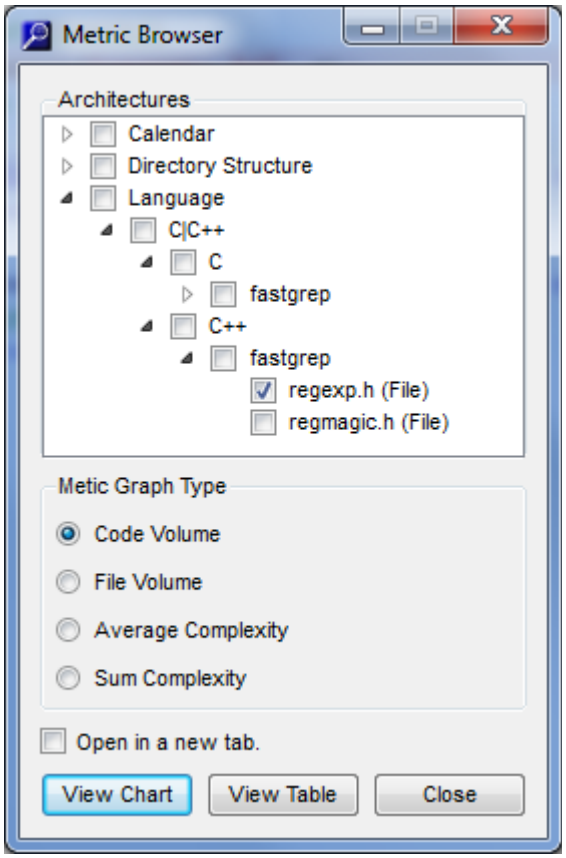
A complete and accurate list of the available metrics is available at: scitools.com/support/metrics_list/.

# Configuring Metric Charts

Commands in the Metrics menu provide fast access to metrics charts for the current version of the entire project. These commands are:

- **Metrics > Project Metric Charts > Code Volume**
- **Metrics > Project Metric Charts > File Volume**
- **Metrics > Project Metric Charts > Average Complexity**
- **Metrics > Project Metric Charts > Sum Complexity**

You can choose **Metrics > Configure Metric Charts** from the menus to open the Metric Browser, which lets you display graphs of various metrics on an architecture basis.

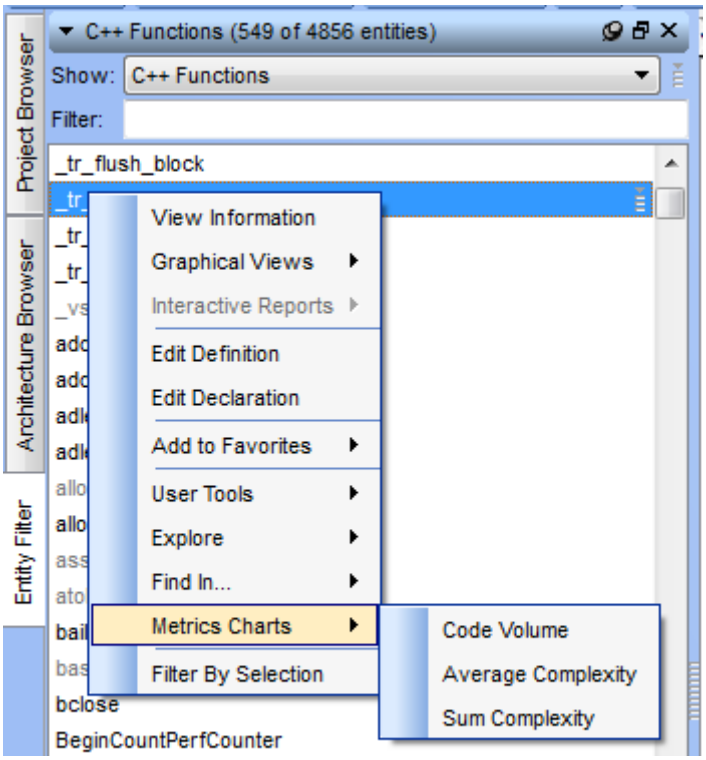



In this browser, select the following:

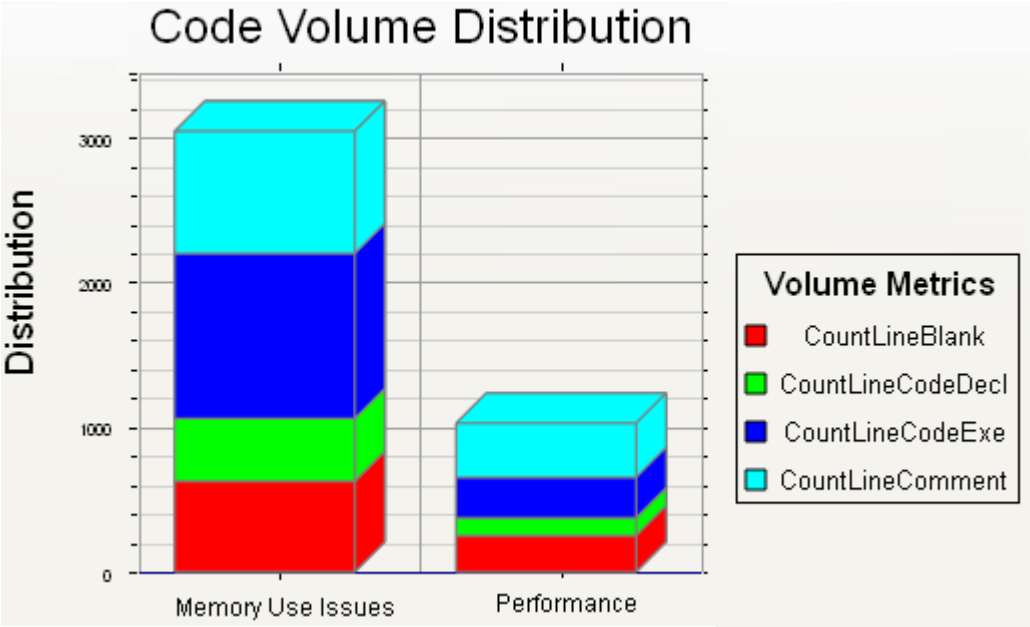
- **Architectures:** Checkboxes next to one or more architecture nodes, files, and/or entities. The graph will provide a set of vertical bars for each of the items you select.
- **Metrics:** Select the type of metrics you want to graph from the drop-down list.
  - **Code Volume:** Provides a stacked vertical bar chart showing the count of lines that are blank, contain declarations and executable code, and contain comments.
  - **File Volume:** Provides a vertical bar chart showing the number of files in the selected architecture node that are code files vs. header files (or the number of files for languages that do not have header files).
  - **Average Complexity:** Provides a vertical bar chart of the average and maximum cyclomatic complexity for all nested functions or methods in the architecture node, along with the maximum nesting level of control constructs in the node's files.
  - **Sum Complexity:** Provides a vertical bar chart showing the number of possible paths through the code and the sum of the cyclomatic complexity and the essential complexity of all nested functions or methods.

Click the **View Chart** button to display a chart for your selections or **View Table** to see the values in a table. If you have already selected a graph for this type of metrics, that tab will be reused unless you check the **Open in a new tab** box.

You can open metrics charts for various entities, including files and functions, from the context menus throughout *Understand*.




A typical metrics chart looks similar to the following:

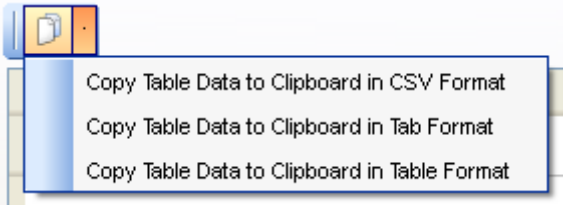


In a graph, you can choose the **Graph View** or the **Table View**. Both views have a toolbar that lets you save the graph or data.

In the **Graph View**, you can use the toolbar to:

- Save the image as a PNG, JPEG, or BMP image.
- Copy the image to the clipboard.
- Print the image using the standard Print dialog.

In the **Table View**, the numeric values for each pie slice or vertical bar is shown in a table. You can use the toolbar to copy the data to the clipboard in comma-separated (CSV), tab-separated, or table format (spaces used so columns align with headings if a font such as Courier is used).

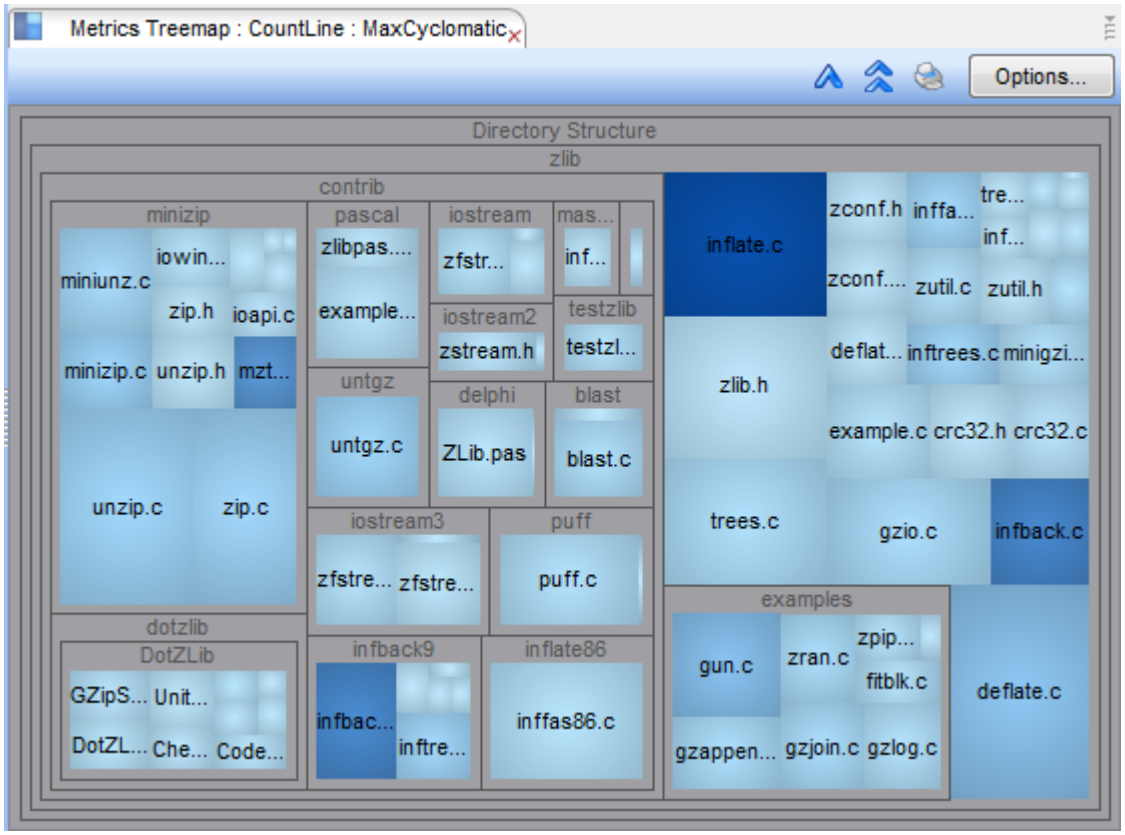



If you are viewing data for several architecture nodes, you can change the number in the lower-right corner to the number of vertical bars you want to view on each page and click the checkmark icon. Then use the arrows in the corners to move from page to page. The text shows which vertical bars are currently shown out of the total number. For example, the figure below indicates that bars 3 and 4 out of a total of 5 are currently shown.

# Using the Metrics Treemap

Treemaps show metrics graphically by varying the size of node blocks and the color gradient. Each node block represents a code file. Different metrics can be tied to size and color to help you visualize aspects of the code.

For example, the following treemap ties the number of lines in each file to the size of the block and the MaxCyclomatic complexity metric to the darkness of the blue. This allows you to learn which files are large and complex vs. files that are large and relatively non-complex.

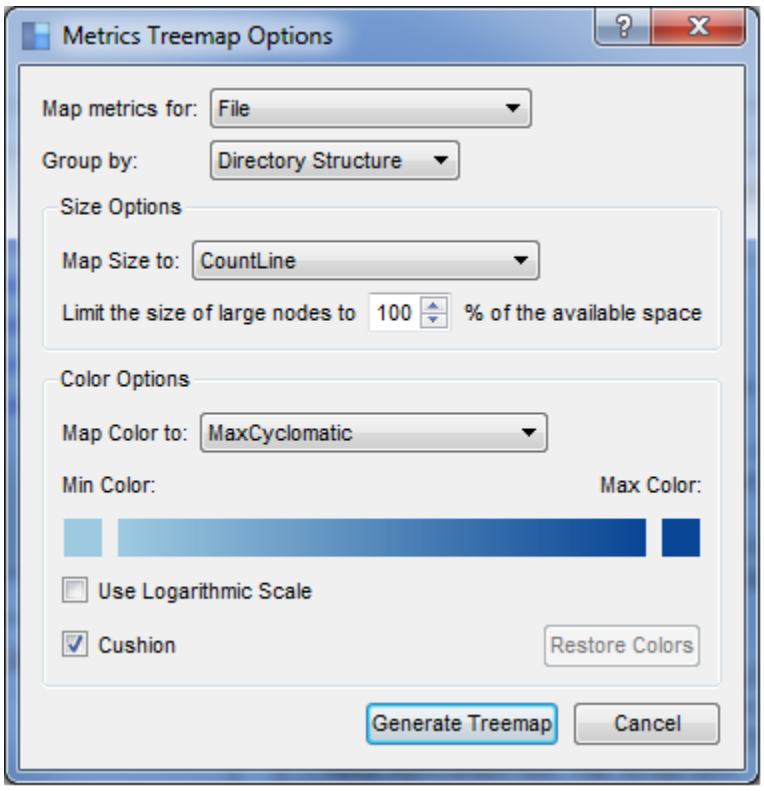



So we learn that unzip.c is large, but not particularly complex, while inflate.c is large and highly complex.

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

To open the treemap for your project, follow these steps:

1 Choose **Metrics > Metrics Treemap** from the menus. You will see the Metrics Treemap Options dialog.




2 In the **Map metrics for** field, choose whether you want to select from metrics for Files, Classes/Interfaces/Structs, or Functions/Methods.

3 In the **Group by** field, choose how to group blocks in the treemap. The default is to group by the project's directory structure. Alternately, you can choose to group according to any other defined architecture or no architecture (flat).

4 In the **Size Options** area, choose a metric to control the size of the blocks. You can also limit the size of the largest blocks to some percentage of the treemap. You might want to use this if one node is taking up so much of the map that you can't see differences between the smaller nodes.

5 In the **Color Options** area, choose a metric to control block colors. You can click the left color square to set a color for blocks with the lowest value for this metric; click the right color square to set a color for blocks with the highest value for this metric.

6 Check the **Use Logarithmic Scale** box if you want the color scaled by powers of 10 of the selected metric value. This is useful for treemaps with extreme value ranges.

7 Uncheck the **Cushion** box if you want to see solid colors in the blocks. By default, the blocks have a gradient fill.

8 Click **Generate Treemap** to display the treemap. You can return to the Options dialog by clicking **Options** in the upper-right corner of the treemap.

Within the treemap, when your mouse cursor hovers over a block, the two metric values chosen for the size and color are shown.

You can double-click on an architecture node (shown as a gray border around a set of colored blocks) to display only the contents of that node. You can also zoom in by right-clicking on a node and choosing **Drill down** from the context menu.

After drilling down in the architecture, you can use the  icons to **Pop up one level** or **Pop up all levels** the treemap. You can also right-click to use the **Pop up one level** and **Pop up all levels** commands in the context menu.

The **Print** icon lets you print the treemap.

# Exporting Dependency Metrics

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

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

The options available are as follows:

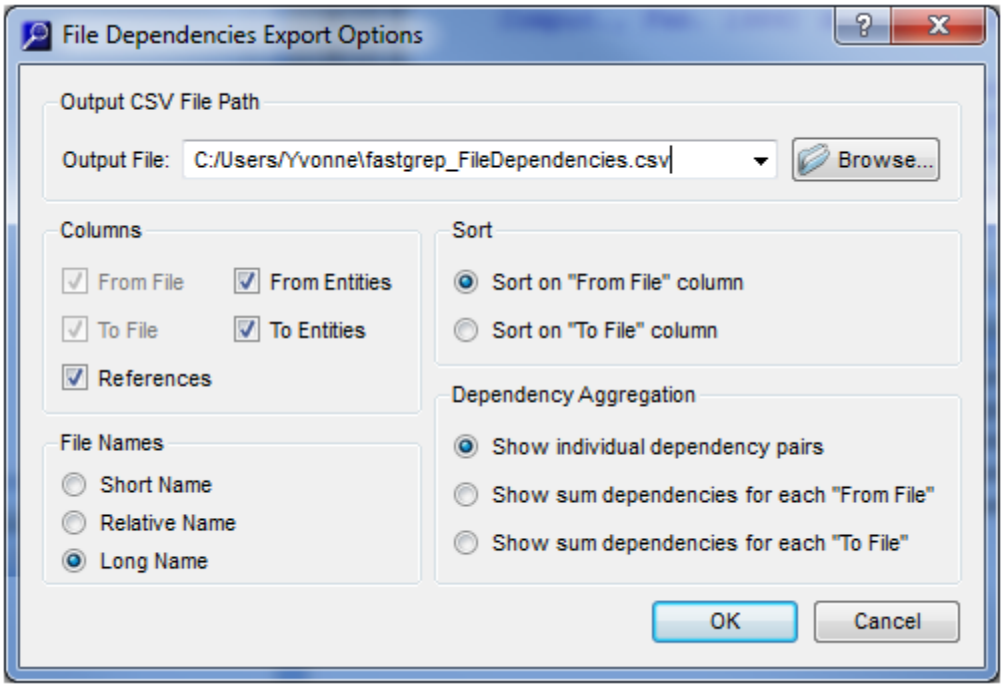
- **Architecture Dependencies >**
  - **Export CSV:** This output lists pairs of architecture nodes for which the node in column A is dependent upon the node in column B. The number of dependencies for each pair is listed in column C. (See page 243.)
  - **Export Matrix CSV:** This output lists all architecture 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 244.)
  - **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 244.)
  - **Export Dot:** This output format uses a plain text graph description language. Various programs are available that can import DOT files to create graphs, perform calculations, and manipulate the data.
- **File Dependencies >**
  - **Export CSV:** This output lists pairs of files for which the file in column A is dependent upon the file in column B. The number of dependencies for each pair is listed in column C. (See page 243.)
  - **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 244.)

  - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 244.)

- **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 243.)
  - **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 244.)
  - **Export Cytoscape XML:** See the description of the Cytoscape XML export for architecture dependencies. (See page 244.)

## 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.)



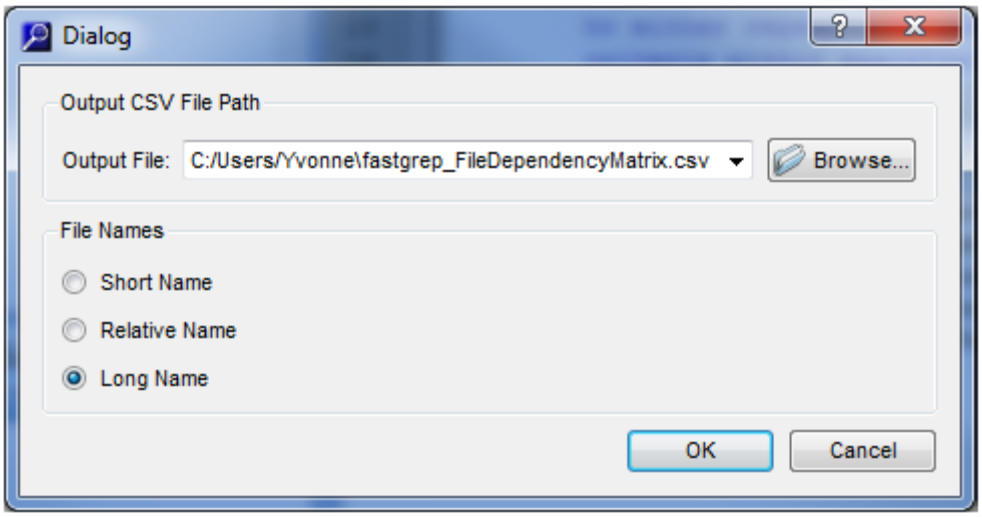

In this dialog, you can set the following options:

- **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 108 for how to configure the location of the Cytoscape installation.

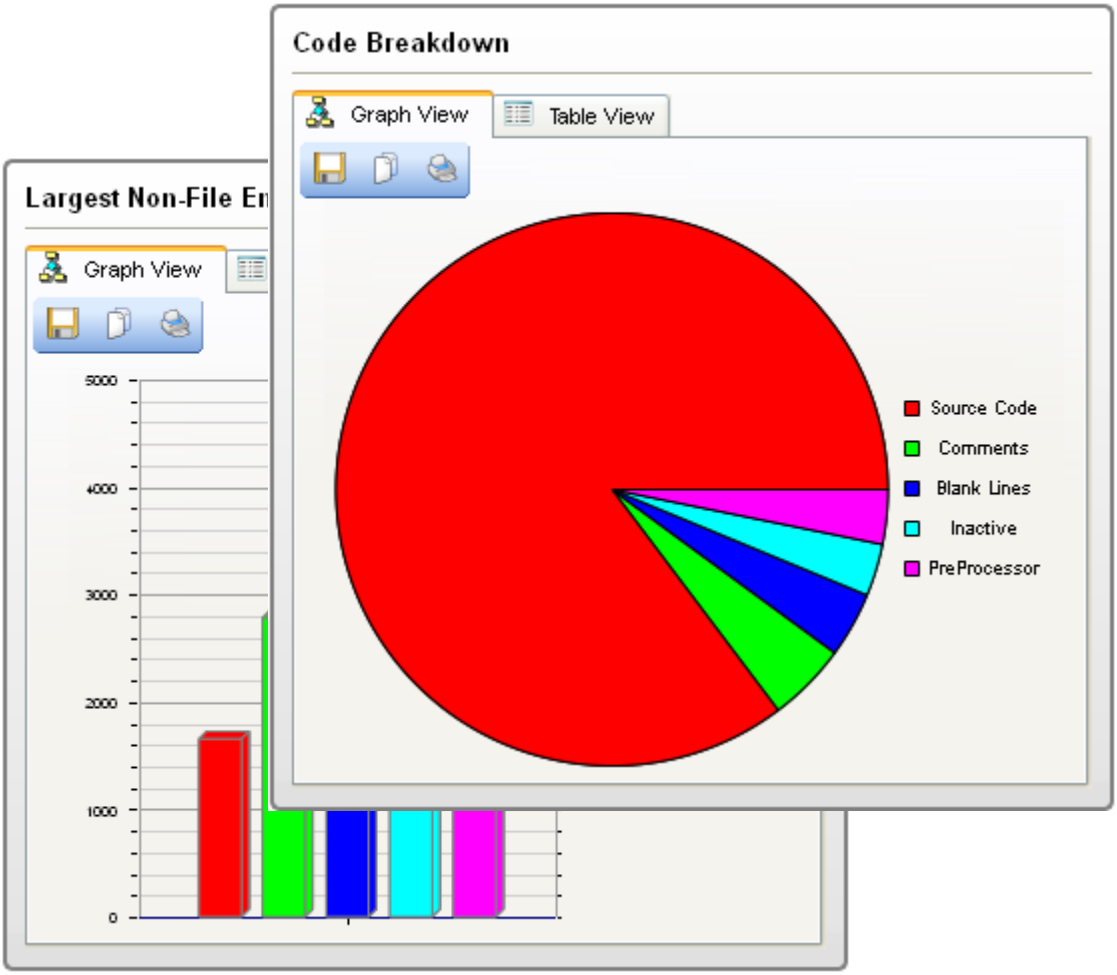
# Chapter 10 Using Graphical Views

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

This chapter contains the following sections:

# Project Overview Graphics

You can create graphics that provide an overview of your entire project by choosing **Project > Project Overview Charts** from the menus. This opens a tab in the document area that contains a number of pie charts and vertical bar charts. For example:




The graphs provided are code breakdown (line types), function breakdown, class breakdown, most complex functions, largest non-file entities, largest files, largest functions, best comment-to-code ratio entities, and most complex files.

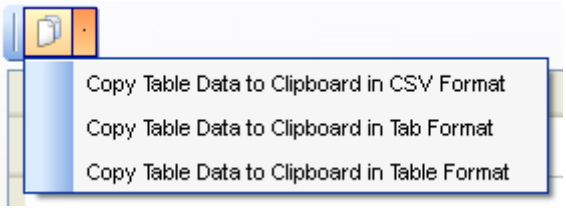
In each area, you can choose the Graph View or the Table View. Both views have a toolbar that lets you save the graph or data.

In the Graph View, you can use the toolbar to:

- Save the image as a PNG, JPEG, or BMP image.
- Copy the image to the clipboard.

- Print the image using the standard Print dialog.
- Zoom in on the graph in a new tab.

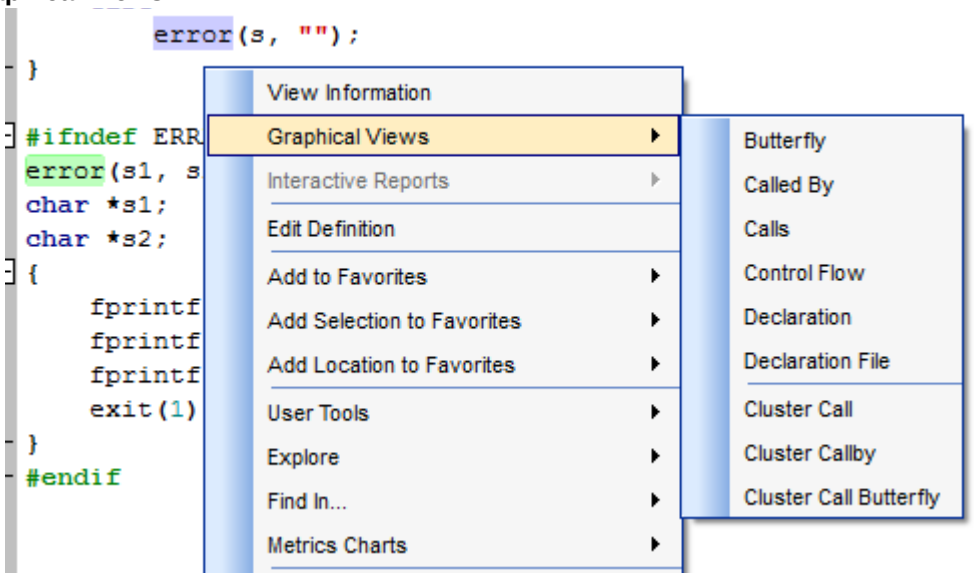
In the Table view, the numeric values for each pie slice or vertical bar is shown in a table. You can use the toolbar to copy the data to the clipboard in comma-separated (CSV), tab-separated, or table format (spaces used so columns align with headings if a font such as Courier is used).



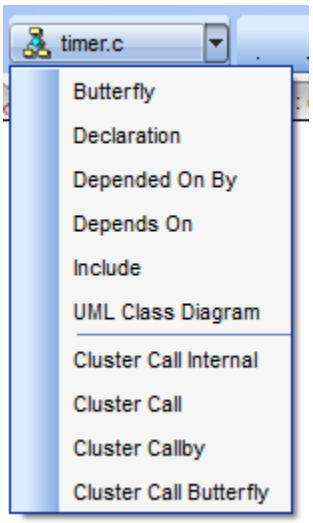

In addition to the Project Overview Charts, you can display metrics graphs that provide additional statistical information about your project or portions of your project. For details, see page 237.

# Graphical View Browsers

The context menu of an entity that has a structure or hierarchy offers a choice called **Graphical Views**:




You can also use the Graphs drop-down menu in the toolbar to select from the types of graphs available for the entity at the current cursor position in a Source Editor tab. The same list of graphical views is available by choosing **Graphs > Graphs for <*current_entity*>** from the menus.



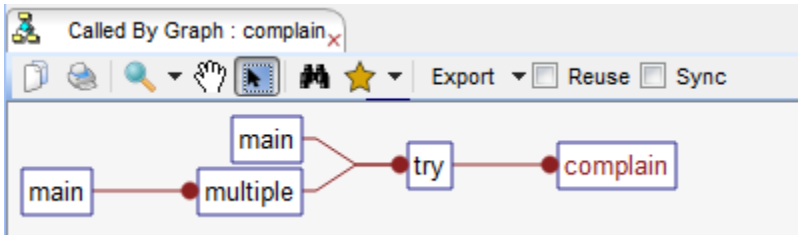

The **Graphical Views** menu adapts based on the kind of entity right-clicked. An item is grayed-out if information is normally available for this kind of entity but is not applicable to this particular entity (for instance a package that could be WITHed but isn't).

There are two main types of graphical views in these menus: hierarchy views and structure views.

(Dependency graphs are a separate type of graph described in the chapter on architectures. See page 196 for details. Project overview graphs are described on page 246.)

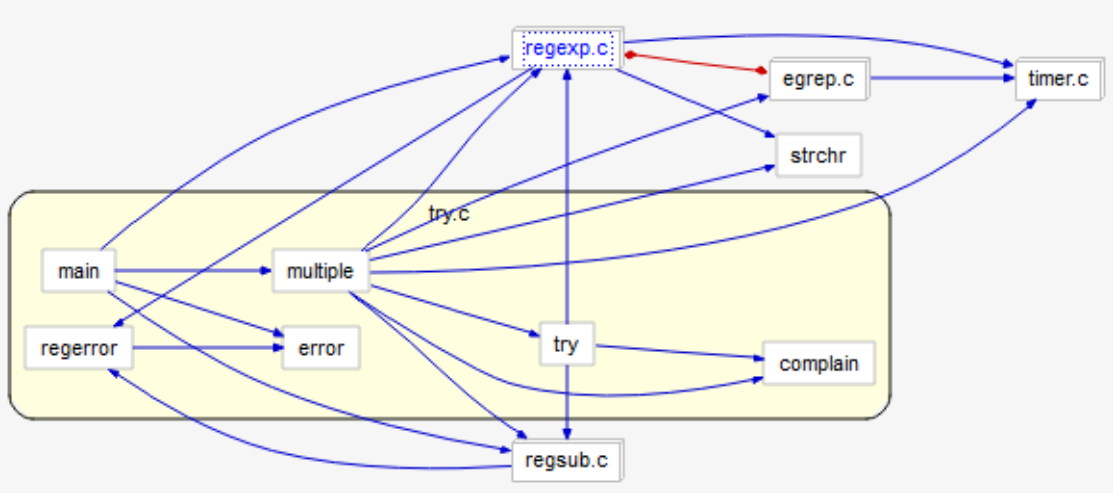
## Hierarchy Views

A *hierarchy view* shows multiple level relationships between entities. All relationships are multi-level and are shown to the top or bottom of their respective tree unless a level option is set in the preferences. The following is a Call By graph for a function.



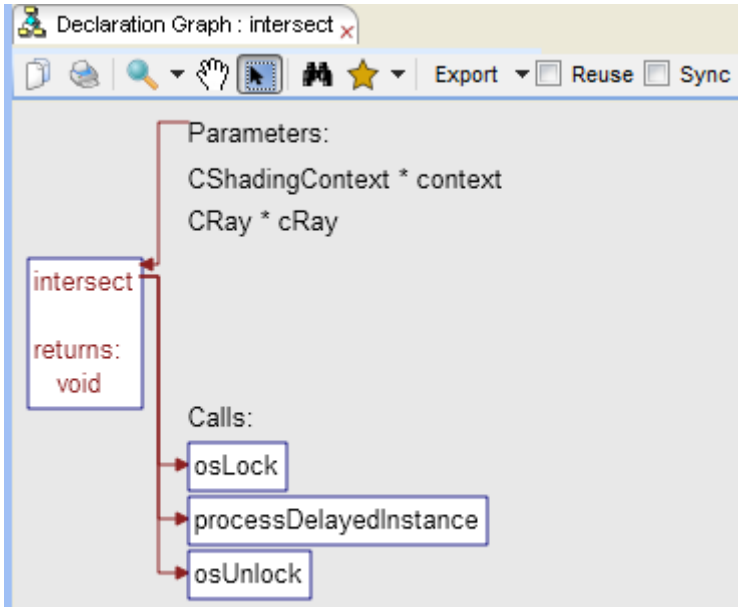

See *Hierarchy View Types* on page 254 and *Hierarchy View Examples* on page 255.

Cluster views are a special type of hierarchy view. They provide a more interactive view of call relationships. The Call, Callby, Butterfly and Internal Call variants are available, and can be accessed from the function, class, file, or architecture level. See *Controlling Cluster Graph Layout* on page 272.



## Structure Views

Structure views offer a one glance way to see important structure and relational information about a given entity. The following is an example of a Declaration structure view:




See *Structure View Types* on page 258 and *Structure View Examples* on page 258.

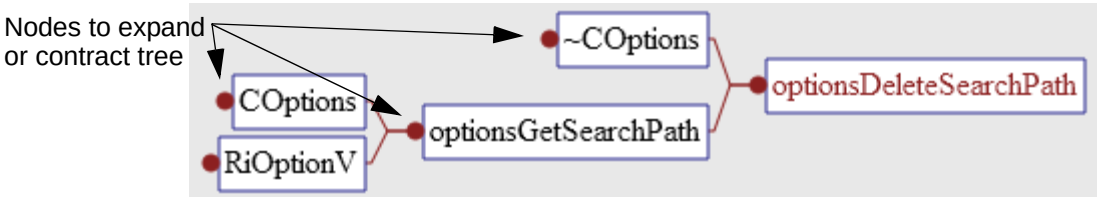
## General Commands for Using Graphical Browsers

There are some general commands that can be used for browsing graphical views. Note that some of these tools are not available in all types of graphs.

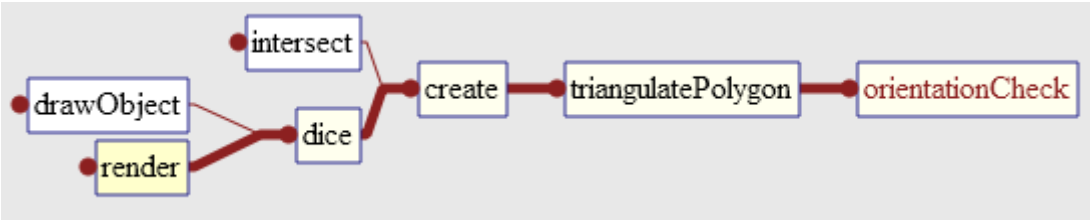
- **Entity info:** Anywhere you see an entity, you can right-click on it to see a menu that offers many ways to learn more about that entity. Single-clicking shows information about the entity in the Information Browser. If you are in Screen Drag mode or Zoom mode, click the  icon to be able to select entities.
- **Searching:** Click the  **Search** icon at the top of a graphical view or press Ctrl+F to display the incremental search bar. You can use this bar that same way you use it in the Source Editor to find entities by name or other text in the current graphical view. As you type search text, all instances of the string are highlighted in the graphical view. See page 163 for details.

- **Opening source:** Right-click on an entity in a graphical view and choose **Edit Definition** to open the source location where the entity is declared.
- **Listing open views:** You can choose **Window > Windows** from the menus or look at the tabs across the top of the document area to see a list of all the separate graphical views you have open.
- **Scrolling:** You can scroll around a graphical view by dragging your cursor within the view when you have selected Screen Drag Mode by clicking the  icon.

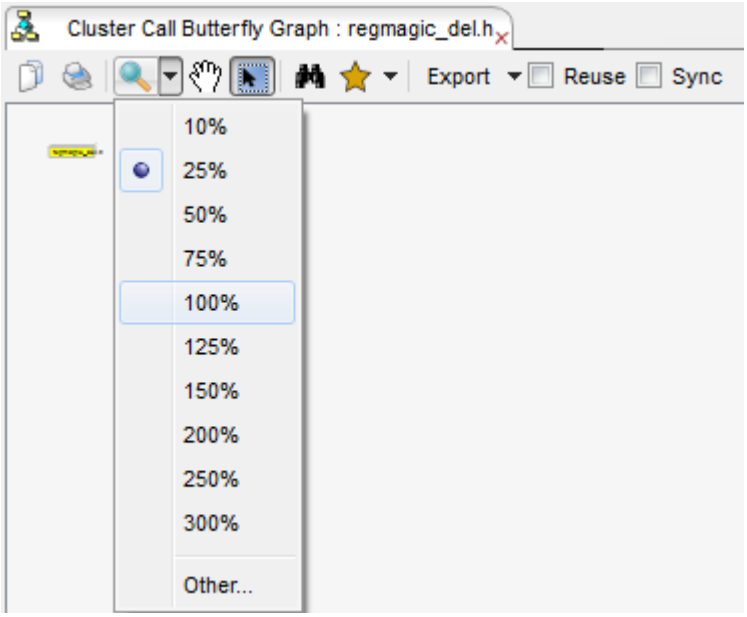
- **Expanding hierarchy:** You can expand and contract tree views by clicking the red circle to the right of a node. Right-click on the background of a view and choose **Open All Nodes** or **Close All Nodes** to expand or contract all nodes at once. Choose **Close Unselected Nodes** to contract nodes that are not in the path of any selected nodes. Hold down the Ctrl key if you want to keep multiple nodes open.




- **Path highlighting:** To highlight the path for one or more entities in a tree view (such as a Callby view), select the entity and right-click. In the context menu, choose **Highlight Path**. Hold down the Ctrl key to select multiple entities for path highlighting.




- **Add to favorites:** You can add graphs you may want to reopen to a favorites list by clicking the  Favorites icon in the toolbar.
- **Zooming:** You can zoom in or out using the toolbar.




- **Printing and saving:** Everything you see can be printed or saved. Printing may be done to one page (squeezing the picture) or across multiple pages (poster style).

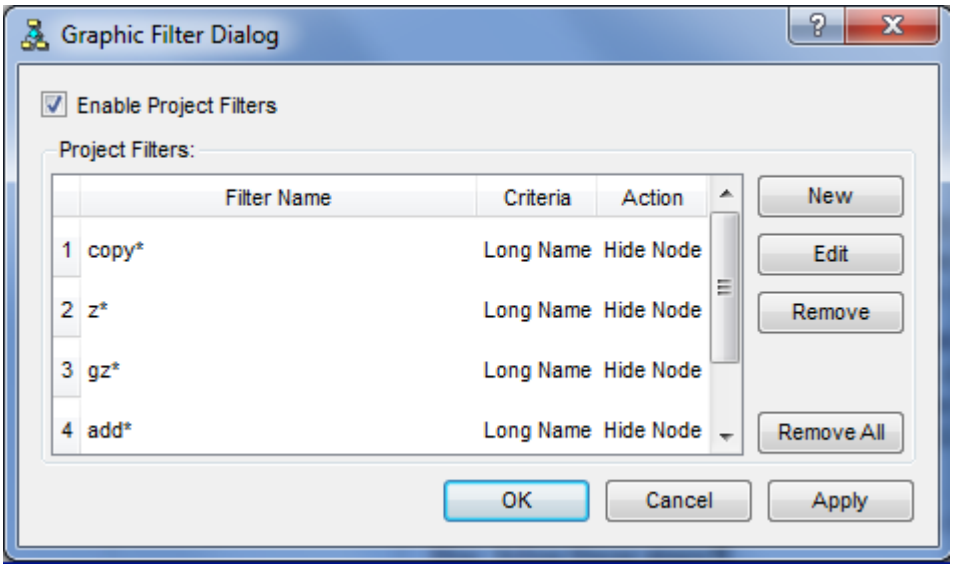
See *Printing Graphical Views* on page 278 for details on printing. Graphical views can be saved as BMP, JPEG, PNG, Visio XML, and DOT files. See *Saving Graphical Views* on page 275 for details on saving to a graph file.

- **Layout control:** Layout is done automatically; there is no need to move lines or boxes around for a better view. Options are available for changing the layout. For example, you can control whether entities are sorted according to their order in the code or alphabetically. See *Controlling Graphical View Layout* on page 263.

## Filtering Out Entities

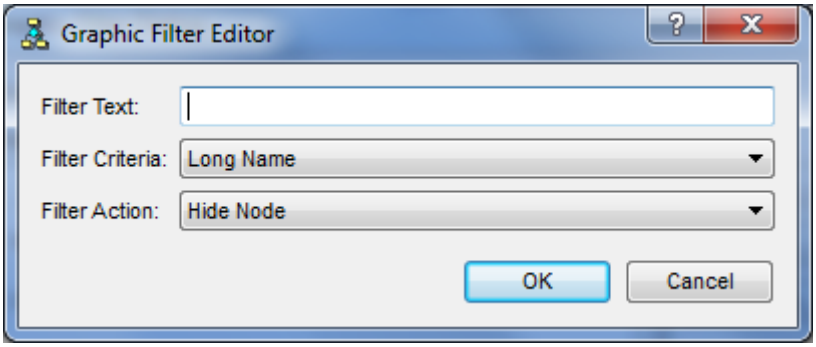
You can apply filters to hide certain entities in some graphical views. To create such a filter, follow these steps:

1 Right-click on the background of a graphical view and choose **Edit Graphic Filters** from the context menu. (Note that this option is not available for some types of graphs. For example, it is available for Call graphs and Declaration graphs.)




2 In the Graphic Filter dialog, put a checkmark in the **Enable Project Filters** box.

3 Click **New**. This opens the Graphic Filter Editor dialog.




4 Type a filter in the **Filter Text** field. For example, use gr* to match entity names beginning with gr. Filters are case-sensitive.

5 In the **Filter Criteria** field, select whether to compare the filter to long names, definition files, or the type text of entities. For example, if you choose long names, a filter of print* does not match SomeProc::printWide. Instead, you can type *print*.

6 In the Action field, select one of the following options:

- **Hide Node:** Items that match the filter are not included in the output.
- **Hide Sub Nodes:** The item that matches the filter is shown, but any subnodes of these items are removed from the output.
- **Collapse Sub Nodes:** Any subnodes of items that match the filter are collapsed in the output. An icon is shown after the node to indicate that there are subnodes. Items that match the filter are shown.

7 Click **OK** to add the filter to the project.

You can also create filters by right-clicking on an entity in a graphical view and choosing one of the filtering options. The options allow you to quickly filter out entities with that name or in that file.

You can remove filters you have created by clicking **Remove** or **Remove All**.

The filters you create apply to all graphical views that support filtering. You can temporarily disable filtering in the Graphical Settings dialog or by right-clicking on any graphical view and choosing **Disable Graphic Filters** from the context menu.

**Reuse Checkbox**

The **Reuse** checkbox controls whether a view is reused or a new window is opened when another graphical view is requested. The Reuse box is unchecked by default. At most one graphical view can have the **Reuse** box checked at any time.

You can cause views to be reused if a similar type of graphical view is opened from within a graphical view, no matter whether the **Reuse** box is checked. Change this behavior in the User Interface > Windows category of the **Tools > Options** dialog (page 96).

**Sync Checkbox**

The **Sync** checkbox controls whether this graphical view changes when a different entity is selected in the Project Browser, Entity Filter, and other windows that let you select an entity. For example, if you check the Sync box in a Declaration graph window and then select a different entity in the Entity Filter, the graph shows declaration information for the newly selected entity.

**Graph Options**

See page 118 for information about the **Graphs** category in the **Tools > Options** dialog. You can control how the display of relationships between graph nodes changes when you hover the mouse over a graph or double-click on a node.

# Types of Views

There are two main types of graphical views: hierarchy views and structure views.

(Dependency graphs are a separate type of graph described in the chapter on architectures. See page 196 for details. Project overview graphs are described on page 246.)

## Hierarchy View Types

*Hierarchical* views show multi-level relationships between entities. *Understand* offers hierarchy graphs of the following types of relationships. Some types apply to specific source languages.

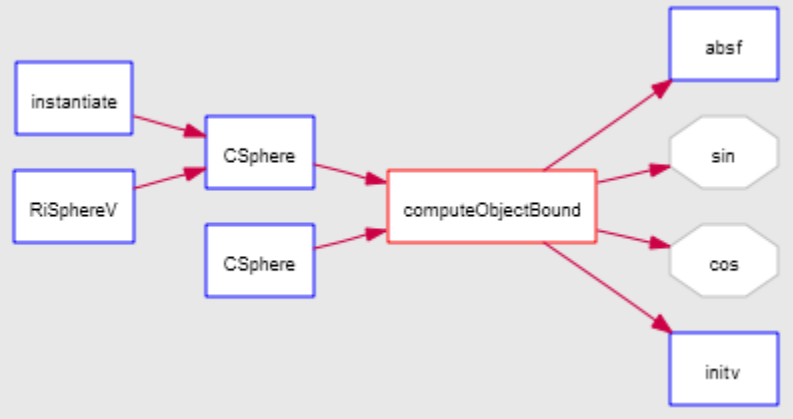
- **Butterfly:** Shows both calls and called by.
- **Calls:** Shows who this entity calls.
- **Calls Relationship:** Show the call relationships between two entities.
- **Called By:** Shows who calls a given entity.
- **Calledby Relationship:** Show the callby relationships between two entities.
- **Include:** Shows who this file includes.
- **IncludeBy:** Shows who includes this file.
- **Depends On Graph**, **Depended On By Graph**, and **Butterfly Graph:** Available for classes, packages, and architectures only. See page 196 for architecture graphs.
- **Derived Classes:** Shows classes derived from a given class.
- **Base Classes:** Show what classes are the base for a class.
- **Extended By:** Shows which classes are extended by this class.
- **Class Inheritance:** Shows who inherits from a given class.
- **Child Lib Units:** Shows Child Library Units of a compilation unit. (Ada 95 only)
- **Declared In:** Shows the declaration tree from where this program unit is declared.
- **Declaration Tree:** Shows declaration nesting of program units in a compilation unit.
- **Instantiated From:** Shows instantiation tree of a generic type or compilation unit.
- **Instantiations:** Shows who instantiates a given generic unit.
- **Invocation:** Shows what compilation units a unit invokes.
- **Parent Lib Unit:** Shows the parent lib units of a given entity.
- **Type Derived From:** Shows tree of types a type is derived from.
- **Type Tree:** Shows types that derive new types from an entity.
- **With:** Shows what compilation unit an entity “Withs” into scope.
- **WithBy:** Shows what compilation units “Withs” a given entity.
- **Uses:** Shows which modules use this item.
- **Used By:** Shows which modules are used by this item.
- **Cluster Call Internal:** Shows call relationships within a file.

- **Cluster Call:** Shows who this entity calls.
- **Cluster Callby:** Shows who calls this entity.
- **Cluster Call Butterfly:** Shows both calls and called by.

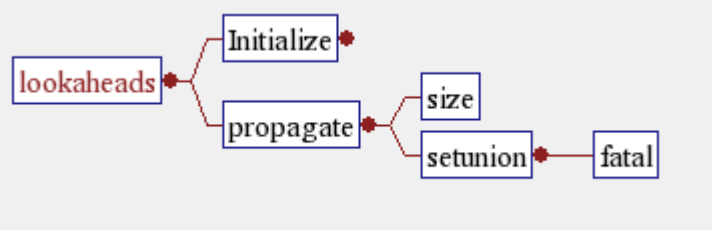
## Hierarchy View Examples

Hierarchy views show multi-level relationships between entities. Here are examples of the types of hierarchy views that *Understand* offers.

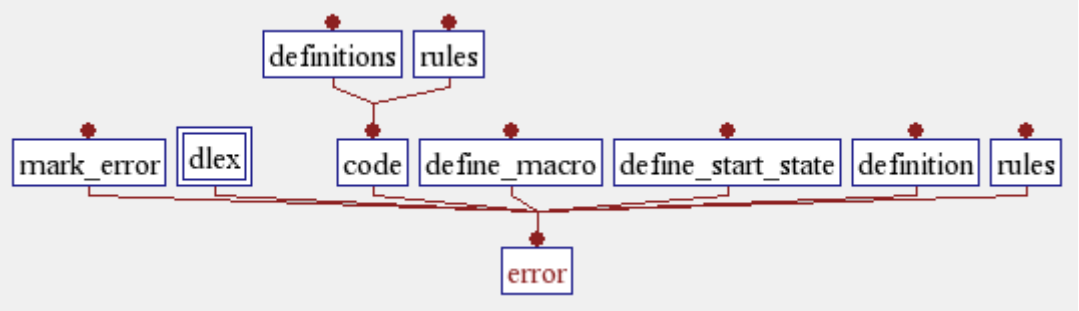
- **Butterfly:** Shows both calls and called by relationships if they exist. The selected entity is outlined in red.



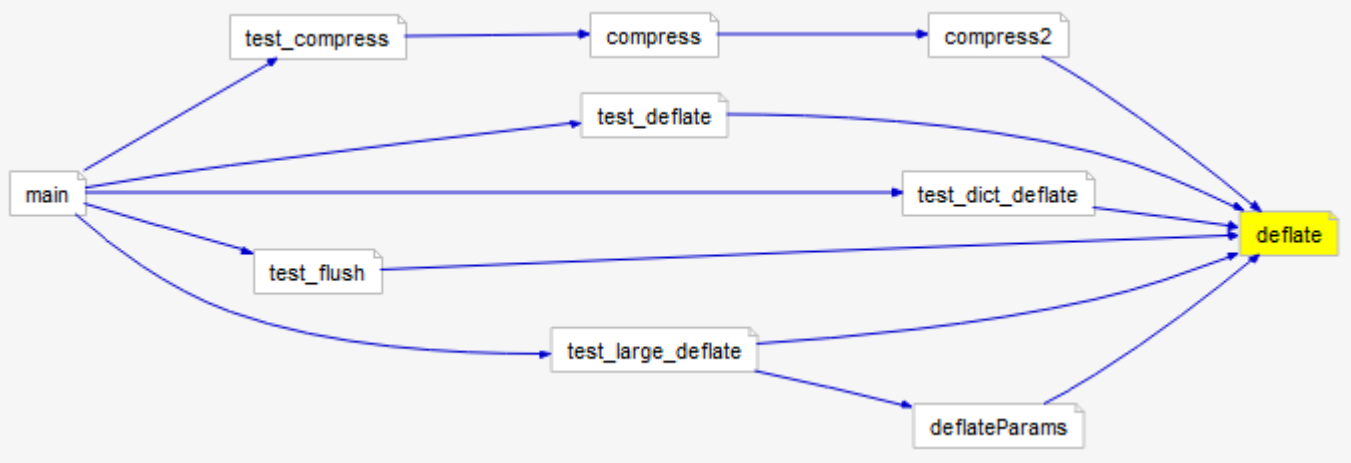

- **Calls:** Shows the entire chain of calls emanating from this function. Each line between entities is read as “x calls y”.



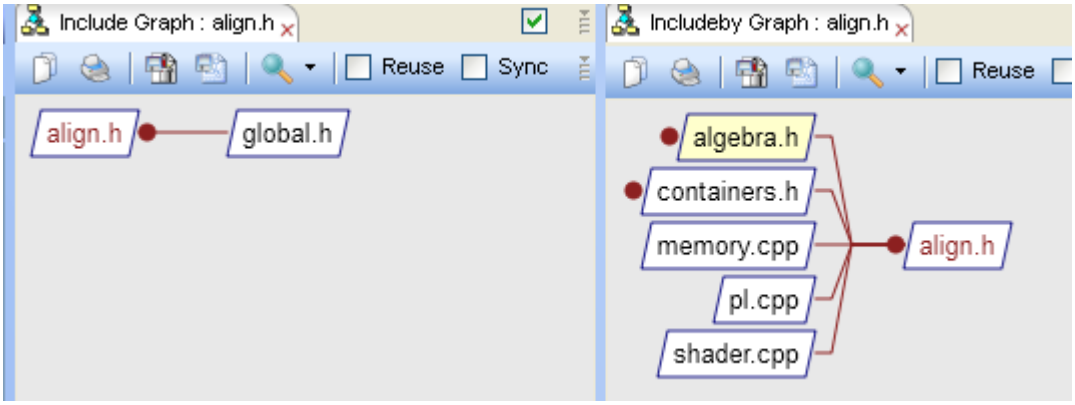

- **Called By:** Shows what calls an entity. Each line connecting an entity is read as “x is called by y”. In this example, error is called by code (and others), which is called by rules (and others). Note that this view is read from the bottom up or right to left.



- **Calls Relationship / Calledby Relationship:** Shows the call or callby relationships between any two entities. First, right-click on the first entity and select the graph you want to view. Then, click on another entity whose relationship to the first entity you want to find. You can click on the second entity anywhere in the *Understand* interface. The entity name will appear in the “Select a second entity” dialog. This example shows the callby relationship from the deflate() function to main().

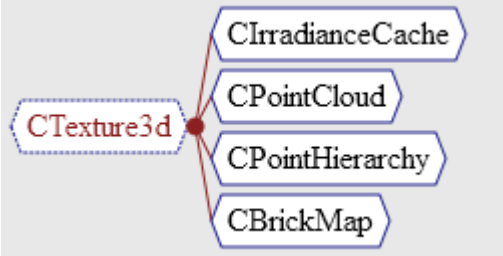



- **Include:** Shows the include hierarchy of an entity, such as a file. A connecting line is read as “x includes y.” In this example, align.h includes global.h.

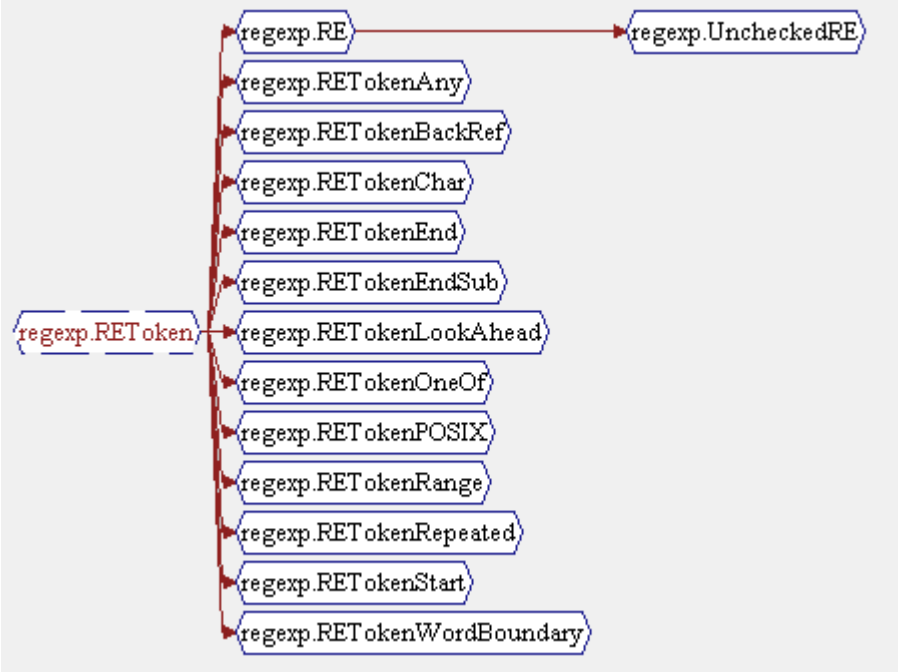



- **Include By:** Shows the include tree in the other direction. In the previous example, align.h is included by several files such as algebra.h.
- **Base Classes:** For classes, shows the base classes from which this class is derived from. In this example, class *CLInearCurve* is derived from class *CCurve*, which is derived from class *CSurface* and so on.

- **Derived Classes:** Shows the classes that are derived from this class. In this example, class *CTexture3d* is a base class for classes *CIrradianceCache* and others.




- **Extended By:** Shows which classes are extended by other classes. A line is read as “class is extended by class.” In this example, the *regexp.REToken* class is extended by a number of classes, including the *regexp.RE* class, which in turn is extended by the *regexp.UncheckedRE* class.



## Structure View Types

*Structure* views offer a one glance way to see important structure and relational information about a given entity. *Understand* structure views include the following:

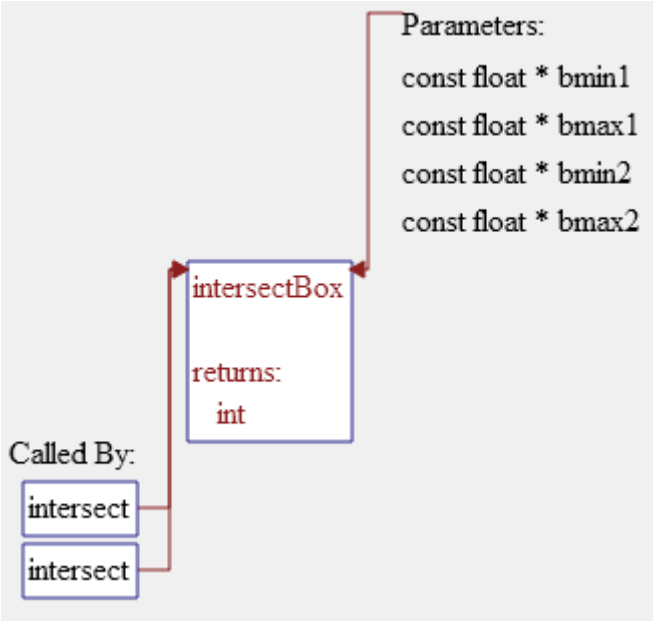
- **Graph Architecture:** Shows the hierarchy of an architecture node. See page 199.
- **Declaration:** Shows what a structure is composed of. For example, shows the parameters, return type, and callbys of a function. For classes, shows what members are provided, who inherits this class, and who it is based on.
- **Parent Declaration:** Shows what a structure is composed of. Shows Calls instead of the Called Bys shown by a Declaration graph.
- **Declaration File:** Shows what entities (such as functions, types, macros, and variables) are defined within a given file.
- **Declaration Type:** Shows what a type is composed of.
- **Class Declaration:** Shows the members defining the class and the parent class
- **Data Members:** Shows what components a class, struct, or type contains.
- **Control Flow:** Shows a flow chart of the function or similar entity type. Clicking on a node in the graphs jumps to the line of code referenced.
- **Cluster Control Flow:** Shows a flow chart of the function or similar entity type. This view type is more interactive than the Control Flow view.
- **UML Class Diagram:** Shows the classes defined in the project or a file and related classes. Adheres to the Unified Modeling Language (UML) structure diagram format.
- **UML Sequence Diagram:** Shows interactions between entities arranged by time sequence. This graph is available for functions and methods that call member methods. See the Scientific Toolworks website for a sample UML Sequence diagrams and information about how events are displayed.
- **Package:** Shows what entities are declared in a given package (body or spec).
- **Task:** Shows the parameters, invocations, and what entities/entry points are declared in a task. Also shows what the task Withs.
- **Rename Declaration:** Shows what entities are renamed in the entity.

## Structure View Examples

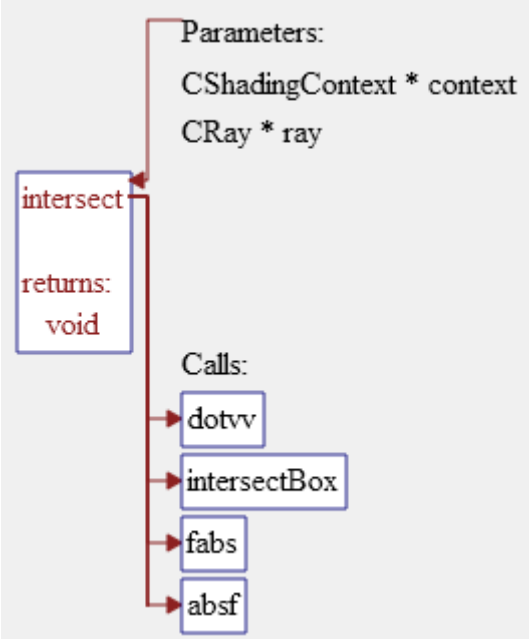
Structure views quickly show structure and relations.

*Understand* structure views are designed to present essential information about an entity in a small and concise manner. The structure diagram is derived from the graphs presented by Booch and Buhr in their respective books “Software Engineering with Ada” and “System Design in Ada.” Where needed, symbols and annotations have been extended or altered to represent new kinds of information available from *Understand*.

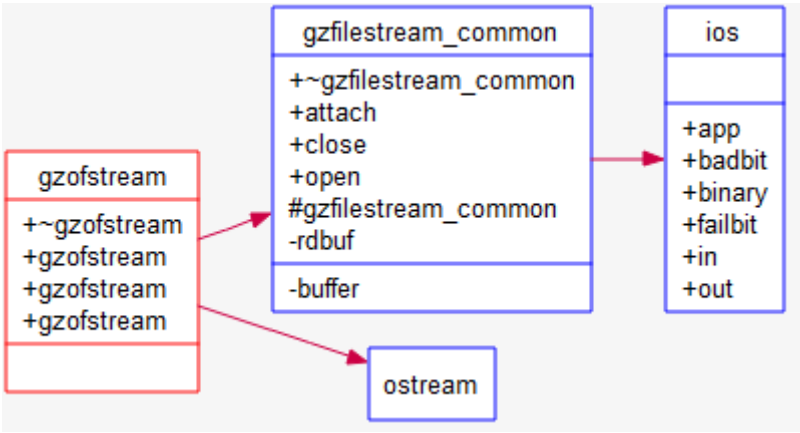
- **Declaration:** Shows the structure of the entity. For example, shows the parameters, return type, and callbys of a function.



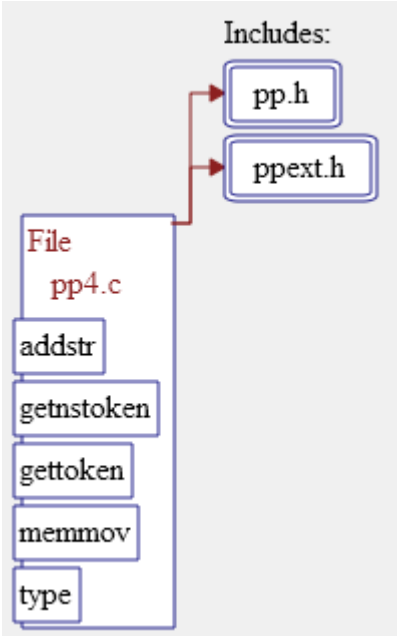

- **Parent Declaration:** Similar to a Declaration graph but shows what the entity calls.



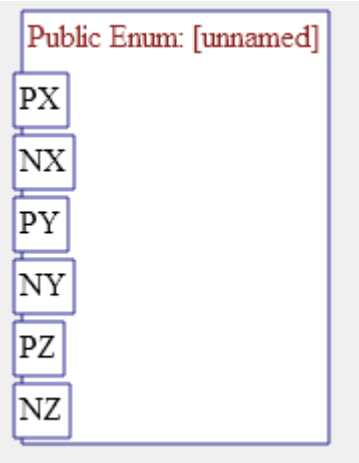
- **UML Class Diagram:** Shows the classes defined in the project or a file and related classes. Right-click to show or hide class details, related classes, and solo classes.



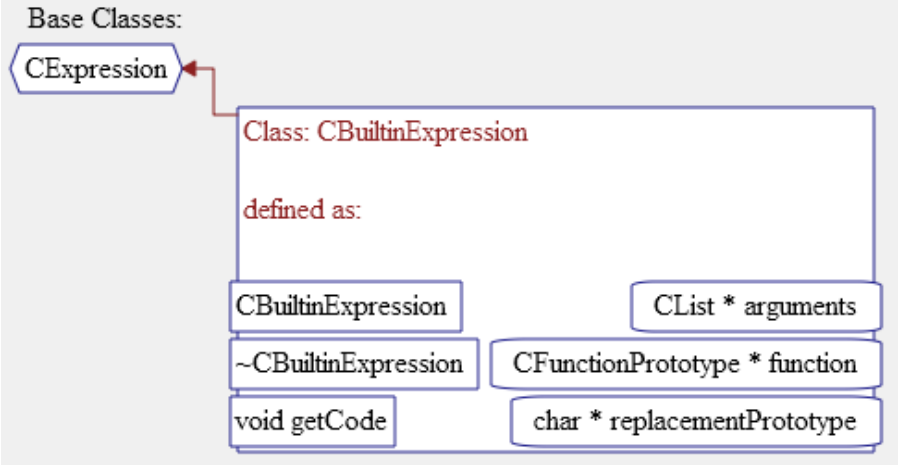

- **Declaration File:** Shows the entities declared in the file. Also shows files included by the file and classes imported by the file.



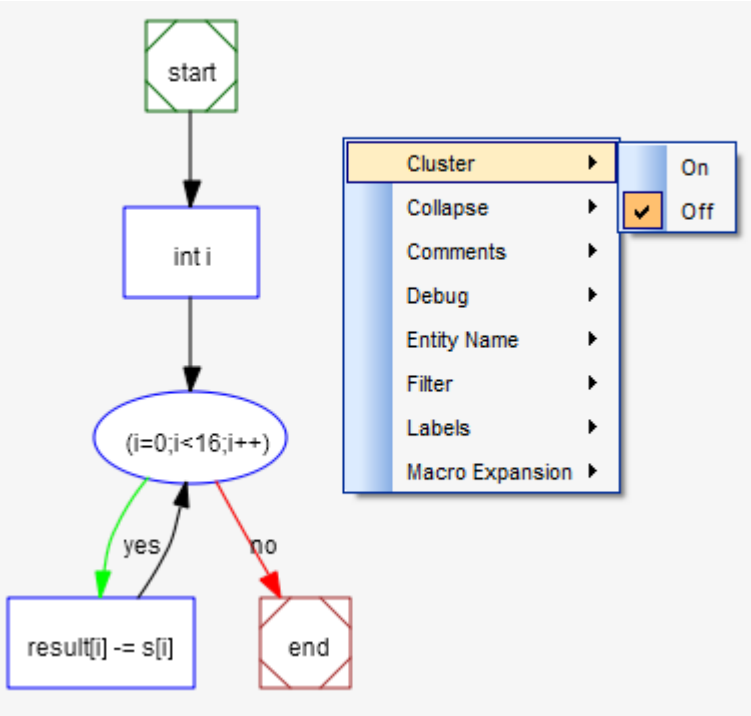
- **Declaration Type:** Shows information about a type declaration.




- **Class Declaration:** Shows the members defining the class and the parent class from which it is derived.



- **Control Flow:** Shows a flow chart of the function or similar entity type. As the following figure shows, a number of specialized options can be set when you right-click on this type of graph.



# Graphical Notation

The following symbols are used by *Understand* to represent various language constructs. The symbols vary somewhat depending upon the type of view.

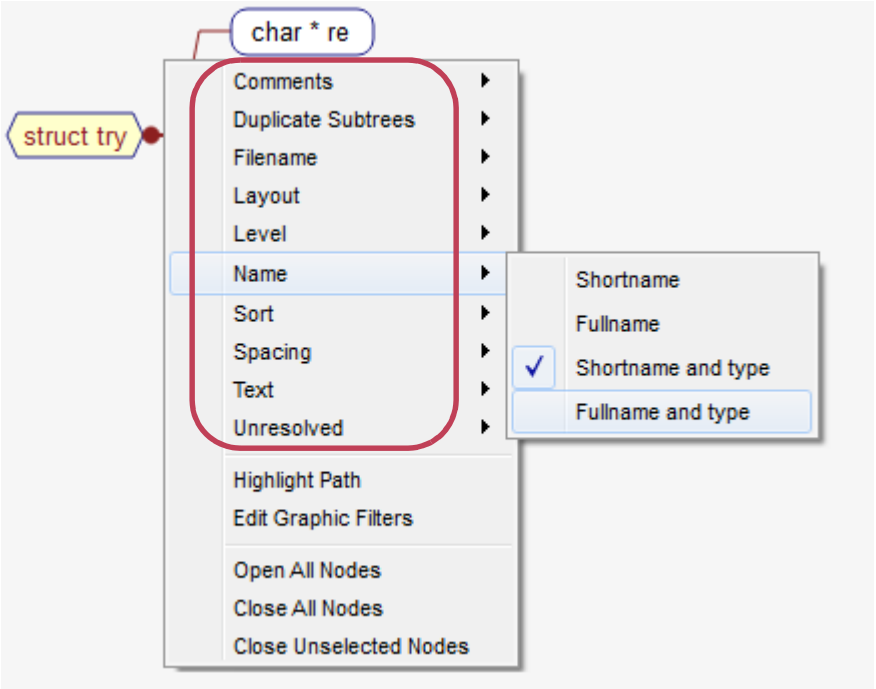
- Entities such as functions and other program units are shown in rectangles.
- Files and system-level entities are usually shown in parallelograms.
- Classes and types are shown in flattened hexagons.
- Macros are usually shown in flattened octagons.
- Objects such as variables are usually shown in slightly rounded rectangles.
- Unknown or unresolved entities are drawn with dashed outlines or in gray.
- Other shapes are language-specific.

In Control Flow views, standard flow chart symbols, such as diamonds for decision points, are used.

# Controlling Graphical View Layout

This section applies to non-cluster graphs. For information about using cluster graphs, see Controlling Cluster Graph Layout on page 272.

The two main types of graphical view windows, ***Hierarchy*** and ***Structure***, have a variety of configuration options. You can set them by right-clicking on the background of a graphical view and choosing the option you want to modify from the context menu.



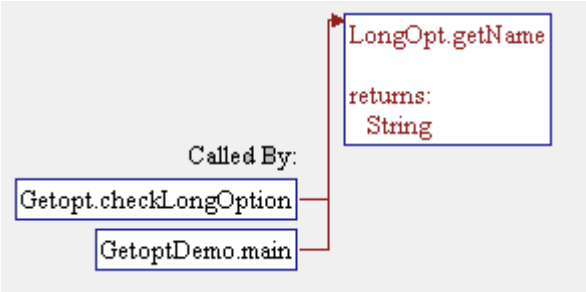
These options control the layout and drawing of the graphic views. The following subsections describe a number of these options. The list of available options varies depending on the type of view.

Note that the options for Cluster graphs are different, and are described in *Controlling Cluster Graph Layout* on page 272.

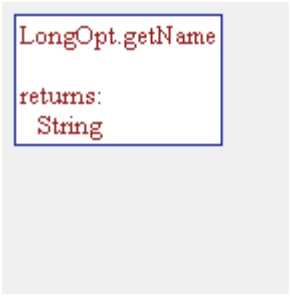
**Called by Menu**

The **Called by** menu controls whether program units that call the current entity are shown in declaration views.

**View with Called By set to On**




**View with Called By set to Off**




**Comments Menu**

The **Comments** menu controls whether to show any comments associated with an entity. The default is Off.

**Constants Menu**

The **Constants** menu controls whether to show constants in Declaration views. The default is On.

**Default Members Menu**

The **Default Members** menu controls whether declaration views show default members of the class.

**Dependent Of Menu**

The **Dependent Of** menu controls whether files a C file is dependent on are drawn in the C File Declaration view. The default is On.

**Dependent Menu**

If **Dependents** is on (the default) then files dependent on the current C file are shown in a File Declaration view.

**Depth**

Sets the number of levels to which a dependency graph is expanded. The default is 1 level.

**Duplicate Subtrees Menu**

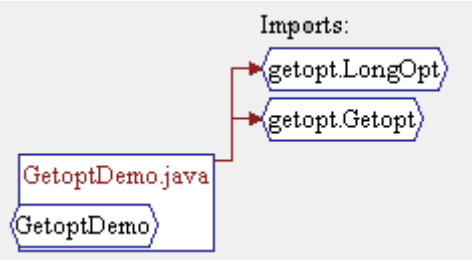
The **Duplicate Subtrees** menu controls whether multiple occurrences of the same subtree are shown in hierarchy views. The options are to Hide or Show such subtrees. The default is to show duplicate subtrees. In some applications, hiding duplicate subtrees can dramatically simplify hierarchy views. Duplicate subtrees are not shown if a view has over 1000 nodes.

| | |
|---|---|
| **Expand Recursive Notes** | Controls whether recursive nodes in a dependency graph are shown as separate items. The default is to show the expansion for recursive nodes. If you turn this setting off, a particular item is expanded only at the highest level where it occurs in the architecture, class, or package hierarchy. Unexpanded nodes that are recursive at lower levels display “(recursive)” as part of their text. |
| **Expand Repeated Notes** | Controls whether repeated nodes in a dependency graph are shown as separate items. The default is to show the expansion for repeated nodes. If you turn this setting off, a particular item is expanded only at the highest level where it occurs in the architecture, class, or package hierarchy. Unexpanded nodes that are repeated at lower levels display “(repeated)” as part of their text. |
| **Extended By Menu** | The **Extended By** menu controls whether declaration views show classes by which the selected class is extended. |
| **Extends Menu** | The **Extends** menu controls whether declaration views show classes that the selected class extends. |
| **External Functions Menu** | If **External Functions** is on then functions defined in a header file or in a file included by a header file are shown in the Declaration View for a header file. The default is On. |
| **Filename Menu** | The **Filename** menu controls how filenames are displayed in views. It is available for both declaration and hierarchy views.<br>• **None:** Filenames are not shown in the view.<br>• **Shortname:** Where filenames are relevant, only the name of the file is shown in square brackets.<br>• **Fullname:** Where filenames are relevant, the full file path and filename are shown in square brackets. |
| **Function Pointer Menu** | The **Function Pointer** menu controls whether function pointers are displayed as invocations in the Call and CallBy trees. |
| **Globals Menu** | The **Globals** menu controls whether to show globals in Declaration views. The default is On. |
| **Implements Menu** | The **Implements** menu controls whether declaration views show entities that the selected entity implements. |
| **Implemented By Menu** | The **Implemented By** menu controls whether declaration views show entities by which the selected entity is implemented. |

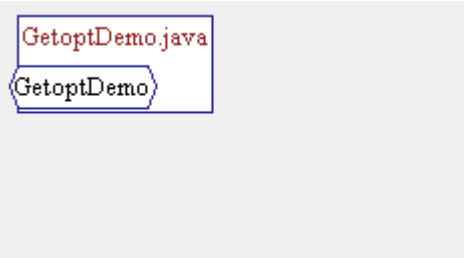
**Imports Menu**

The **Imports** menu controls whether declaration views show entities imported by the current entity.

View with Imports set to On




View with Imports set to Off




**Included By Menu**

If **IncludeBy** is on (default) then files that include the Header File being drawn in a Header File Declaration view are shown.

**Includes Menu**

The **Includes** menu controls if include files are drawn on file declaration diagrams (C file, Header file). The default is On.

**Inherits Menu**

The **Inherits** menu controls whether declaration views show entities that the selected entity inherits.

**Inherited By Menu**

The **Inherited By** menu controls whether declaration views show entities inherited by the selected entity.

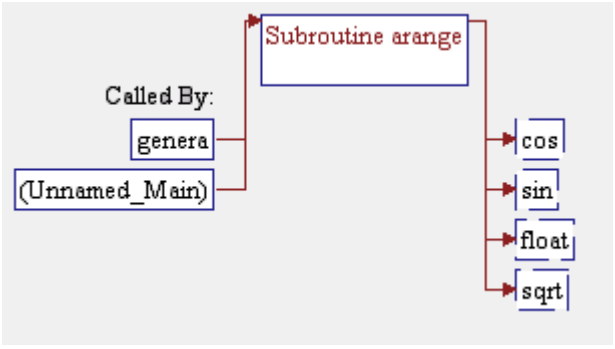
**Intrinsic Functions Menu**

The **Intrinsic** menu controls whether intrinsic functions (for example, cos and sin) are displayed or hidden.

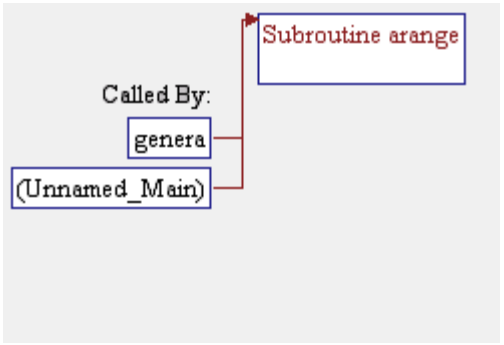
**Invocations Menu**

The **Invocations** menu controls whether procedures and functions called by the current procedure or function are shown in Declaration views.

**View shows Invocations**



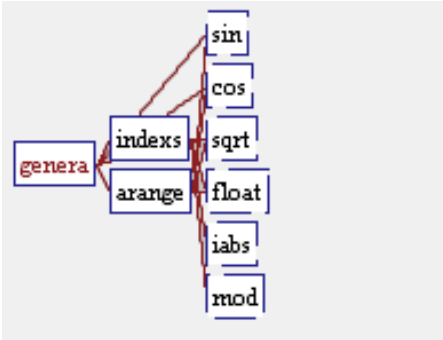

**View without Invocations shown**



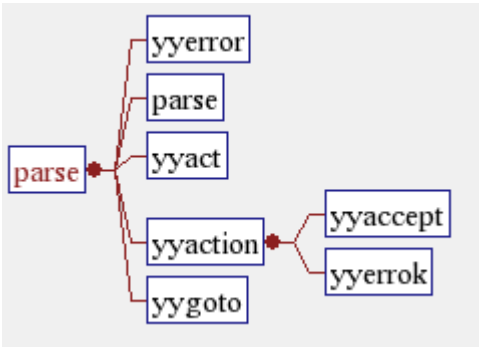
### Layout Menu

The **Layout** menu controls the layout algorithm for a hierarchical chart. It is available only in hierarchy views (calls, callby, etc.). The options are:

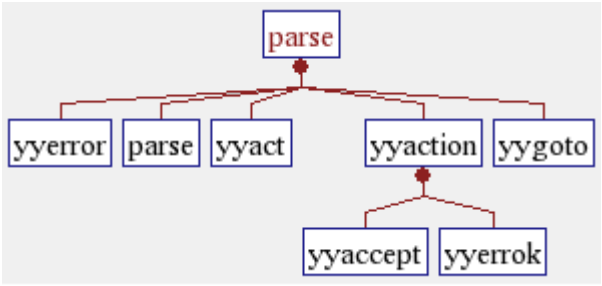
- **Crossing:** A left-to-right view, minimizing space used but sacrificing some readability by permitting lines between entities to cross.




- **Horizontal Non-Crossing:** A left-to-right layout, using more space in some situations but enhancing readability by having no crossing lines.

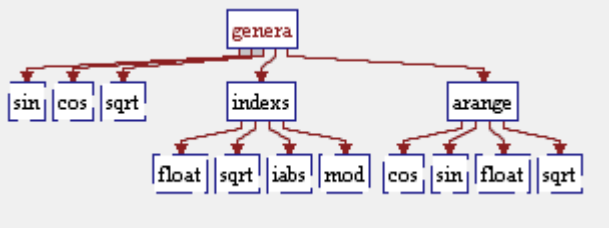



- **Vertical Non-Crossing:** A top-to-bottom layout similar to Horizontal Non-Crossing.

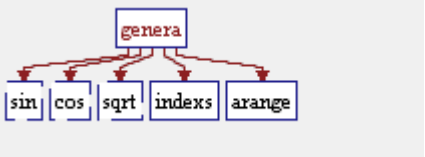


**Level Menu**

The **Level** menu controls the number of levels to be traversed when laying out a hierarchical view. The default value is “All Levels”. Values of 1 to 5 may be set. It is available only in hierarchy views.




**All Levels**




**One Level**

**Locals Menu**

The **Locals** menu controls whether local items are shown in Declaration views. The default is On.

**Members Menu**

The **Members** menu controls whether members and operators are shown in the Type Tree and Type Derived From views. The choices are to show None, Components, Operators, or Operators and Components.

**Name Menu**

The **Name** menu controls whether or not fullnames are used in views. It is available for both declaration and hierarchy views.

A fullname includes its parent compilation units. For example:

- Text_Io.Put is the fully specified name.
- Put is the Short Name

Longer versus shorter names can alter the layout of pictures substantially.

**Objects Menu**

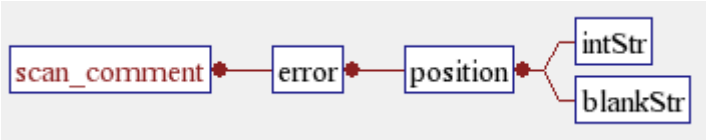
The **Objects** menu controls whether to show objects in Declaration views. The default is On.

**Operators Menu**

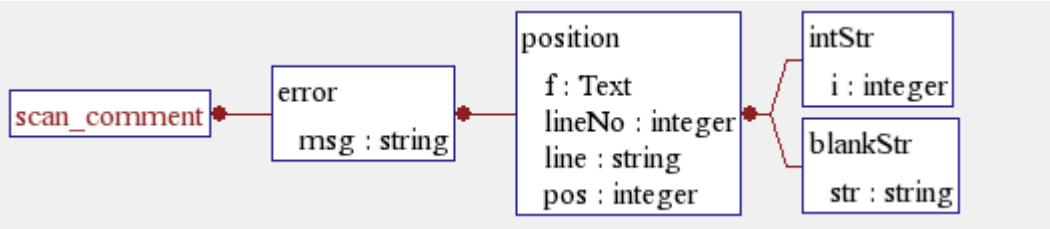
The **Operators** menu controls whether entities that are operators are shown in the Callby, Declaration, Declaration Tree, and Invocation views.

**Parameters Menu**

The **Parameters** menu controls whether parameters are shown in hierarchical views. 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.




**Parameters Off (the default)**




**Parameters On**

**Private Members Menu**

The **Private Members** menu controls whether declaration views show private members of the entity.

**Protected Members Menu**

The **Protected Members** menu controls whether declaration views show protected members of the entity.

**Public Members Menu**

The **Public Members** menu controls whether declaration views show public members of the entity.

**Renames Menu**

The **Renames** menu controls whether declarations that are renames are shown in Declaration views. The default is to show rename declarations.

**Routines Menu**

The **Routines** menu controls whether to show routines (procedures, functions, etc.) in Declaration views. The default is On.

**Scale Menu**

The **Scale** menu allows you to choose the size of the text used. It is available for both declaration and hierarchy views. All picture sizes and layouts vary with text point size. The currently selected size is indicated by a check mark.

Other point sizes can be added by customizing configuration files found in the *Understand* installation directory. Contact support@scitools.com for information on how to do this.

**Sort Menu**

The **Sort** menu lets you specify whether entity names in tree views should be sorted alphabetically. If this option is off (the default), entities are sorted in the order they are encountered in the project.

**Spacing Menu**

The **Spacing** menu lets you choose to change the space between boxes. You can choose compact, small, normal, wide, or extra wide.

**Sql Menu**

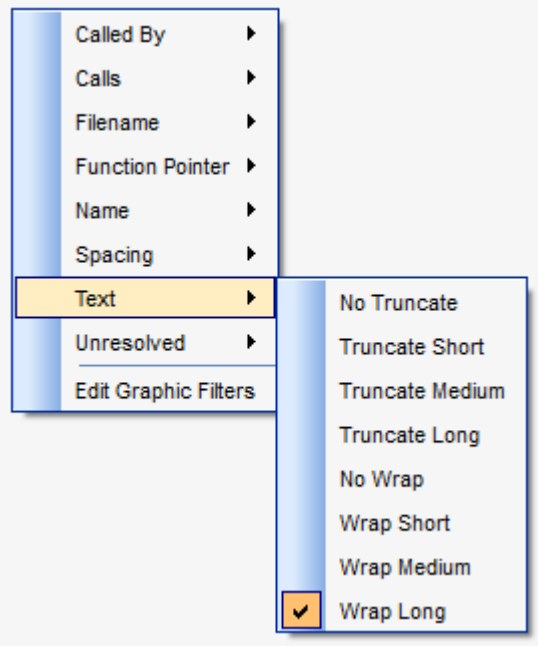
The **Sql** menu lets you specify whether SQL entities should be shown in graphical views. This option is on by default.

**Static Menu**

The **Static** menu controls if static functions are drawn in function, C File and Header File declaration views. Static functions are those declared using the “static” keyword. They are visible only within the file they are declared in. If enabled static functions are drawn with the edge of their box inside the edge of the outer declaration box for their enclosing unit (C file). The default is On.

**Text Menu**

The **Text** menu sets the way entity names are trimmed or altered to accommodate the layout of graphics. It is available for both declaration and hierarchy views. Names may be truncated to a certain length or wrapped at a certain length.




- **No Truncation:** Uses the name as defined in the source code. 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.

**Types Menu**

The **Types** menu controls whether to show types in Program Declaration views. The default is On.

**Typetext Menu**

The **Typetext** menu tells declaration views (Function Declaration, C File Declaration, Header File Declaration) to include types on the view. The default is On.

**Unknown Menu**

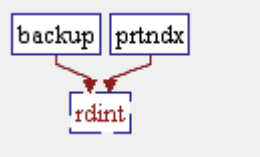
The **Unknown** menu controls whether entities should be shown if they are used in the source, but are never declared or defined.

**Unresolved Menu**

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. This option is available on hierarchy and structure views.

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:




**Usedby Menu**

The **Usedby** menu tells Declaration views whether to show items that use this item.

**Uses Menu**

The **Uses** menu tells Uses views whether to show only items that are used directly, or to also show items that are used by nested subprograms. The default is to show both.

**Variables Menu**

The **Variables** menu controls whether to show globals in Declaration views. The default is On.

**Withs Menu**

The **Withs** menu controls on Declaration views of compilation units (packages, tasks, separate procedures, etc...) if Withs are drawn. The default is On.

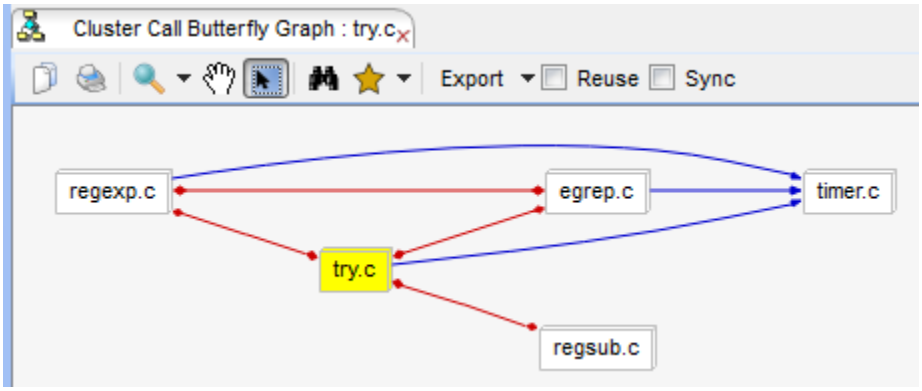
**With Bys Menu**

Controls if **With Bys** (who Withs a given compilation unit) are shown on Declaration views. The default is On.

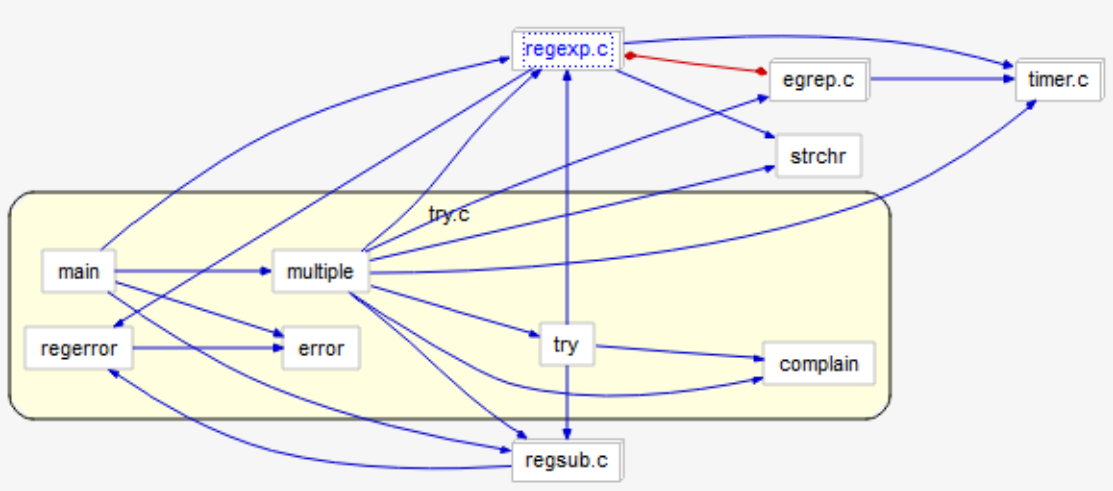
# Controlling Cluster Graph Layout

Cluster graphs are a special type of hierarchy view. They provide a more interactive view of call relationships than other hierarchy views. The **Cluster Call**, **Cluster Callby**, **Cluster Call Butterfly, Cluster Call Internal**, and **Cluster Control Flow** variants are available, and can be accessed from the function, class, file, or architecture level. The Architecture Dependency graphs (page 196) behave similarly to the cluster graphs. There are some special menus for Cluster Control Flow graphs that are described on page 274.

For example, if you open a Cluster Call Butterfly graph for a file, you see a graph similar to the following:




If you then double-click on some of the file boxes, you can see call relationships for functions within the files that you expand.



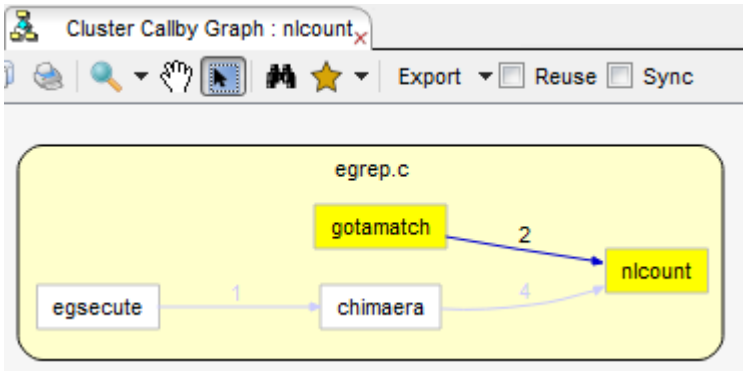

Hold down Ctrl+Shift and use the mouse wheel to zoom in and out in cluster graphs.

Hold down Alt and click on an entity to go to the code that defines that entity.

The toolbar for cluster graphs is the same as for other graphs, and the context menu for entities in the graph is similar to elsewhere.

The context menu when you right-click on the background of a cluster graph offers the following options:

- **Aggregate Nodes by:** Choose an architecture you want to organize entity nodes.
- **Edges Shown:** Choose which relationships to the originally selected entity you want shown. “Forward” is call relationships. “Reverse” is callby relationships. “Butterfly” is both call and callby relationships.
- **Entity Name Format as:** Choose whether you want to display short or long names for entities.
- **Highlight Paths to Selected Node(s):** You can highlight all paths between the node for which you opened a Cluster Call or Cluster Callby graph and some other node. To do this, select a node (not the original node), right-click on the background of the graph (not on an entity or within a box), and choose this option. You can hold down the Ctrl key to select multiple entities for path highlighting




- **Include Virtual Edges:** Set this item to On if you want to show override and overriddenby edges. Such edges are light blue by default. If you want to change this color, choose **Tools > Options**. In the **Graphs** category, make sure **Use custom style on cluster call graphs** is checked. Then.create a new Edges style called "overrides,overriddenby".
- **Show Edge Labels:** Set this item to On if you want the number of occurrences of this relation to be shown in the graph. For bi-directional call relationships, the two numbers in the label show calls in each direction.
- **Show Edges Between Children By Default:** Set this item to On if you want 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.
- **Show Legend:** Set this item to On if you want to show a graph legend in the upper left. The legend identifies the shapes and arrow styles used in the graph.
- **Show Node Children By Default:** Set this item to On if you want nodes to be opened by default when you open a cluster graph. For example, all functions within files will be shown by default if this option is enabled when you open the Cluster Callby graph for a file.

The context menu when you right-click on a node in a cluster graph offers commands similar to those you see when right-clicking on an entity name elsewhere in *Understand*.

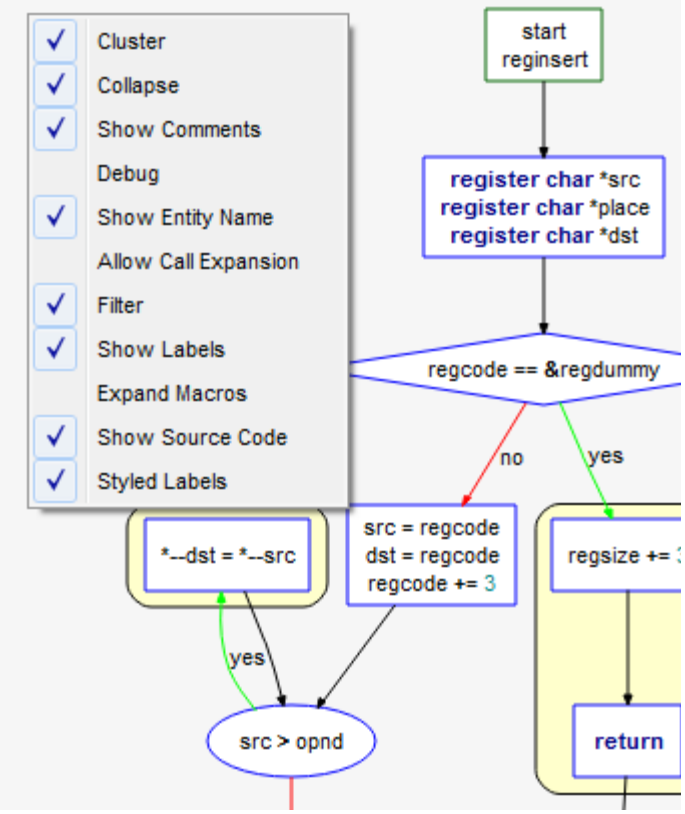
The context menu when you right-click on an edge (arrow) in a cluster graph provides a list of the references that constitute the edge. Choose an item from the list to visit the source code for this relationship. You can limit the length of this list as described for the **Tools > Options** dialog on page 118.

The Graph Customizer to the right of a cluster graph offers the same settings as those described for Architecture Dependency Graphs in *Graph Customizer Toolbar* on page 197.

You can customize the display colors, shapes, and arrows used in cluster graphs in the Graphs category of the **Tools > Options** dialog (page 118).

## Cluster Control Flow Graphs

Cluster Control Flow graphs show the execution flow of an entity such as a function. These graphs have the same toolbar as other graphs and a Graph Customizer toolbar (page 197) similar to those in other Cluster Graphs and Architecture Dependency Graphs. However, the context menu when you right-click includes the following options:




- **Allow Call Expansion:** Allows called functions to be expanded by clicking. If this option is on, expandable calls are shown as a 3D box. Off by default.
- **Cluster:** Uses a box to enclose statements in a group such as the “if” or “else” branch of a conditional statement. On by default.
- **Collapse:** Combines statements into a single box if there are no decision points between them. On by default.
- **Debug:** Shows details about the information about each item in the flow. In order, the detail information is: nodeID, nodeKind, startLine, startCol, endLine, endCol, endNode, commaSepararatedListOfChildren. Off by default.

- **Expand Macros:** Enabling this option shows macros expanded if you have enabled the **Save macro expansion text** option in the C++ project configuration (page 68). Off by default.
- **Filter:** Hides implicit actions, such as “endif”. On by default.
- **Layout:** Choose whether to arrange the graph vertically or horizontally. The default is Vertical.
- **Show Comments:** Shows comments associated with statements in the graph. On by default.
- **Show Finally-Block Flows:** Shows edges representing exceptional exits from a try-catch block in languages like Java and C#. On by default.
- **Show Entity Name:** Shows the name of the entity in the Start box at the beginning. Off by default. You can also choose to show entity names with parameters included.
- **Show Labels:** Shows text for edges (for example, yes/no) and start block. On by default.
- **Show Legend:** Set this item to On if you want to show a graph legend in the upper left. The legend identifies the shapes and arrow styles used in the graph.
- **Show Source Code:** Shows source code in boxes. On by default.
- **Styled Labels:** Highlights keywords, comments, and strings in source code shown in the graph. The formats defined in the Editor > Styles category of the Understand Options dialog (page 115) are used. On by default.

## Saving Graphical Views

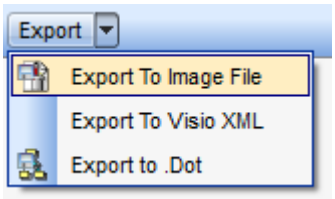
*Understand* offers a number of ways to export your graphical views and use them in other ways. The toolbar for each graphical view provides the following icons for copying and printing graphs.


In addition to printing, you can save graphical views as JPEG, PNG, SVG files (page 276), Visio XML files (page 276), and DOT files (page 277). The first three formats are common graphics formats.

For cluster graphs, you can save and load customized graph settings as described for Dependency graphs (page 197).

## Saving Views to Files

To save a graphical view in one of the following formats, use the **Export** drop-down the graphical view toolbar to choose the **Export to Image File** option. Or choose **File > Export to Image File** from the menus. In the Export dialog, choose a location, filename, and file type for the view.




- **JPEG** files are compressed bitmaps. They can be viewed with most web browsers, document editors, and graphics programs. This format is “lossy”; some data is lost in the compression.
- **PNG** files store compressed bitmaps similar to GIF files. They can be viewed with most web browsers, document editors, and graphics programs. They use a non-patented compression method.
- **SVG** files are Scalable Vector Graphics files. This file type uses XML to describe a 2-dimensional vector-based image.

You can also copy a graphical view to the clipboard and paste it as a bitmap into the image program or word processor of your choice. To do this, click the **Copy** icon on the graphical view toolbar or choose **Edit > Copy Image to Clipboard** from the menus. Then, paste the image into another program.

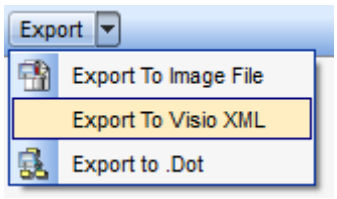
Note that if the graph would result in an image larger than 200 MB, the graph will be resized to a smaller size.

## Saving Views as Visio Files

Microsoft Visio is a vector-based graphics program used for drawing flowcharts and similar graphics. That is, it deals with shapes and objects rather than pixels. Visio XML is an Extended Markup Language that is supported by Visio and a number of other graphics applications.

You do not need to have Visio installed in order to save a graphical view as a Visio XML file.

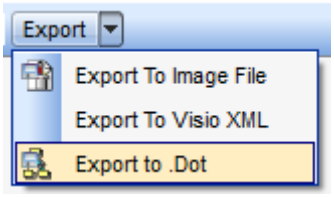
To save a Visio XML file, use the **Export** drop-down the graphical view toolbar to choose the **Export to Image File** option. In the Export dialog, choose a location and filename for the view. The file extension for Visio XML files is *.vdx.



## Saving Views as DOT Files

DOT is a language used to describe graphs in plain text. This format can be imported and edited by a number of external tools. You can export many (but not all) types of graphs produced by *Understand* to a DOT file.

To save a DOT file, use the **Export** drop-down the graphical view toolbar to choose the **Export to .Dot** option. In the Export dialog, choose a location and filename for the view. The file extension is *.dot.




If this option is not shown in the **Export** drop-down, the current graph cannot be exported to the DOT format.

# Printing Graphical Views

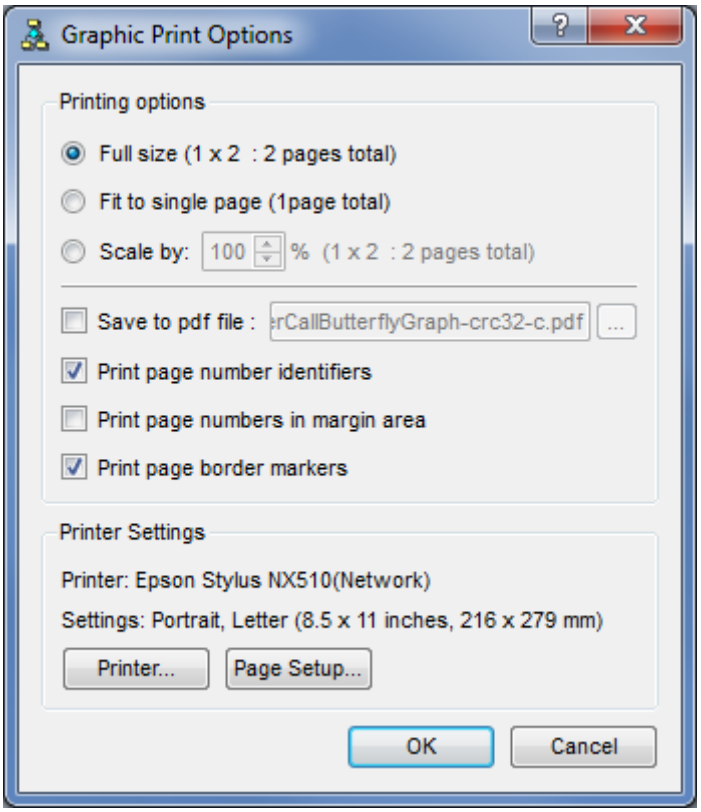
*Understand* has these printing modes:

- **Source File** printing sends a text file to the printer using 66 lines of source per page. See *Printing Source Views* on page 190.
- **Graphical view printing** provides options for how to fit the image to a page. See *Graphical View Printing* on page 278.

## Graphical View Printing

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

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




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

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

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

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


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

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

Click the **Printer** button to open the standard Print dialog for your operating system. When you click **Print** or **OK** in that dialog, you return to the Graphic Print Options dialog.

Click the **Page Setup** button to open a Page Setup dialog, which allows you to choose the paper size, paper source (if applicable), page orientation, and margin width. Click **OK** to return to the Graphic Print Options dialog.

Click the **OK** button in the Graphic Print Options dialog to send the graphical view to the printer (or a PDF file).

***Note:*** The **File > Page Setup** menu option applies only to printing source code and other text files. The **Page Setup** button on the Graphic Print Options dialog saves its settings separately.

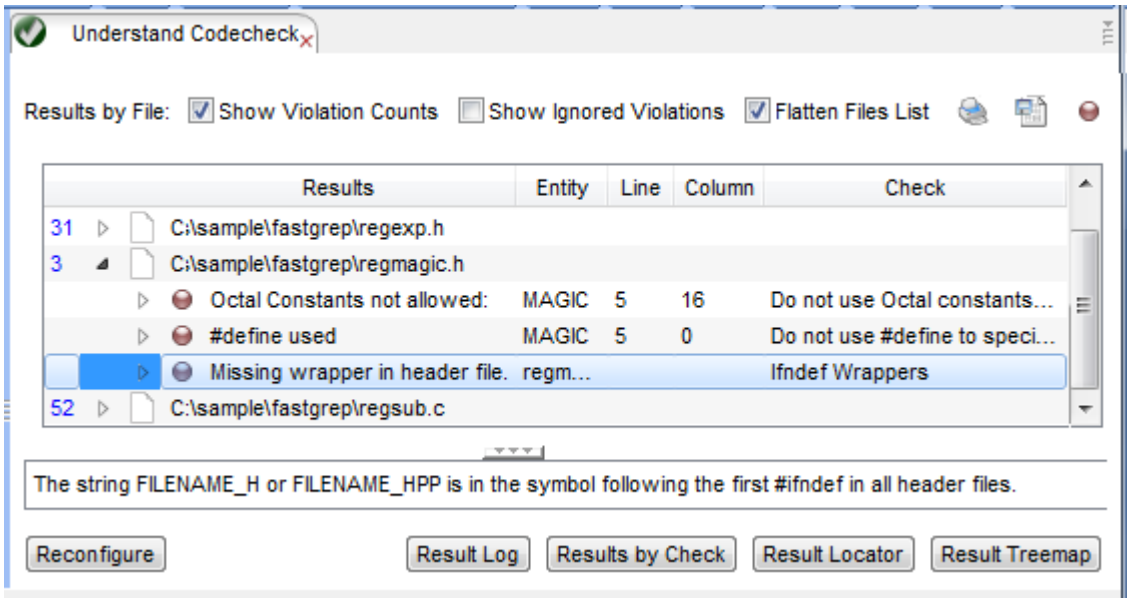
# Chapter 11 Using CodeCheck for Standards Verification

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

This chapter contains the following sections:

# About CodeCheck

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




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

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

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

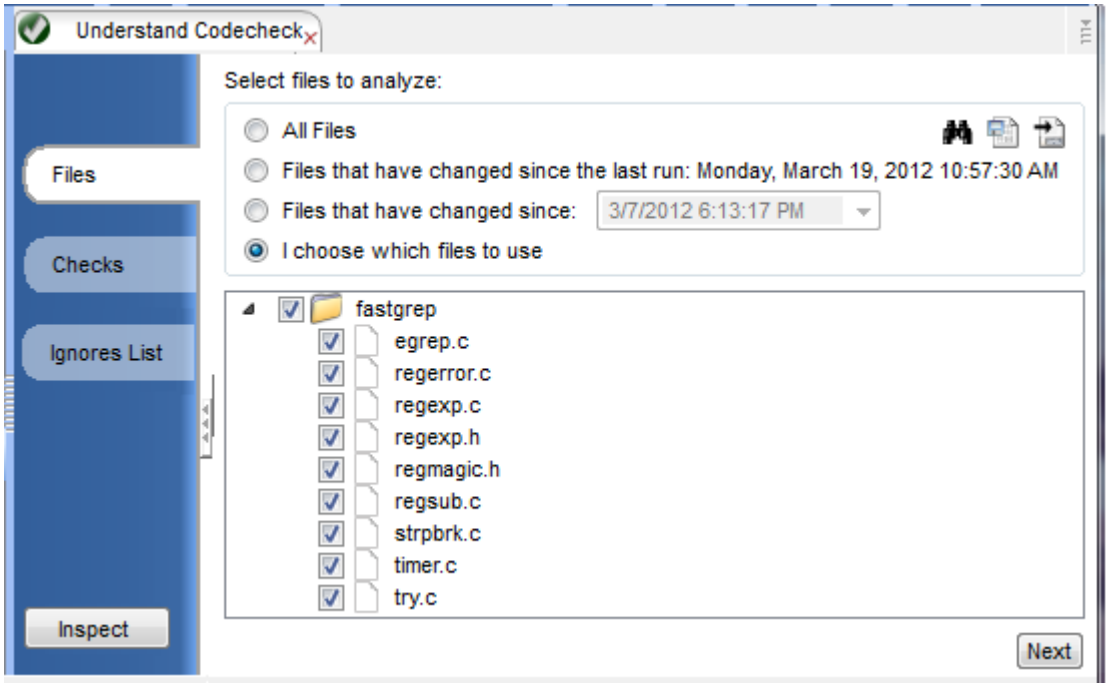
CodeCheck validation suites are available from the SciTools website.

# Running a CodeCheck

To open the CodeCheck tool, choose **CodeCheck > Open CodeCheck** from the menus.

**Files Tab**

In the **Files** tab, choose whether to check all files in the project, only files that have changed since you last ran CodeCheck, only files that have changed since a specific date, or the files you select.




If no files have been changed since a date you select or since the last time you ran CodeCheck, you will see a message that says no files meet the criteria.

You can click the **Search** icon to open a search bar below the list of files that lets you search by filename.

If your project contains many files, you can more easily select specific files by following these steps:

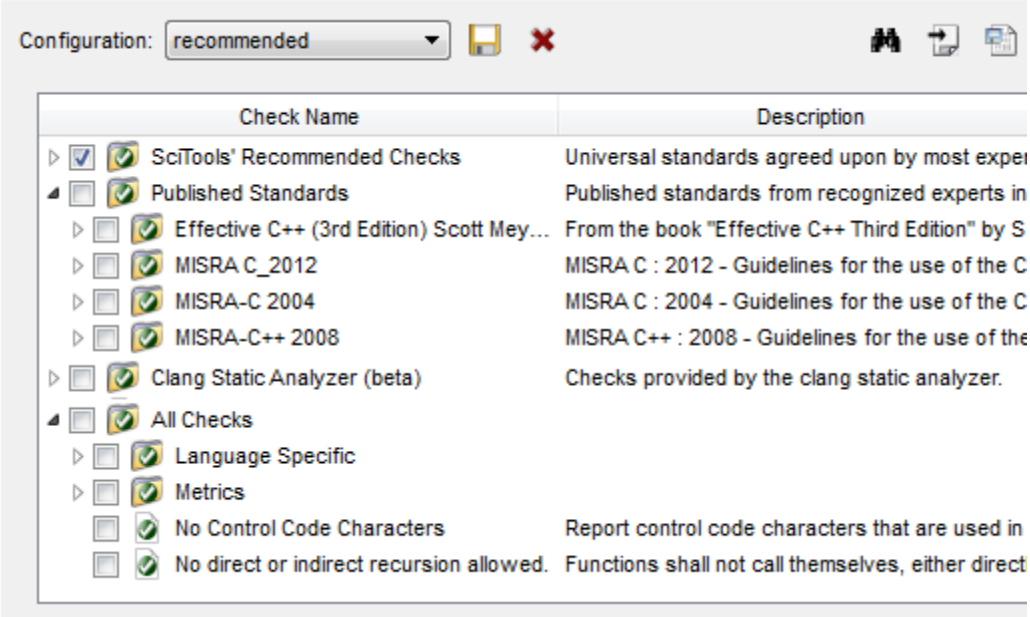
1. Export the entire list of currently selected files (with full file paths) to a text file by clicking the **Export Selected Files** icon.
2. Edit the file with a text editor. Delete the lines for files you do not want to inspect, and save the file as plain text.
3. Click the **Import File List** icon and import the file you edited. This will select only the files listed in the file.

Once you have finished selecting the files to inspect, click **Next** to move to the Checks tab.

### Checks Tab

In the **Checks** tab, the default configuration is **New** and all the **SciTools' Recommended Checks** that apply to your programming languages are selected.

If you want to use a different CodeCheck configuration or save the configuration you are creating for later use, see *Using CodeCheck Configurations* on page 292.




Choose the checks you want to perform. The following types of options are provided:

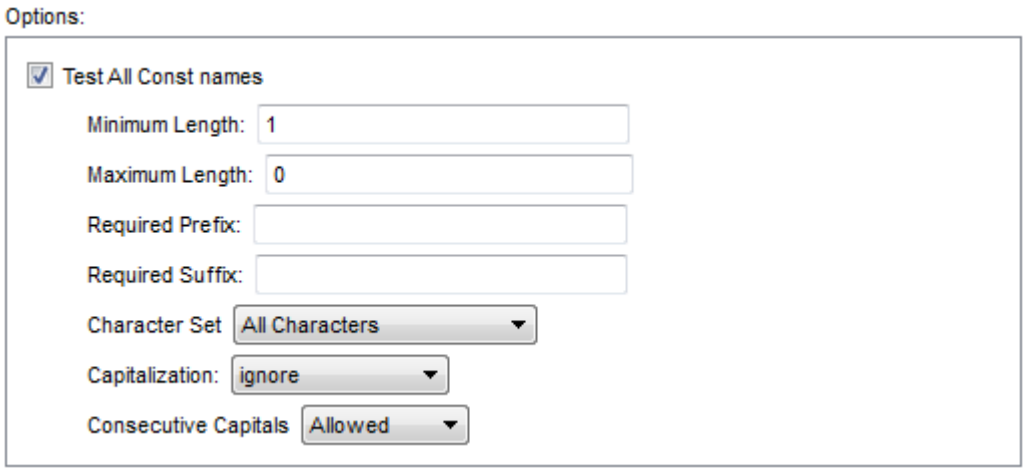
- **SciTools' Recommended Checks:** This category lists recommended checks for your source code languages. These are standards violations that we feel are most serious.
- **Published Standards:** Collections of checks are provided to see if your code conforms to the following standards or recommendations:
  - Effective C++ (3rd Edition) Scott Meyers
  - MISRA-C 2012
  - MISRA-C 2004
  - MISRA-C++ 2008
- **Custom Checks:** Any custom checks you have installed are listed in the Checks tab. See *Writing CodeCheck Scripts* on page 293.
- **Clang Static Analyzer:** Checks provided by the Clang Static Analyzer tool. These checks are most accurate when used with the strict C parser.
- **Language Specific Checks:** All checks that *Understand* can perform are listed here. They are organized first by programming language, then by category, and finally by check. Currently, most checks apply to C/C++ code, but some checks are available for other languages. You can confirm that your naming conventions are met for various entity types in most supported languages. You should check specific languages under the All Checks node; if you check the **All Checks** box, some of the checks may conflict with others, and errors are likely to occur.

- **Metrics Checks:** You can perform checks based on the values of complexity, function length, and nesting depth metrics, which are described in the list at scitools.com/support/metrics_list/.
- **General Checks:** General checks for preferences such as lines longer than a maximum number of characters or the presence of various control characters.

Press Ctrl+F or click the **Search** icon to be able to search for a check.

When you select a check, information about that check is shown in the **Detailed Description** area. You can copy the text in the description if you want to paste it into a report or email message. You can hide the description by clicking the arrowhead above the description.

For a number of checks, options are shown below the description. For example, if you select a metrics check, you can set a value that needs to be met. If you select a Naming Conventions check, you can specify a minimum and maximum length for acceptable names, any required prefix or suffix, and the types of characters and capitalization rules that names need to follow. The description provides details about the options.




Check the **Use Verbose Logging** box if you want the Result Log to include a separate line for each violation found. Otherwise, the Result Log will present a summary and you can use the other tabs to sort through and view specific violations.

Once you have finished selecting the checks to perform, click **Inspect**.

You can choose **CodeCheck > Analyze Changes and Re-Run Previous Checks** from the menu bar to run the **Analyze Changed Files** command and then the **Re-Run Previous Checks** command.

If you have made changes to the checks being performed, you are asked if you want to save the configuration before performing the analysis. Click **Yes** if you might want to perform this same set of checks in the future. If you were using the “New” configuration, you are then prompted to type a name for this configuration. Type a name and click **OK**. See *Using CodeCheck Configurations* on page 292 for more information.

If more than 300,000 violations are detected, you are asked if you want to continue the check.

### Exporting and Importing Configurations

If you want to edit the list of checks being performed using a text file, do the following:

1. Export the entire list of currently selected checks (with full file paths) to a *.ini file by clicking the **Export Configuration to File** icon in the Checks tab.
2. Edit the file with a text editor. Delete the lines for checks you do not want to apply, and save the file as plain text.
3. Click the **Import Configuration from File** icon in the Checks tab and import the *.ini file you edited. This will select only the checks listed in the file.

Exporting the configuration to a file is also required if you want to be able to perform a CodeCheck analysis from the command line.

You can export the list of checks performed to a text file by clicking the **Export Selected Checks to Text File** icon. This file includes a description of each test.

# Viewing CodeCheck Results

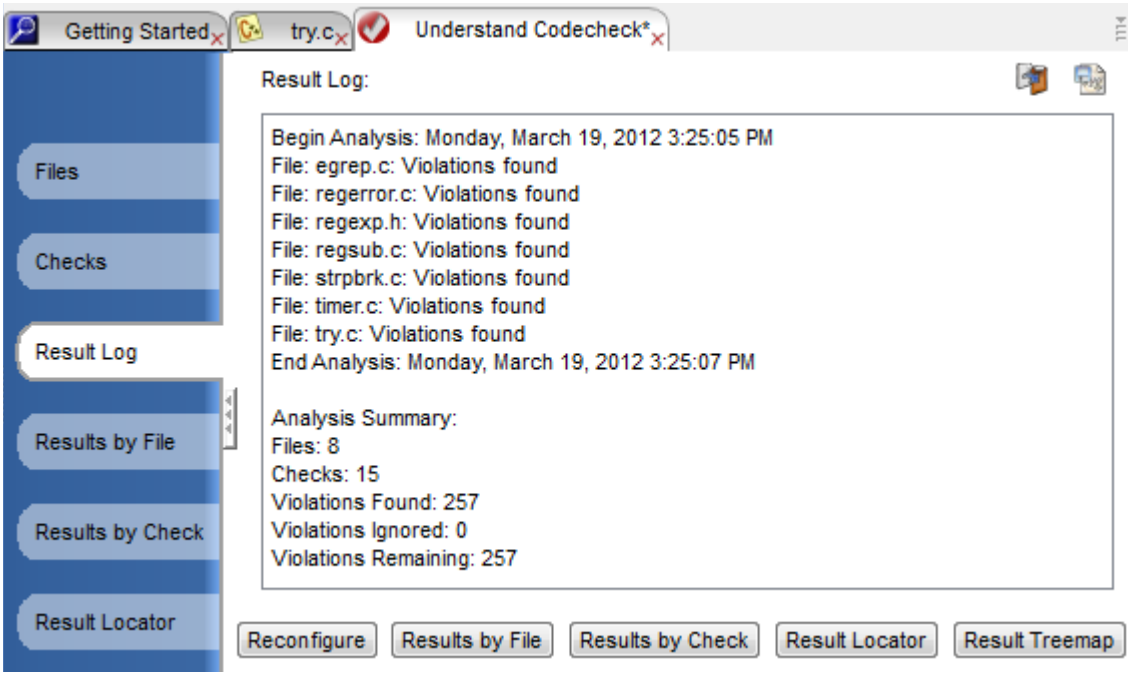
After you perform a CodeCheck analysis, you can view the results in the Results Log (page 285), Results by File tab (page 286), Results by Check tab (page 287), Result Locator tab (page 288), and Result Treemap (page 289). You can also print or export the results (page 290).

You can hide results by ignoring particular checks and violations (page 290).

### Using the Result Log

When you run a CodeCheck analysis, you automatically see the **Result Log** tab, which provides a summary of the results.

The Result Log includes the number of files checked, how many checks were performed, and the number of violations found.



If you checked the **Use Verbose Logging** box in the **Checks** tab, the Result Log also includes a separate line for each violation found.

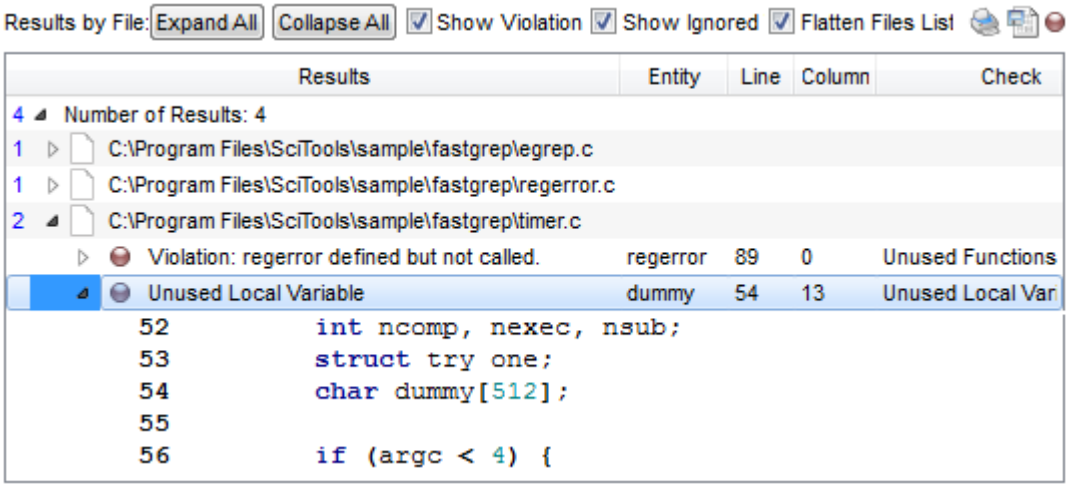
You can copy the log to your clipboard for pasting by clicking the **Copy** icon. To save the log to a file, click the **Export** icon.

### Using the Results by File Tab

Choose the **Results by File** tab to list the problems in each file of your project.

The table lists the number of violations in each file and full file paths. Uncheck the **Show Violation Counts** box above the table to hide the number of violations. Uncheck the **Flatten Files List** box to organize files in a folder hierarchy that you can expand as needed.

Click the arrow next to a filename to expand the list of violations found in that file. The line for a violation shows the problem, the name of the entity, the line number on which the problem occurs, the number of the column (0 indexed) where the problem began, and a short description of the check performed.




Click the arrow next to a violation to see the 5 lines of code surrounding the problem. You can double-click on the code to open the source file.

When you select a violation, the description of that check and any exceptions to the check are shown below the table. You can select text in this area and press Ctrl+C to copy it to your clipboard for pasting into other applications.

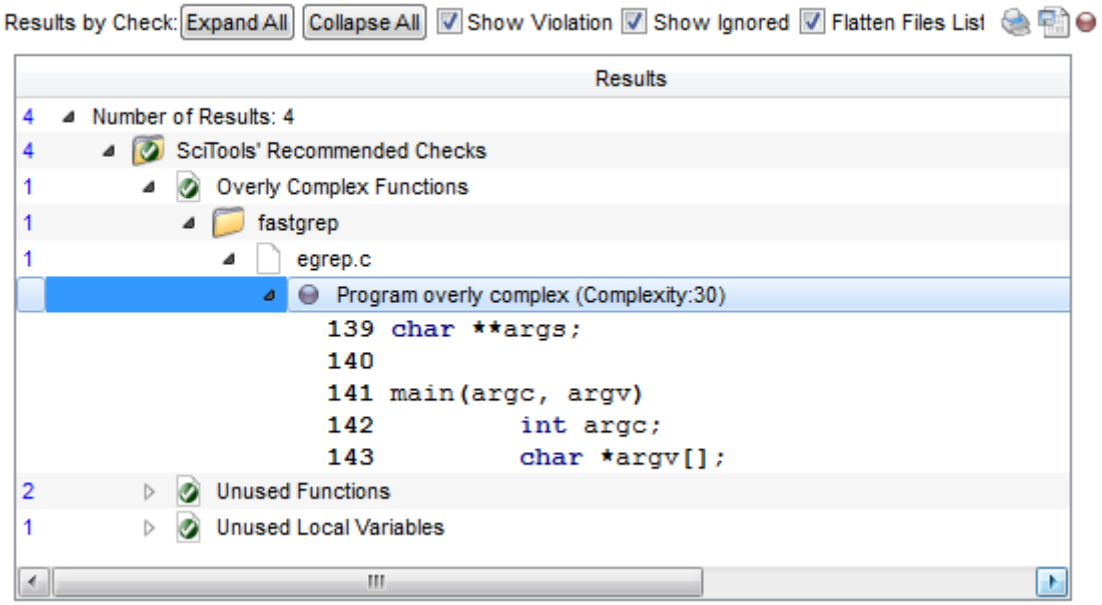
Click **Expand All** or **Collapse All** to show or hide all violation details.

Click the **Print** icon to print the list or the **Export** icon to send the full list of violations to HTML files in a directory you select, your clipboard, or a text file.

Check the **Show Ignored Violations** box if you want to see the full list of violations by overriding any ignored checks and violations. See page 290 for details.

## Using the Results by Check Tab

The **Results by Check** tab is similar to the Results by File tab (page 286). However, all violations of a particular type are listed together. The organization under each violation is either a list of files if the **Flatten Files List** box is checked or a folder hierarchy if the **Flatten Files List** box is unchecked.



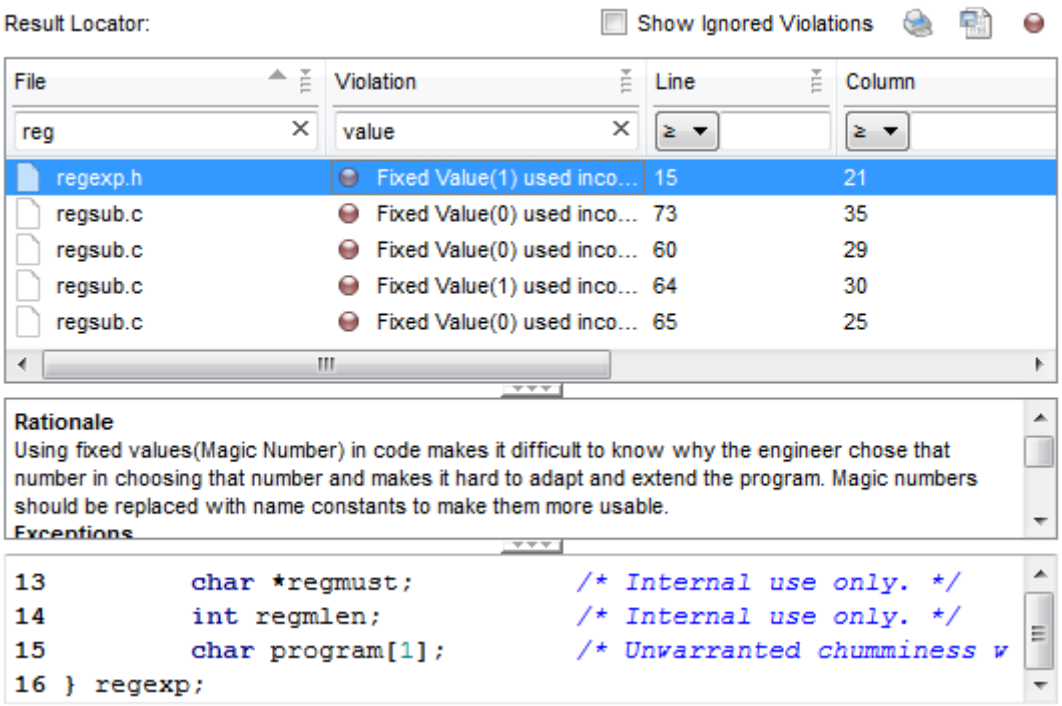

If many violations of a particular type are detected, you might want to look at the individual checks in the **Checks** tab to see if you can set options to control the sensitivity of the checks. For example, for the “Magic Numbers” check, you can specify that bitfields can be set to fixed values and you call allow exceptions for values like 0 and 1. Another example is that for the “Functions Too Long” check, you can set the length that is considered too long and choose to ignore comment lines and blank lines.

In the Results by Check tab, the **Export** icon allows you to either a detailed or summary report to HTML files in a directory you select, your clipboard, or a text file. 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
```

## Using the Result Locator

The **Result Locator** tab lets you search for violations using pattern matching and sorting on the file, violation name, line number, and column number.




You can type values to match filenames and violations. Right-click a column header or click the small drop-down icon to see the context menu for that column. You can choose for the filter to be case sensitive or not. You can also choose for the filter pattern matching syntax to use fixed strings (the default), wildcards, or regular expressions. To search for field values that do not contain a particular string, type **!** (exclamation mark) and then the filter.

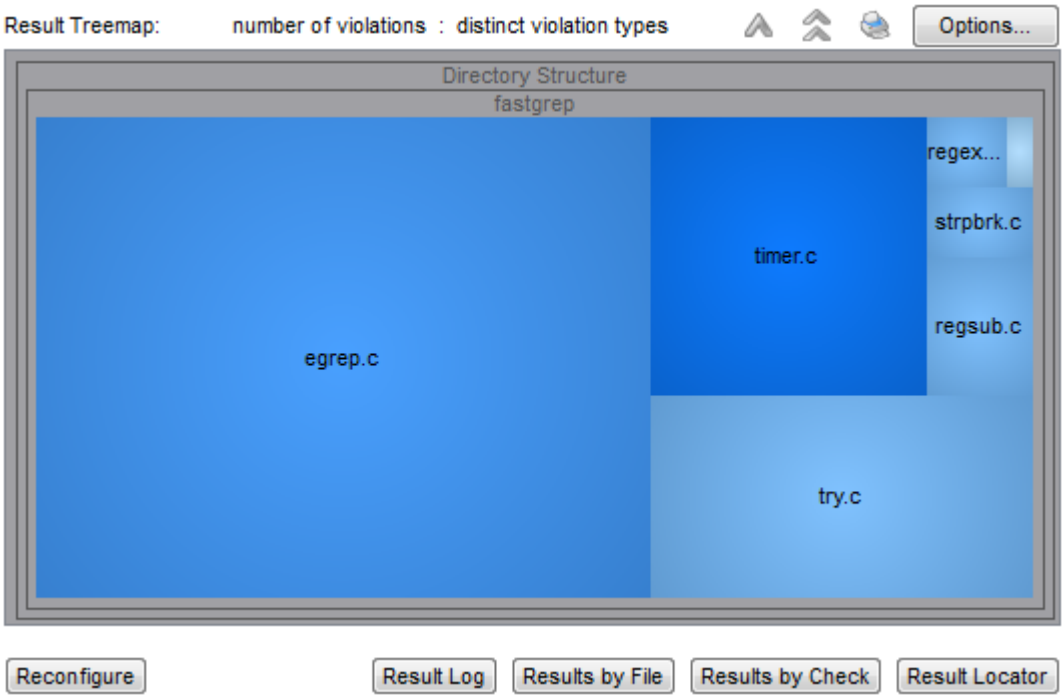
For details about using the locator fields, see *Filtering the List* on page 157.

Five lines of code surrounding the violation are shown at the bottom of the window. You can double-click this code to open the file.

## Using the Result Treemap

Treemaps show metrics graphically by varying the size of node blocks and the color gradient. Each node block represents a code file. Different metrics can be tied to size and color to help you visualize aspects of the code.

CodeCheck lets you create treemaps that show the total number or density of check violations and the number of types of violations. For example, in this treemap larger block sizes indicate more violations in that file and darker blue indicates more types of violations in that file. So, while egrep.c has the most violations, timer.c has more types of violations. Notice that the text above the treemap indicates the settings used.

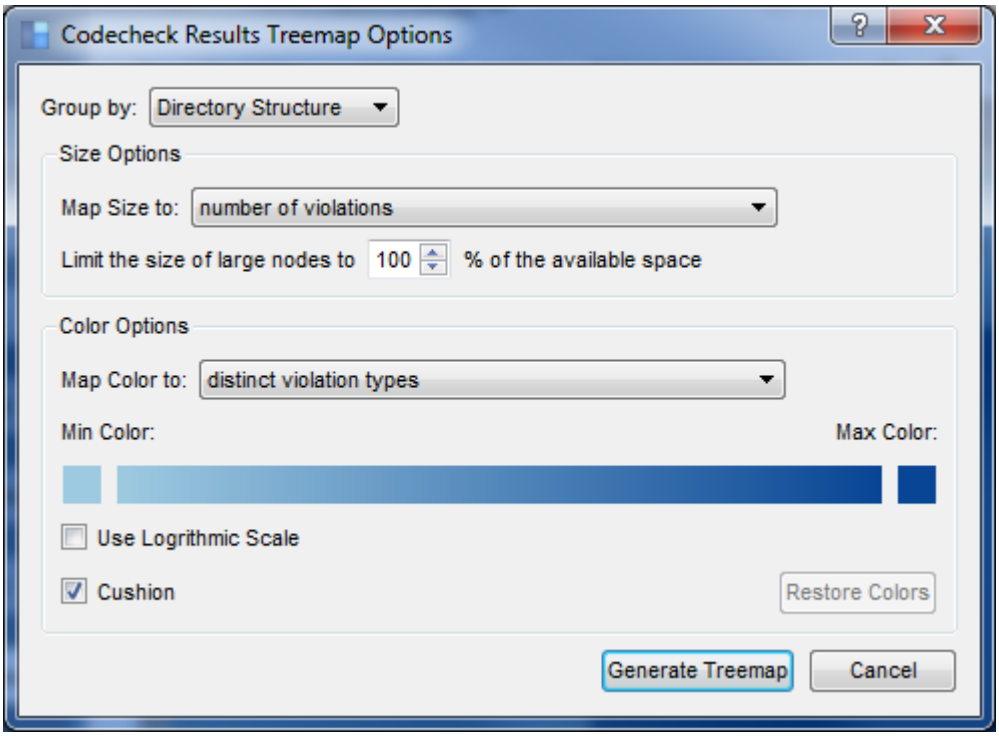



If you double-click on a file block, you see the Results by File listing with the list of violations for that file expanded.

By default, the treemap is organized using the file structure of the project as architecture nodes. Within the treemap, you can double-click on an architecture node (shown as a gray border around a set of colored blocks) to display only the contents of that node. You can also zoom in by right-clicking on a node and choosing **Drill down** from the context menu.

After drilling down in the architecture, you can use the  icons to **Pop up one level** or **Pop up all levels** the treemap. You can also right-click to use the **Pop up one level** and **Pop up all levels** commands in the context menu.

Click the **Options** button to modify which metrics are assigned to size and color:




For information about using these fields, see *Using the Metrics Treemap* on page 240.

## Printing and Exporting Results

In the Result Log tab, you can copy the log to your clipboard for pasting by clicking the **Copy** icon. To save the log to a text file, click the **Export** icon.

In the Results by File, Results by Check, and Result Locator tabs, you can click the Print icon to print the currently displayed results. You can click the **Export** icon to export the detailed results to an HTML directory, your clipboard, or a text file.

In the Result Treemap tab, you can click the Print icon to print the currently displayed treemap diagram.

See page 278 for details about the Print Options dialog.

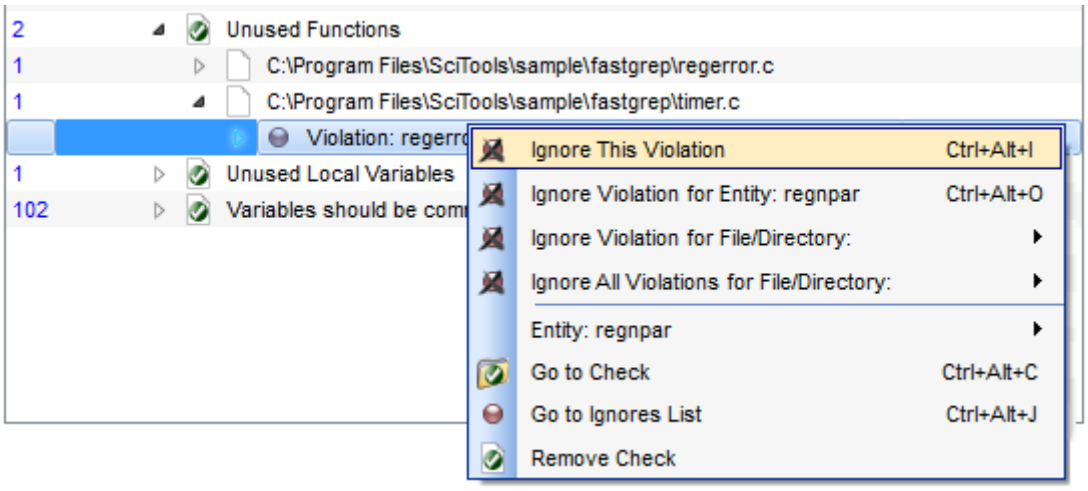
## Ignoring Checks and Violations

A number of options let you ignore some CodeCheck violations in all or part of your project. For example, you might want to ignore violations in third-party code used by your project.

If many violations are detected, you might want to look at the individual checks in the **Checks** tab to see if you can set options to control the sensitivity of the checks. For example, for the “Magic Numbers” check, you can specify that bitfields can be set to fixed values and you call allow exceptions for values like 0 and 1.

Wherever you see a violation listed in the results, you can right-click and choose to ignore this violation instance, ignore violations of this check for the specified entity, or ignore violations of this check for the current file or a selected directory level within the project. You can also choose to ignore all violations in a specific file or directory. If you

select a directory, violations in all of its subdirectories will be ignored. You can also click the Ignore icon above the results to access the Ignores menu.




When you choose to ignore a violation, you are first asked to confirm that it is OK to ignore multiple violations of the same type in the same file. Next, you are asked if you want to add a note about this ignored violation. If you click **Yes**, you can type text regarding this violation and why it is ignored.

If you are not sure whether you want to ignore the violation, choose **Go to Check** for more information about the check that was violated.

Violations that you choose to ignore are not listed in the Results by File or Result Locator tabs unless you check the **Show Ignored Violations** box. They are highlighted with a pink background if they are shown. The violation totals in the Results by File tab *do not* include the ignored violations. Totals in the Results by Check tab *do* include ignored violations.

If you choose to ignore any violations, you can use the **Ignores List** tab to find and sort ignored violations. Only one item is listed if you have ignored multiple violations of the same type in a file or directory. You can search this list as you would the Result Locator list. See *Filtering the List* on page 157 for details.

The toolbar above the Ignore List lets you perform the following actions:

Add or edit the text note for the selected violation to be ignored. You can also open the Ignored Violation Note dialog from the right-click menu or by pressing the + key.

Remove the text note for the selected ignored violation. You can also remove the note from the right-click menu or by pressing the - key.

Stop ignoring this violation. You can also open the Ignored Violation Note dialog from the right-click menu or by pressing the Delete key.

Export a list of the violations to ignore to a text file. You are prompted to specify the location and filename. The exported file is a comma-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.

Import a list of violations to ignore from a text file. The file must be in the same format exported from the CodeCheck tool.

# Using CodeCheck Configurations

If you have a set of checks you want to use, you can save that list of checks as a “configuration”. Such configurations are stored outside of the project, so that you can use the same CodeCheck configuration with multiple projects.

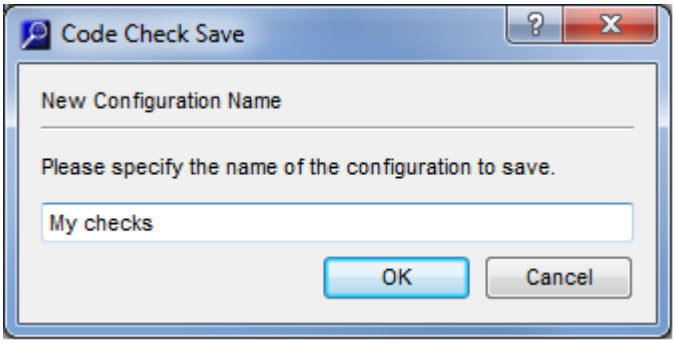
Note that the set of files to be inspected is not saved as part of a CodeCheck configuration. The most recent set of files used is the default.

To save a configuration, follow these steps:

1 In the Checks tab, select the boxes for all the checks you want performed when this configuration is used.

2 Click the **Save** icon next to the Configuration drop-down list.


3 Type a name for your configuration in the dialog, and click **OK**.




You can use a configuration you have saved by selecting it in the **Configuration** drop-down list at the top of the Checks tab.

Another way to run a configuration is from the **CodeCheck > Saved Configurations** menu item, which you can use even if the CodeCheck window is not open.

The **CodeCheck > Re-Run Previous Checks** command runs the most recently selected CodeChecks. If you have made changes to any files, you will be asked if you want to re-analyze the changed files before running CodeCheck.

The **CodeCheck > Analyze Changes and Re-Run Previous Checks** command runs the **Analyze Changed Files** command automatically and then runs the **Re-Run Previous Checks** command.

# Writing CodeCheck Scripts

CodeCheck scripts are special Perl scripts that let you provide custom checks for verifying your team's coding standards. They can be used to verify naming guidelines, metric requirements, published best practices, or any other rules or conventions that are important for your team.

You can develop these scripts using the Understand Perl API along with a set of special functions designed to interact with the Understand CodeCheck interface.

CodeCheck script files have a *.upl extension.

To begin writing your own check, follow these steps:

1. Choose **CodeCheck > Implement Your Corporate Standard** from the menus.
2. In the web page this command takes you to, save the codecheck_template.upl to a file with the same name on your computer.
3. Edit this template file (with a text editor).
4. Modify the name, description, and detailed_description to match what you plan for this check to do. For example, you could use the following descriptions for a check to make sure lines do not exceed a specified length:

```
# Required - Return the short name of the check
sub name { return "Characters per line";}

# Required - Return the short description of the check
sub description { return "Lines should not exceed a set number of characters";}

# Required - Return the long description of the check
sub detailed_description { return "For readability, lines should be limited to a certain
number of characters. The default is 80 characters per line.";}
```

5. Modify the test_language subroutine to test for the desired languages. For example, the following test makes the check apply to C++, Java, and Python. You can look at other scripts in the \conf\plugin\SciTools\Codecheck subdirectory of your installation for more examples.

```
sub test_language {
    my $language = shift;
    return $language =~ /C++|Java|Python/;
    return 1;
}
```

6. If your check should be run on a per-entity basis, return 1 for the test_entity subroutine. If the check should be run only once per file, return 0 for the test_entity subroutine. For example:

```
sub test_entity { return 1;}
```

7. If your check should be run only once per project, return 1 for the test_global subroutine. Otherwise, return 0 for the test_global subroutine. For example:

```
sub test_global { return 0;}
```

8 If your check requires the user to set options, modify the define_options subroutine. For example:

```
sub define_options{
    my $check = shift;
    $check->option->integer("charPerLine","Max Characters per line",80);
}
```

Modify the check subroutine to include the check and to signal a CodeCheck violation reporting the problem. The following example reports filenames that do not begin with a letter character:

```
if ($file->name =~ /^[^a-zA-Z]/){
    $check->violation(0,$file,-1,-1,"File name does not begin with a letter");
}
```

The following example reports lines longer than the specified maximum number:

```
sub check {
    my $check = shift;
    my $file = shift;
    return unless $file->kind->check("file");

    my $maxChar = $check->option->lookup("charPerLine");
    my $lineNum = 1;
    foreach my $line (split('\n',$file->contents)){
        my $length = length($line);
        if( $length > $maxChar){
            $check->violation($file,$file,$lineNum,-1,
                   "$length characters on line(Max: $maxChar)");
        }
        $lineNum++;
    }
}
```

9 Verify that your Perl syntax is correct. The easiest way to do this is to open a command line and run the Perl application that ships with Understand: uperl -c mysample.upl.

To learn more, you may want to read about Understand's Perl API. Browsing the CodeCheck scripts that are shipped with Understand can also be very beneficial. If you have questions about CodeCheck scripts, the SciTools website can be a great place to ask them.

## Installing Custom Scripts

You can install a script in *Understand* by dragging and dropping the script file into the *Understand* window. You will be asked if you want to install the plugin. Click **Install**.

When you install a custom check, you will see a message that identifies the directory where the check was installed. For example, C:\Users\YourName\AppData\Roaming\SciTools\plugin\Codecheck. You can install future checks by copying files directly to this directory.

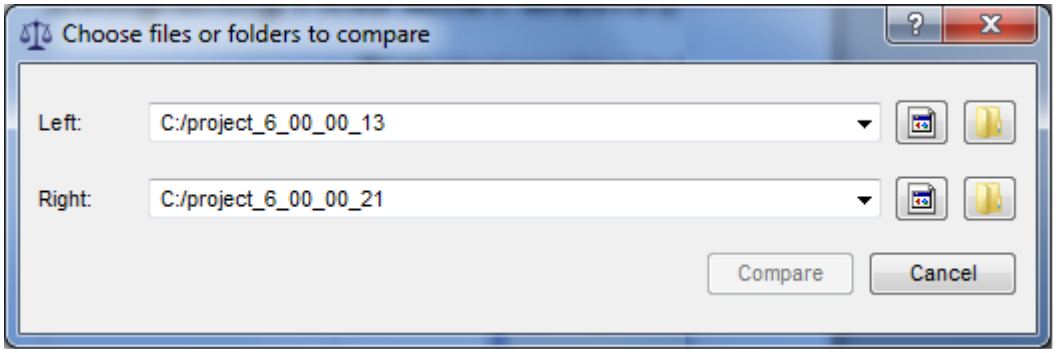
# Chapter 12 Comparing Source Code

This chapter is explains the source-code comparison features provided by *Understand*.

This chapter contains the following sections:

# Comparing Files and Folders

*Understand* provides a tool for comparing files and folders. To open this tool, choose **Tools > Compare > Compare Files/Folders** from the menus.




In this dialog, select a file or folder for the left and right comparison. Both sides should be similar files or similar folders. Click the file button to browse for a file; click the folder button to browse for a directory.

Subdirectories of the directories you choose are also compared.

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**.

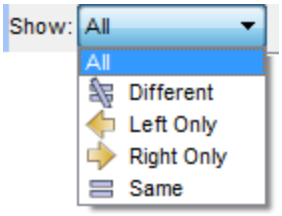

When begin the comparison, the status bar at the bottom of the *Understand* window shows what is being compared.

The code comparison section of the results window is described on page 301. If you are comparing folders, there is an additional top section that lets you see the folder-level and file-level differences and select individual files whose contents you want to compare.

The comparison uses the following file and folder icons.

Same in both versions
Only in left version
Only in right version
Different in left and right

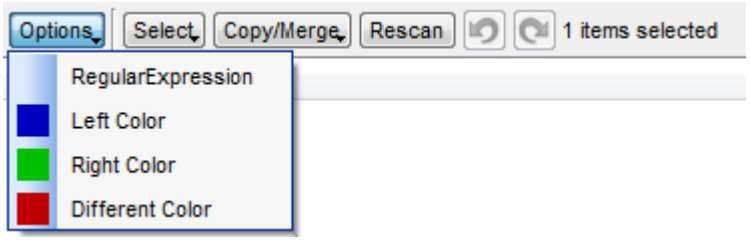
By default, all files and folders are listed. You can use the **Show** drop-down to choose whether to restrict the list to showing only:



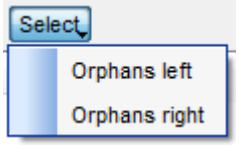

- **Different:** Show files and folders that either exist in only one version or are different in the two versions.
- **Left Only:** Show files that are contained in the left version only. All different folders are shown because some may contain files that are only in the left version.
- **Right Only:** Show files that are contained in the right version only. All different folders are shown because some may contain files that are only in the right version.
- **Same:** Show files that are the same in both versions. All folders are shown because some that are different may contain files that are the same.

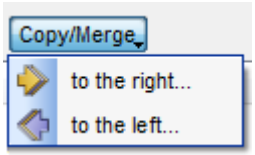
The **Filter** field lets you type characters you want to match in the directory path or filename. For example, “sim” matches any folders or files with “sim” in their names. All files within folder that match the **Filter** (and the **Show** drop-down setting) are shown. Filtering occurs as you type. If you want to use regular expressions, enable that option by clicking the **Options** button to see that menu. Wildcards are not recognized.

You can change the colors use for folder and file names that differ in the two versions by choosing a color from the **Options** menu and selecting a new color in the color picker.



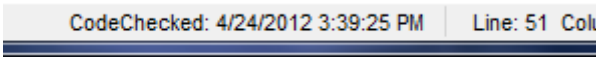

You can highlight all items that exist in only the left or right version. To do this, first right-click on the file list and choose **Expand All**. Then click the **Select** button and choose either **Orphans left** or **Orphans right**. You will see a warning that some items may have been skipped; this applies only if you did not use **Expand All**.




You can copy folders and files from one side to the other. The copied items overwrite any items with the same names. To copy, first select the items you want to copy. (To copy a folder and its contents, select the folder *and* all the folders and files it contains.) Then click the **Copy/Merge** button and choose either **to the right** or **to the left**. This opens the Copy Files dialog, which lists the files or folders to be copied. If the list is correct, click **OK**. Click the  icon to undo your last copy/merge change. Click the  icon to redo you last undo.

The code comparison section of the results window is described on page 301. 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.



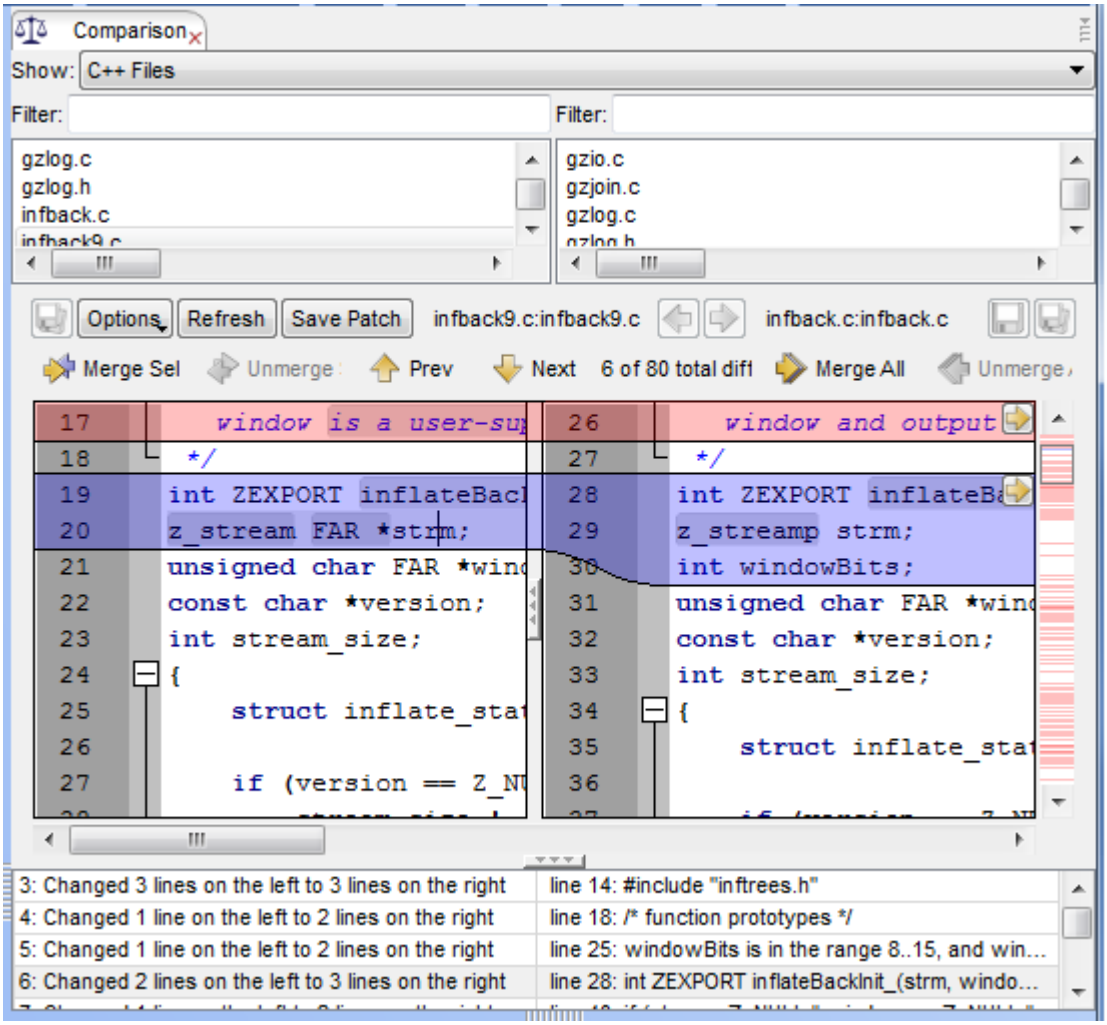

If you modify one or more files that are listed in the comparison area and then switch to the comparison of another file, you are asked whether you want to save and recompare the files.

You can save changes you make to files in the file and folder comparison. If you have modified a file on the right, you can click the **Save** icon to save that file to its existing location. You can use the **Save As** icon on either the left or right to save a file to a different location. After you save changes, you can click **Rescan** to compare the modified files and folders.

# Comparing Entities

You can compare two entities by choosing **Tools > Compare > Compare Entities** from the menus. You see the Comparison window.

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 middle and lower portions are similar to file comparisons (page 301).

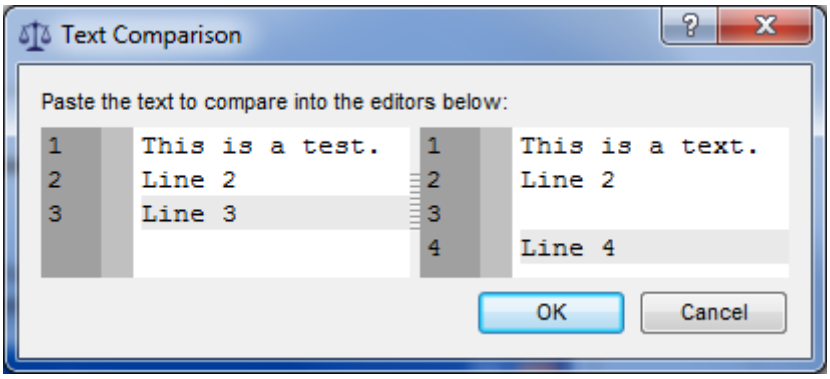
At the top of the comparison is an entity filter (page 129). Select a type of entity in the **Show** drop-down. Then use the lists to select two entities you want to compare. The **Filter** fields let you type a string you want to match anywhere in the entity name. Filtering occurs as you type. Wildcards 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. You can still use the Save Patch button to create a patch file in “unified format”.

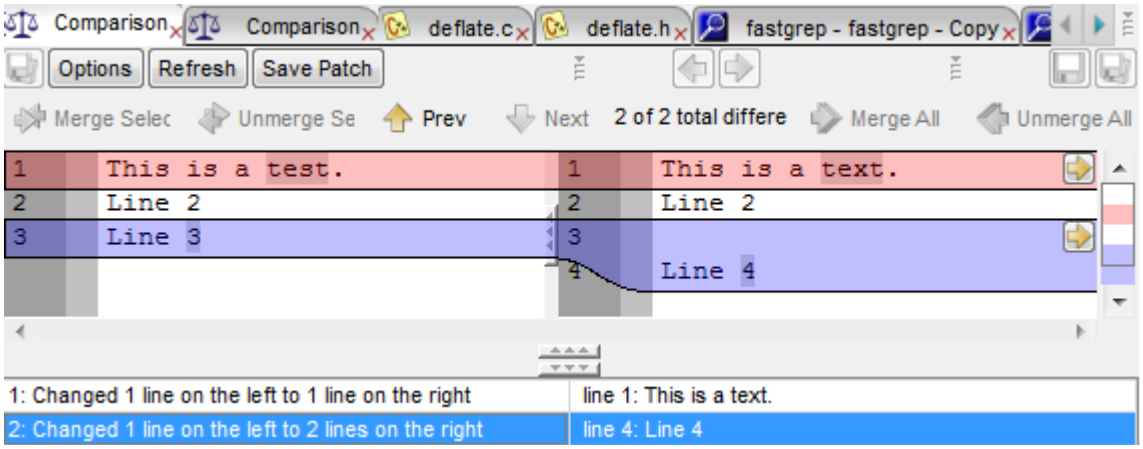
If you are comparing a file, you can merge changes and use the **Save** and **Save As** buttons for the version of the file on the right.

## Comparing Text

You can compare text that you paste into a window by choosing **Tools > Compare > Compare Arbitrary Text** from the menus. You see a window like this:




Paste the before and after text you want to compare into the left and right sides. Then, click **OK** to see the comparison.




The text comparison is similar to the comparison between two entities. You can merge and unmerge differences, but cannot save files.

Click the fold icon to hide or view the patch file syntax and/or the list of differences.

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**.

# Exploring Differences

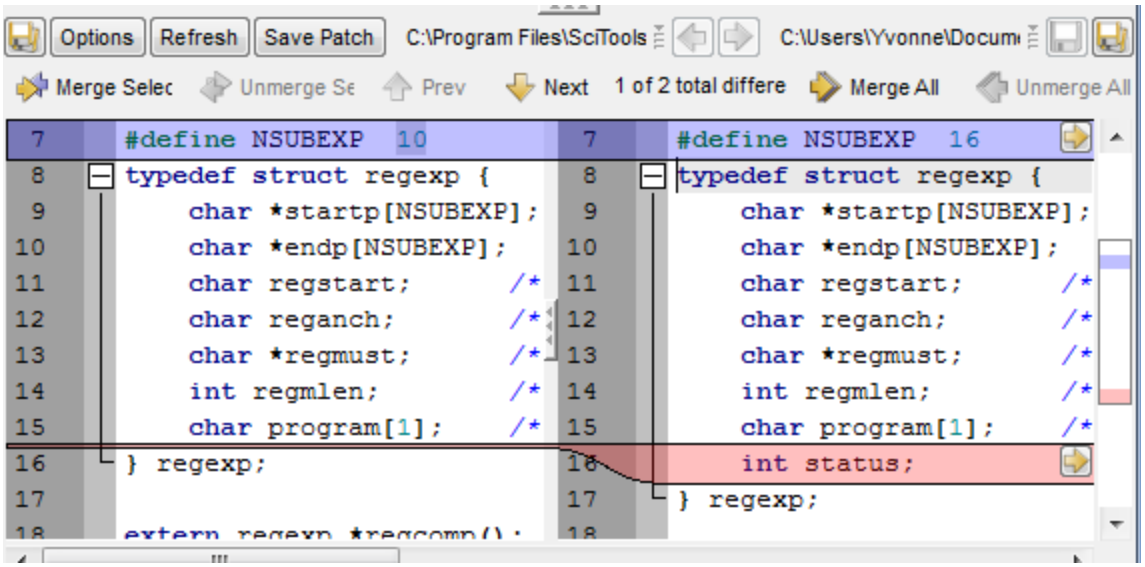
When you compare items, you see a comparison window. Depending on what you are comparing, you see several of the following areas to help you navigate the differences:

- **Changed Entities:** This area lets you select files or entities to compare. It differs depending on what you are comparing.
- **Code Comparison:** This area allows you to examine the differences in the code. See page 301.
- **Patch File:** The patch area shows the patch file syntax to convert from the left version to the right version. See page 304.
- **Difference List:** This list allows you to select individual differences between two versions. See page 304.

The small  fold icon between the areas allows you to close and reopen areas to make more space for the other areas. If you point your mouse to the right or left of either fold icon, you see the pane resize mouse cursor, which allows you to resize the areas as needed.

## Code Comparison

The Code Comparison area shows individual differences between versions of an entity. The display is similar to that of common differencing tools.

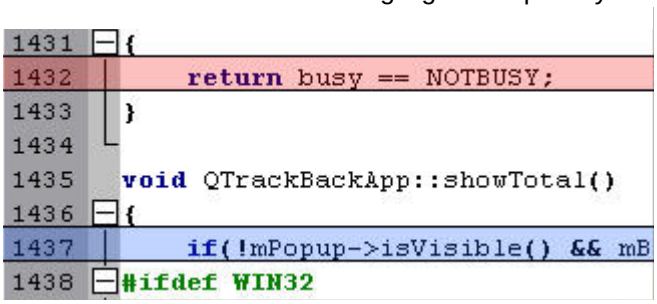



The left side shows the code from the first item you are comparing; the right side shows the code from the second item. The entity path is shown just above the code.

Scrolling of the two versions is synchronized horizontally and vertically. The scrollbar shows the location and size of changed sections of code using the comparison colors.

For certain languages that *Understand* understands—such as C code—you can click the + and - signs in the code to expand and compress code constructs such as if and else statements, functions, extended comments, and so on.

The currently selected difference is highlighted in blue (or bluish purple on some screens) by default. Other differences are highlighted in pink by default.




You can use the small fold icon (like the one shown here) between the two code versions to hide the left version of the code temporarily. Or, click the  right arrow next to a code change to do the same thing.

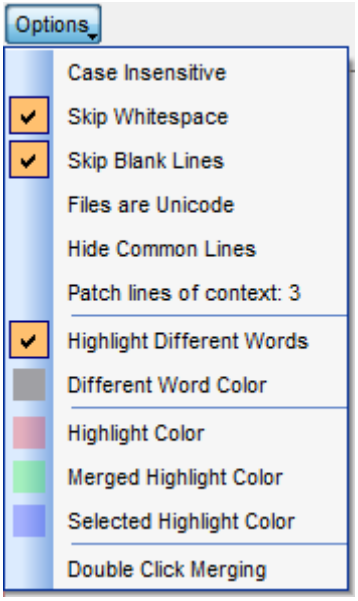
You can edit the source code if you like in the right version of the files. You cannot save code directly to a file. Instead, you can use the Save Patch button to save a patch file or you can copy and paste code with merged differences and edits into another application.

You can select text and copy it to the clipboard. To select text, use the mouse or your keyboard. To select all, press Ctrl+A or right-click and choose **Select All**. To copy text to the clipboard, press Ctrl+C or right-click and choose **Copy**.

As always, right-click on any entity name or other text in the code to see lots of item-specific options in the context menu.

The status bar at the bottom of the window shows your line location in the source code where you last clicked.

The toolbar at the top of the Code Comparison area contains the following controls:




Options **Options button:** Use the Options drop-down to set the following options:

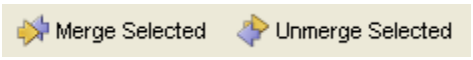
- **Options->Case Insensitive:** By default, changing the case of a letter is not treated as a difference. For example, if you change “a” to “A”, the Difference List shows “No Differences” if that was the only change.
- **Options->Skip Whitespace:** By default, changing the number of spaces or tabs is not treated as a difference. The Difference List shows “No Differences” if only whitespace was changed. You can change this behavior by toggling this option off.
- **Options->Skip Blank Lines:** By default, a different number of blank lines is treated as a difference. You can change this behavior by toggling this option on.
- **Options->Files are Unicode:** By default, differences are reported only for ASCII files. If *Understand* says “File is Binary”, use this command to turn on Unicode file handling.
- **Options->Hide Common Lines:** By default, all lines in both files are shown. If you check this option, most lines that are the same in both versions are hidden in the left (older) version.
- **Options->Patch lines of context:** The patch area shows the patch file syntax to convert from the left version to the right version. By default, 3 matching lines are shown around a change to provide context. You can choose this option and change the number of lines in the Patch Lines of Context dialog.
- **Color choices:** These options let you change the highlighting in the code comparisons. The Different Word color is an overlay that is combined with the other highlight colors as appropriate.
- **Options->Double Click Merging:** A shortcut for merging is to double-click on a difference in the code. This works only if you enable it here.

The Case Insensitive, Skip Whitespace, and Files are Unicode options are not available if you have made a change to a file.

Refresh **Refresh button:** You can use the Refresh button to update the Difference List at the bottom of the Change Results. This list may become out-of-date if you merge differences or edit the file directly.

Save Patch **Save Patch button:** You can use the Save Patch button to create a patch file in “unified format” (or unidiff). This patch file can be used with the Unix patch tool and other similar programs.

Prev Next Click the **Prev** and **Next** buttons above the Code Comparison area to jump to another difference between the entities.




You can merge differences into the version of an entity shown on the right. You cannot save code directly to a file. Instead, you can save a patch file or copy and paste code with merged differences and edits into another application. To merge differences, follow these steps:

1. Select a difference in the code or by selecting a line in the Difference List area.
2. Click the **Merge Selected** button. This copies the older (left) version of this difference to the current (right) version of the code. (If you change your mind, click **Unmerge Selected**.)
3. Click the **Prev** or **Next** button to move to another difference and repeat the previous step.

In the Difference List, merged differences are shown in blue italics. In the code, differences you have merged are highlighted in green. (The currently selected difference is still highlighted in blue/purple, even if it has been merged.)

A shortcut for merging is to double-click on a difference in the code if you have enabled **Double Click Merging** in the Options drop-down.


If you know you want to merge all of the differences, click **Merge All**. If you want to undo all merges you have made, click **Unmerge All**.

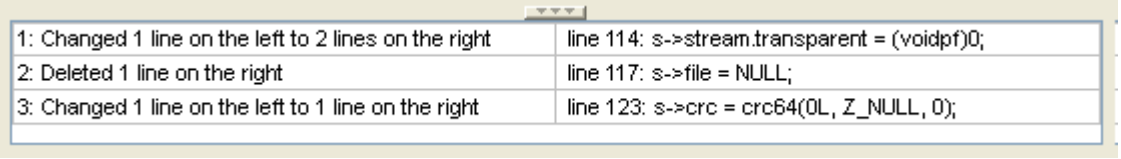
**Patch File**

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

The **Patch lines of context** command in the **Options** button menu lets you adjust the number of unchanged lines shown around a difference.

**Difference List**

The Difference List area shows a list of the differences in the code shown in the Code Comparison area.




In the Difference List, merged differences are shown in blue italics.

You can hide the Difference List portion of the results by clicking the small  fold icon below the area. This makes more space for the Code Comparison area.

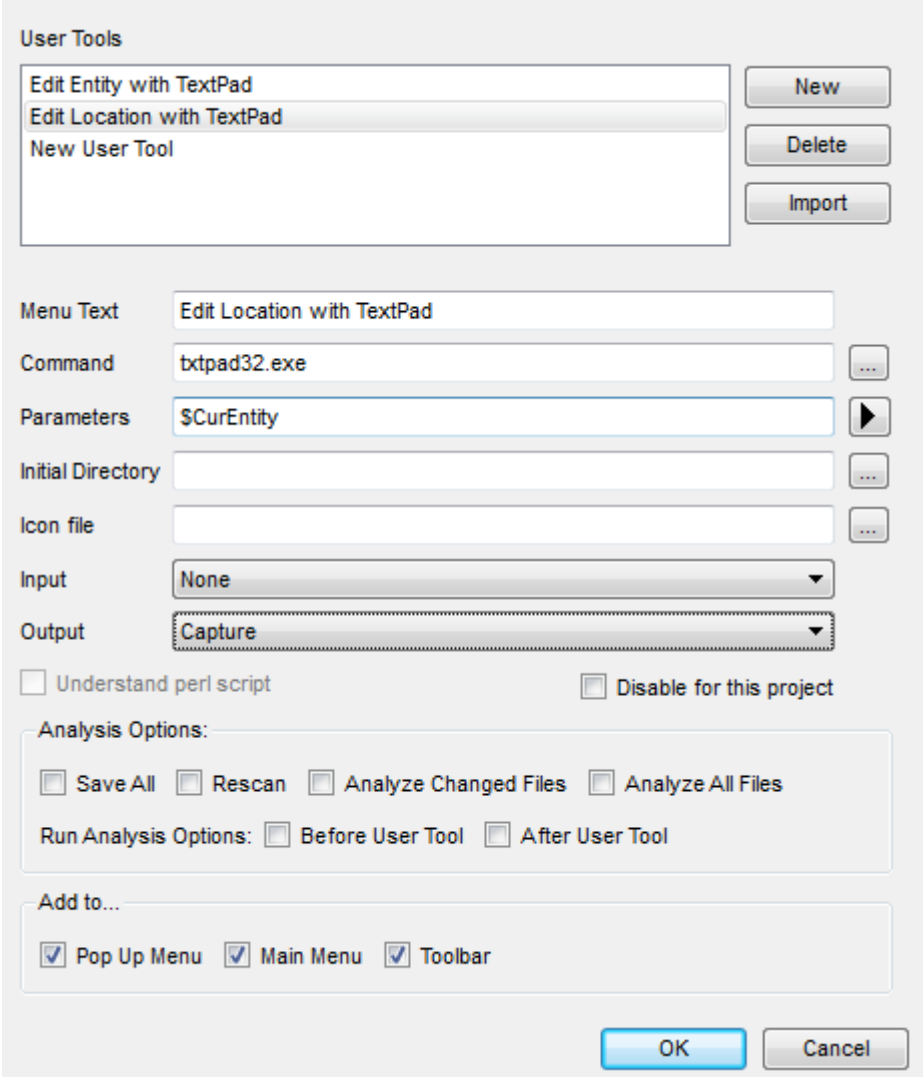
# Chapter 13 Running Tools and External Commands

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:

# Configuring Tools

Select **Tools > User Tools > Configure** from the menus to open the Tool Configurations dialog, where you can configure external tools such as source code editors for use within *Understand*. External tools configured for use will be available for context-sensitive launching. The Tool Configurations dialog provides a number of categories that determine how they are launched.




First, use the **User Tools** category of the Tool Configurations dialog to define a command and parameters as follows:

1 Click **New**.

2 In the **Menu Text** field, type the name you want to appear in *Understand* menus for this tool. You can use variables in the Menu Text. For example, you can use $CurEntity to put the name of the currently selected entity in the tool name. See *Variables* on page 308 for a full list of variables.

3 If the tool you use is on your executable search path, simply type its name in the **Command** field. If not, use the **Browse** button to specify the full path to its executable.

4 In the **Parameters** field, specify parameters that need to be passed on the tool's command line. See *Variables* on page 308 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 **Pop Up Menu** checkbox adds the tool to the right-click context menu. The **Main Menu** checkbox adds the tool to the **Tools > User Tools** submenu. The **Toolbar** checkbox adds the tool’s icon to the toolbar.

To edit settings for an existing tool, select it in the list and make changes as needed. Click **OK** to save your changes. If you want to remove a tool, select it and click **Delete**.

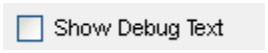
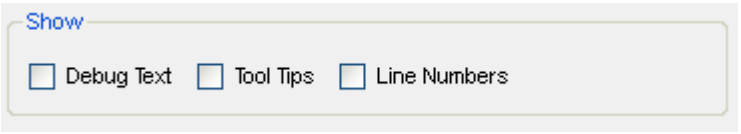
For information about using the **Import** button, see *Importing and Exporting Tool Commands* on page 316.

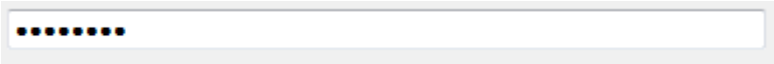
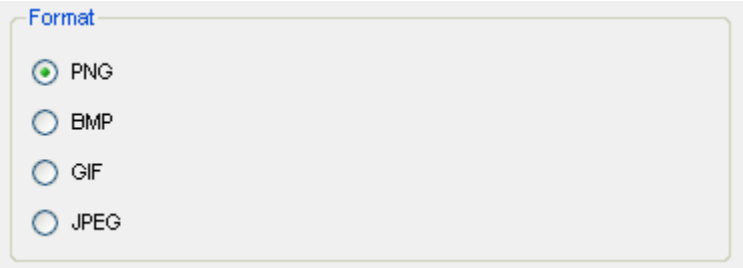
## Variables

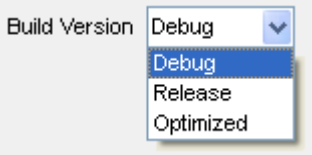
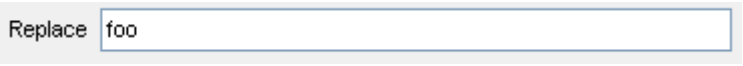
Variables beginning with $Cur are current position variables that apply only from a Source Editor window. Variables beginning with $Decl are declaration variables that apply only when an entity with a declaration is selected. Variables beginning with $Prompt display a dialog to ask the user for some 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. |
| $PromptForCheckBox | Prompts user for a true/false value required by the command. A 0 (unchecked) or 1 (checked) is passed to the command in place of this variable. This variable should be followed by a string to be displayed as text next to the checkbox. For example, `$PromptForCheckBox "Show Debug Text"` displays the following prompt   |
| $PromptForCheckBoxGH | Prompts user with a series of checkboxes displayed in a horizontal group. For example, `$PromptForCheckBoxGH "Show=Debug Text;Tool Tips;Line Numbers"` displays the following prompt. The label (“Show” in this example) is optional. A semicolon must be used to separate items. The text strings for all checked items (separated by spaces) are passed to the command.   |
| $PromptForCheckBoxGV | Prompts user with a series of checkboxes displayed in a vertical group. For example, `$PromptForCheckBoxGV "Show=Debug Text;Tool Tips;Line Numbers"` displays the following prompt. The text strings for all checked items (separated by spaces) are passed to the command.   |

| Variable | Description |
|---|---|
| $PromptForDir | Prompts user to select a directory and passes the full path as a string. For example, `$PromptForDir "Directory Path=$CurProjectDir"` displays the following prompt with the current project directory as the default. The "..." button opens the standard directory selection dialog for your operating system:   |
| $PromptForFile | Prompts user to select a file and passes the full path as a string. For example, `$PromptForFile "Filename=$CurFile"` displays the following prompt with the current source file as the default. The "..." button opens the standard file selection dialog for your operating system:   |
| $PromptForPassword | Prompts user for a password. Characters typed in this field are obscured.  |
| $PromptForRadioBoxGH | Prompts user for a selection from a set of options displayed horizontally. For example, `$PromptForRadioBoxGH "Format=PNG;BMP;GIF;JPEG"` displays the following prompt. The text string for the selected item is passed to the command.   |
| $PromptForRadioBoxGV | Prompts user for a selection from a set of options displayed vertically. For example, `$PromptForRadioBoxGV "Format=PNG;BMP;GIF;JPEG"` displays the following prompt. The text string for the selected item is passed to the command.   |

| Variable | Description |
|---|---|
| $PromptForSelect | Prompts user to select from a drop-down box. For example, `$PromptForSelect "Build Version=Debug;Release;Optimized"` displays the following prompt. The text string for the selected item is passed to the command.   |
| $PromptForSelectEdit | Prompts user to select from a drop-down box or edit the text in the box. For example, `$PromptForSelectEdit "Build Version=Debug;Release;Optimized"` displays the same prompt as the example for $PromptForSelect, except that you can edit the string in the box. |
| $PromptForText | Prompts user for a string required by the command. For example, `$PromptForText "Replace=foo"` displays the following prompt and provides a default value. The text provided is passed as a string.   |

In general, the multiple-selection $Prompt variables accept strings of the format `"label=item1;item2"`. Any number of items may be separated by semicolons. The item strings for all selected items (separated by spaces) are passed to the command.

The label is optional except in the cases of $PromptForCheckBox, $PromptForDir, $PromptForFile, and $PromptForText. The default value is optional in the cases of $PromptForDir, $PromptForFile, and $PromptForText.

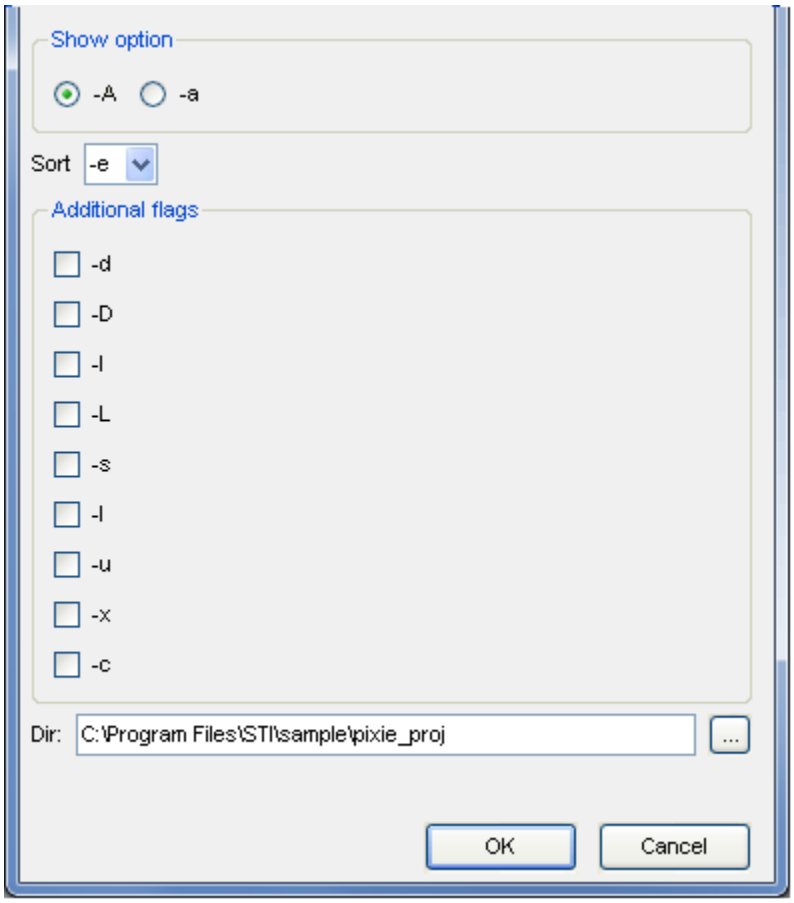
Prompts are processed after the other types of variables, so you can use other variables in the labels and values. For examples, see $PromptForDir and $PromptForFile in the previous table.

In addition, operating system environment variables can be used in prompt syntax. For example, `$PromptForSelect "Dir=$PATH"` presents a drop-down list of all the directory paths in your $PATH definition.

You can optionally provide the item list in a separate file. In that case, the syntax for most $Prompt variables is `label=@fullpath_of_listfile.txt`.

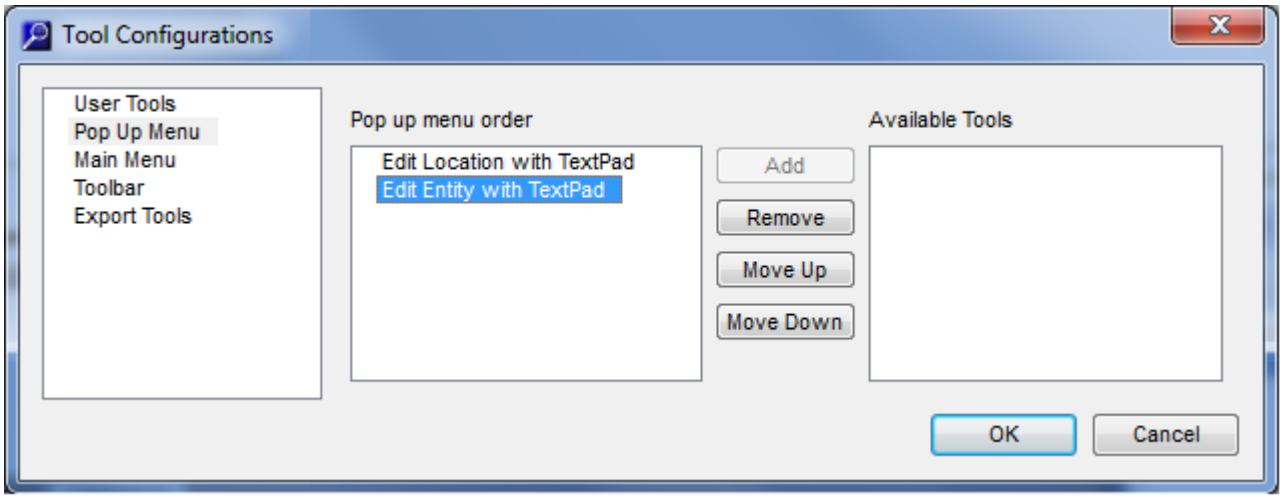
You can combine variables to pass all the parameters needed by a command. All prompts are combined into one dialog. For example if the command is “ls”, you can use the following parameters to create a dialog that lets you select command-line options for the ls command:

```
$PromptForRadioBoxGH "Show option=-A;-a" $PromptForSelect "Sort=-e;-t" $PromptForCheckBoxGV "Additional flags=-d;-D;-l;-L;-s;-l;-u;-x;-c" $PromptForDir "Dir:=$CurProjectDir"
```



# Adding Tools to the Context Menus

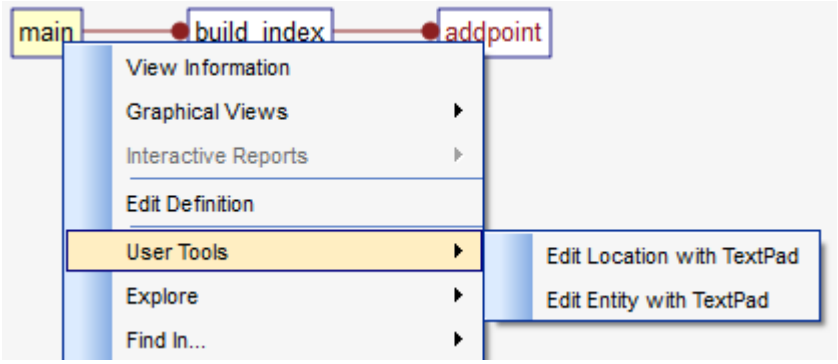
Once a command is defined in the Tools tab, the **Pop Up Menu** category in the Tool Configurations dialog lists user tools that are currently in the context menu on the left and commands you can add to that menu on the right. (Context menus are sometimes called contextual, shortcut, right-click, or pop-up menus.)



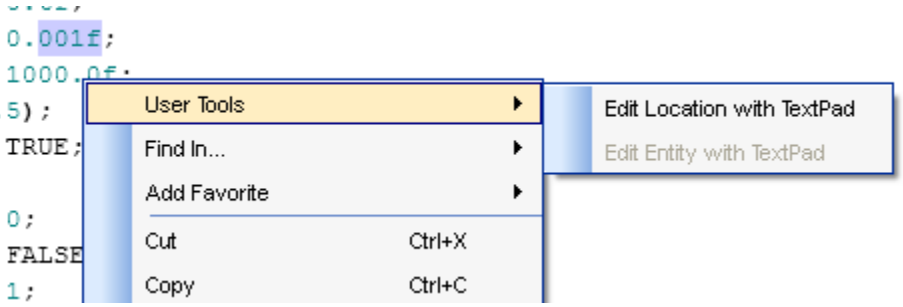

To add a tool to the context menus, select it on the right and click **Add**. To remove a tool from the context menus, select it on the left and click **Remove**.

User tools appear in the context menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.

The following figure shows a context menu for an entity showing the available external tools.

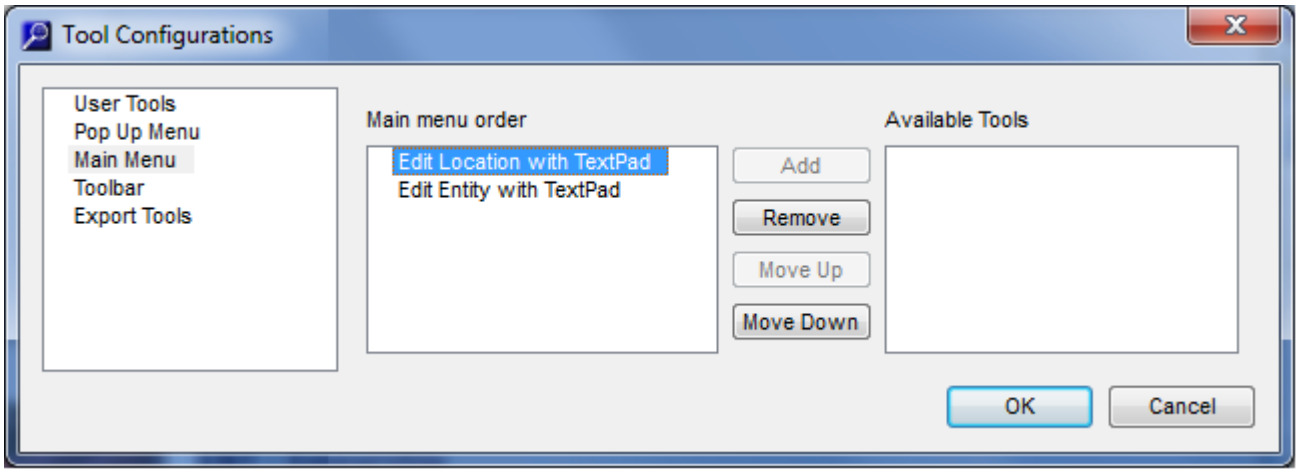


Tools are active or inactive on the context menu based on the context of the parameters provided to the tool. For example, a source editor that specifies $DeclFile as a parameter is selectable from the context menu for any entity where the declaration is known, but will not be active for an undeclared entity or when no entity is selected.



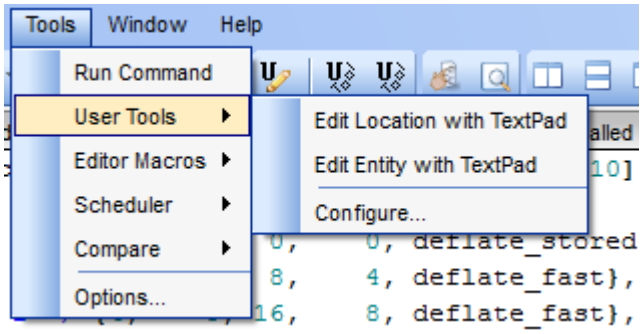

## Adding Tools to the Tools Menu

Once a command is defined in the Tools tab, the **Main Menu** category in the Tool Configurations dialog lists user tools that are currently in the **Tools > User Tools** menu on the left and commands you can add to that menu on the right.



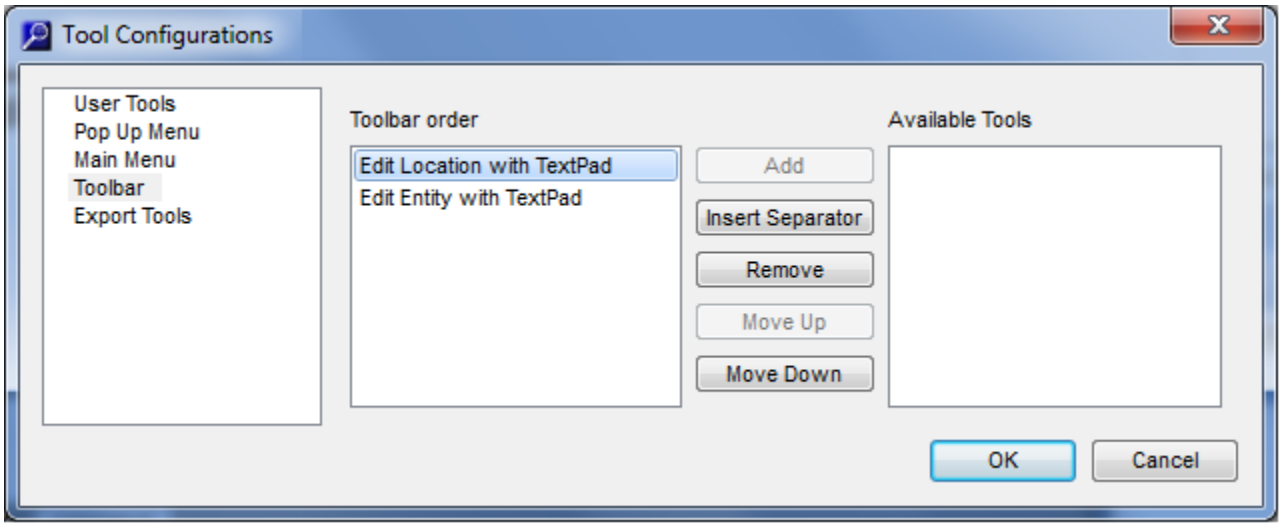

To add a tool to the menus, select it on the right and click **Add**. To remove it from the menus, select it on the left and click **Remove**.

User tools appear on the **Tools** menu in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the tools as desired.




## Adding Tools to the Toolbar

Once a command is defined in the Tools tab, the **Toolbar** category in the Tool Configurations dialog shows user tools currently in the toolbar in the left box and commands you can add to the toolbar in the right box.




To add a tool to the toolbar, select it on the right and click **Add**. To remove it from the toolbar, select it on the left and click **Remove**.

To add a vertical separator to the toolbar, select the item in the Toolbar order box that should have a vertical line to the right of it. Click **Insert Separator** to add "----------" to the list.

Icons for the selected tools appear on the toolbar in the order they are listed in the left column. Use the **Move Up** and **Move Down** buttons to sort the icons as desired.

To change the icon for a particular tool, use the **Icon file** field in the User Tools category.

For example, in the following figure, the first icon is provided by *Understand* to open the Tool Configurations dialog. The second icon is the default icon for a user tool if none is specified.

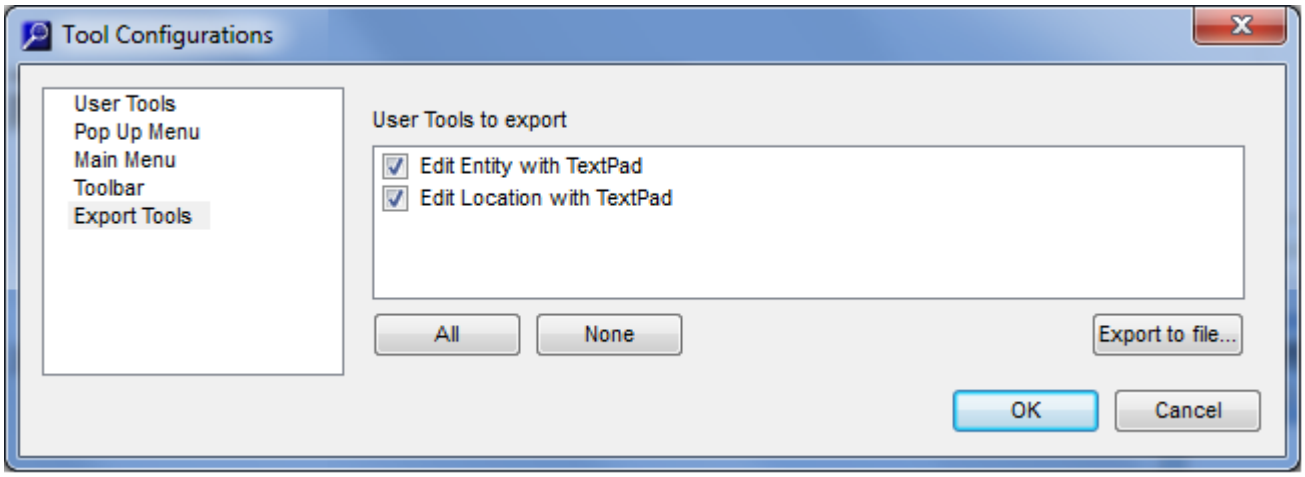
In this toolbar, two icons have been added for user tools. A separator has been added between them.

***Note:*** You can control which icons are visible in the main toolbar by right-clicking on the background of the toolbar and checking or unchecking items for the various toolbar sections.

## Importing and Exporting Tool Commands

You can import and export tool commands from files. This makes it easy to share tool commands with co-workers.

1 To export commands, choose **Tools > User Tools > Configure** from the menus and switch to the Export Tools category. You will see the following dialog.




2 Check the boxes next to commands you want to share.

3 Click **Export to file**.

4 Choose a location and filename for an initialization file (*.ini) that contains the selected user tool information.

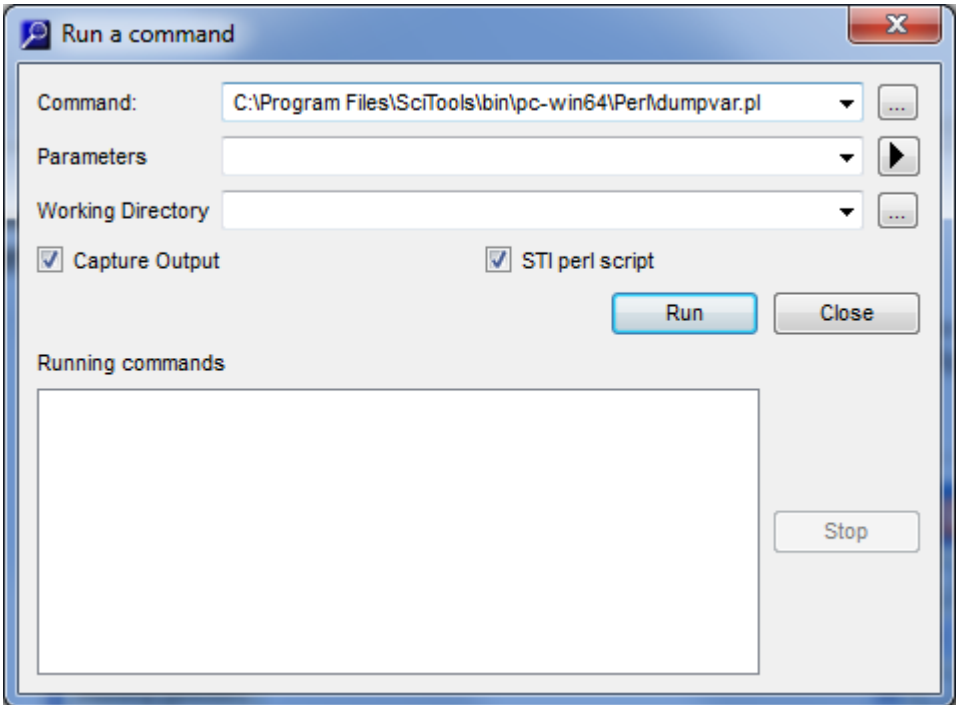
5 Click **Save**.

To import commands, choose the User Tools category in the Tool Configurations dialog and click the **Import** button. Browse for an initialization file created by another *Understand* user and click **Open**. In the Import User Tools dialog, check the boxes next to the tool commands you want to be available in your copy of *Understand*.

# Running External Commands

The **Tools > Run Command** menu item permits any external command to be run directly from *Understand*. Common commands to invoke are compilers, configuration management tools, and Perl programs written using Understand's API.

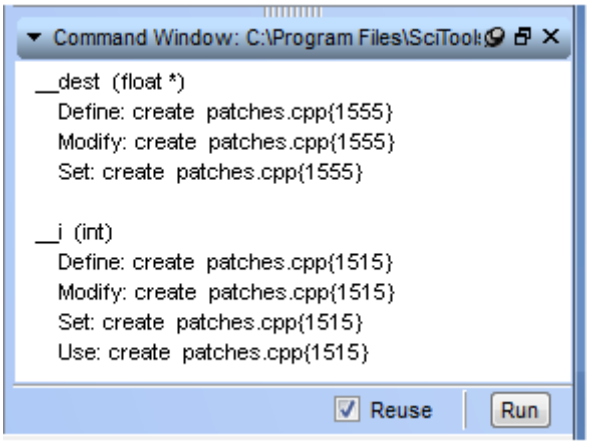
The **Run a Command** dialog looks like this:




To run a command, follow these steps:

1. Type a **Command** or click **...** and browse for a file to run. A number of Perl programs are provided in the *Understand* installation.
2. Type any command-line **Parameters** required by the command. Click the right arrow if you want to select one of the special variables. These are listed on page 308.
3. Click **...** and browse for the directory that should act as the **Working Directory**.
4. If you want the output sent to a window in *Understand*, leave the **Capture Output** box checked.
5. If you are running a Perl script, check the **STI Perl script** box if this is a script provided by Scientific Toolworks.
6. Click **Run**. The output is shown in a Command Window in *Understand* if you checked the **Capture Output** box. Otherwise, the command runs in the background

and output is shown in the **Running Commands** box. You can select a command from this list and click **Stop** to halt the command.

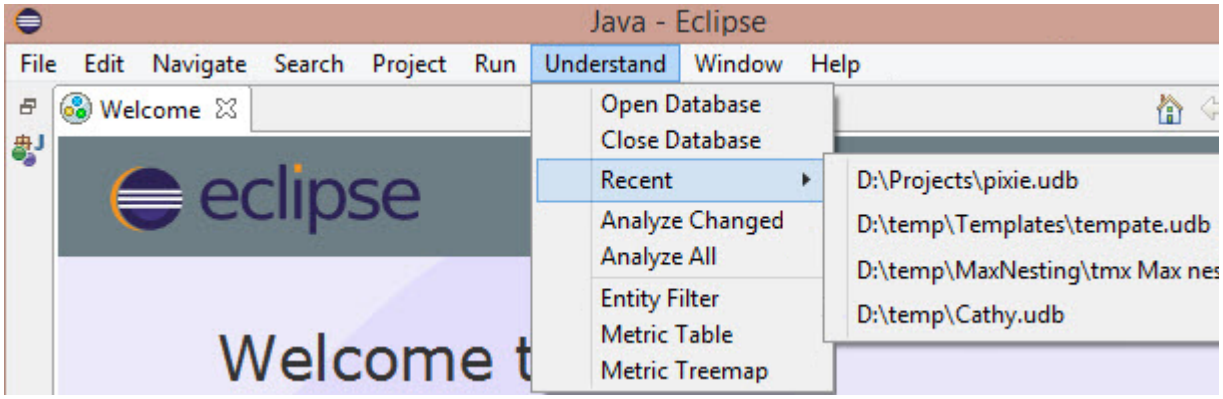



The font used in the Command Window is determined by settings in the Command Window category of the Understand Options dialog, which you can open by choosing **Tools > Options** from the menus. See page 106.

On Unix systems, output to both stdout and stderr are captured.

# Using the Eclipse Plugin

If you use the Eclipse IDE for code development, you can access a number of *Understand* features within the Eclipse IDE by installing the Understand plugin for Eclipse. These features include the Entity Filter, Information Browser, Metrics, Treemaps, Butterfly graphs and Control Flow graphs. See http://scitools.com/eclipse/ for details on installing this plugin.



# Chapter 14 Command Line Processing

This chapter shows how to use *Understand* from the command line. Command line processing can be used in a batch file for automatic re-building and report generation of projects.

This chapter describes the “und” command line, which allows you to analyze sources and create *Understand* databases from the command line. In addition, it allows you to generate metrics and reports.

***Note:*** The “und” commands were standardized in build 571, and the tool should now be much easier to use. Because of the extensive changes, this new version is not backwards compatible with older versions of und. The old und executable has been renamed “undlegacy”. If you have legacy scripts, you should rename the binary run by these scripts in order for them to continue to work.

Most examples in this chapter refer to C/C++ files. However, you can use “und” with any supported language.

This chapter contains the following sections:

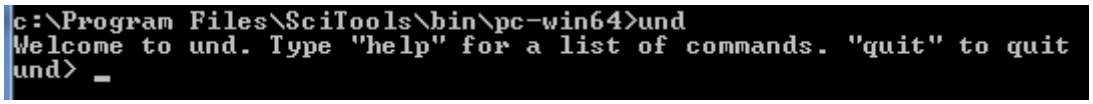
# Using the und Command Line

The command-line tool for creating and building *Understand* databases is *und.*

The *Understand* installer can optionally place the appropriate bin directory in your operating system's PATH definition to simplify running the "`und`" command line. For example the Windows PATH definition might include the C:\Program Files\SciTools\bin\pc-win64 path.

Und can be run in the following modes:

- **Interactive mode:** You enter this mode if you simply type `und` on the command line with no command or text file. While in the interactive shell, settings such as open database are remembered from command to command. This is a good mode to use to test a sequence of commands you want to use in a batch file. You can optionally specify the database to open on the command line to run the interactive shell.




- **Batch mode:** Once you identify a sequence of commands you want to run more than once, you can store them in a text file that you can run in batch mode with the `und process` command. The text file should contain one command per line. Omit the "und" from each command within a batch file. You can use # to begin comments. For example use either of the following commands to run the sequence in the `This.txt` file:

```
und process This.txt
und process This.txt MyDatabase.udb
```

The `This.txt` file might contain commands similar to these:

```
# My command file
c:\projects\MyDatabase.udb
settings -C++MacrosAdd VERSION="Option_2"
analyze     # update database
report      # generate reports
metrics     # generate metrics
```

- **Line mode:** You can specify a single command or set of commands on a single command line. You must specify the database to be used on each command line, because it is not remembered from line to line. Commands are run in the order they appear on the command line. The help and list commands cannot be combined with other commands. For example, you could run either of the following commands to create a database, add files, analyze all, and then exit:

```
und create -db c:\myDb.udb -languages c++ add @myFiles.txt analyze -all
und create -languages c++ add @myFiles.txt analyze -all c:\myDb.udb
```

This is the equivalent of running the following set of commands in interactive mode:

```
create -languages c++ c:\myDb.udb
add @myFiles.txt
analyze -all
```

Alternately, you could run a sequence of line mode commands like the following:

```
und create -languages c++ c:\myDb.udb
und add @myFiles.txt c:\myDb.udb
und analyze -all c:\myDb.udb
```

In general, und commands are case-insensitive.

Und returns a value of 1 if an error occurred.

Und supports the following options that can be added to any command:

| Option | Discussion |
|---|---|
| -db | Specify the database to use |
| -quiet | Print only errors. Do not print warnings or informational messages |
| -verbose | Print extra informational details. |

Und accepts a number of separate commands. A different set of options is supported for each of these commands, and separate help is available for each. For example, for help on the add command, type:

```
und help add
```

The commands supported by und are as follows:

| Option | Discussion | See |
|---|---|---|
| add | Adds files, directories, and roots | page 322 |
| analyze | Analyzes the project files | page 325 |
| codecheck | Runs CodeCheck | page 326 |
| create | Creates an empty database | page 322 |
| export | Exports settings, dependencies, or architectures | page 324 |
| help | Gives help information for a command | page 322 |
| import | Imports project settings and architectures | page 324 |
| list | Lists information about the project | page 323 |
| metrics | Generates project metrics | page 326 |
| process | Runs all the commands in a text file in batch mode | page 320 |
| purge | Purges the database | page 325 |
| remove | Removes files, directories, roots, and architectures | page 323 |
| report | Generates project reports | page 325 |
| settings | Sets project settings and overrides | page 324 |
| uperl | Runs Perl scripts | page 326 |
| version | Shows the current software version | page 322 |

Refer to the sections that follow for details on the commands supported by und.

### Getting Help on Und

Since we do frequent builds of *Understand*, it is likely that this manual may not describe all the options of the “und” command line. We recommend that you check the command-line help. For example, to get details on the report command, type:

```
und help report
```

You can see the version of *Understand* for the und command tool by using the following command:

```
und version
```

### Creating a New Project

Use the `und create` command to create a new database (project). Specify the name of the database either with the -db option or as the last parameter. Any settings allowed with the `settings` command (see page 324) can also be used with `create`. For example:

```
und create -db newDB.udb -languages c++ c#
und create -open_files_as_read_only on newDB.udb
```

For more information, use the following command:

```
und help create
```

### Adding Files to a Project

If you have a small number of source files then it may be easiest to just supply their names to the analyzer using the wildcard abilities of your operating system shell. For example:

```
und -db myproject.udb add \usr\myproject
und -db myproject.udb add file1.cpp file2.cpp
und -db myproject.udb add *.cpp
```

In some cases, there may be too many file locations to use the *-add* technique. A common command line limitation is 255 characters. A directory with hundreds or thousands of files may easily exceed this limit. 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 myproject.udb 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 database to another database. The command format for this is:

```
und add source_project.udb destination_project.udb
```

You can also use the add command to add named roots and Visual Studio projects. Options are available to set the watch behavior, subdirectory adding, the exclude list, file filtering, and languages.

**Exclude strings** are processed relative to the top-level directory passed to the add command, and are applied to all files in all subdirectories. The exclude strings are internally processed as follows:

1. Separate the exclude string into the list of wild cards based on spaces, commas, and semicolons.
2. For each separate exclude string, replace forward and back slashes with the pattern [/\\], which matches either slash.
3. Prepend the absolute path of the top-level directory to the wild card, ensuring that [/\\] separates the path from the initial wild card.
4. Compare the wild card to both the file short name and file long name for every file below that top-level directory. The comparison is done with QRegExp wild card matching.

Named roots definitions on the “und” command line have the highest precedence. The next precedence is named roots defined as environment variables at the operating system level, and finally by named roots defined in the *Understand* project configuration. See page 107.

For more information, use the following command:

```
und help add
```

### Removing Items from a Project

Use the `und remove` command to remove files, directories, Visual Studio files, named roots, and architectures from a project.

Unless there is a name conflict, the type of item to be removed is automatically detected by und. If there is a conflict, the command defaults to deleting the directory with the specified name. You can use the -file, -vs, -root, and -arch options to override this default.

For example:

```
und remove someFile.cpp myProject.udb
und remove C:\SomeDirectory myProject.udb
und -db myProject.udb remove vs1.vcproj vs2.vcproj
und remove -file main.c myProject.udb
```

For more information, use the following command:

```
und help remove
```

### Getting Information about a Project

Use the `und list` command to list file, setting, architecture, or named root settings in a project. For example:

```
und list -tree files myProject.udb
und list settings myProject.udb
und list arches myProject.udb
und list roots myProject.udb
```

There are a number of options for listing settings for the project. You can list all settings, language-specific settings, report settings, metric settings, include directories, macro definitions, and more. For example:

```
und list -override f1.cpp f2.java settings myDB.udb
und list -override @listfile.txt myDB.udb
und list -metrics -reports settings myDB.udb
und list -all settings myDB.udb
und list -lang C++ -macros -includes settings myDB.udb
und list -lang fortran settings myDB.udb
```

For more information, use the following command:

```
und help list
```

### Modifying Project Settings

Use the `und settings` command to modify the settings in a project. You can find the names for each setting by using the following command:

```
und list -all settings myProject.udb
```

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 myProject.udb
```

In general, setting names are the same as the field name in *Understand*, but with spaces omitted. For example:

```
und settings -ReportDisplayCreationDate on myProject.udb
und settings -ReportFileNameDisplayMode full myProject.udb
und settings -ReportReports "Data Dictionary" "File Contents" myProject.udb
und settings -C++MacrosAdd MYLONG="Long Text" myProject.udb
und settings -ReportNumberOfPages 250 myProject.udb
```

For more information, use the following command:

```
und help settings
```

### Importing into a Project

Use the `und import` command to import project settings or architectures from an XML file. In general, you might use this command when creating a new database to import setting that you have exported from another database.

For example:

```
und import settings.xml myNewProject.udb
und import -arch myArch.xml myProject.udb
```

For more information, use the following command:

```
und help import
```

### Exporting from a Project

Use the `und export` command to export project settings, architectures, or a list of dependencies to an XML file.

For example, this command exports project settings to an XML file that you can use with the `und import` command:

```
und export toHere.xml myProject.udb
```

This command exports architectures to an XML file that you can use with the `und import` command:

```
und export -arch "Calendar" toHere.xml myProject.udb
```

These commands export file, architecture, and class dependencies to a CSV, matrix, or Cytoscape file. Several options are available to control the output of dependencies.

```
und export -dependencies file csv output.csv myProject.udb
und export -dependencies class matrix output.csv myProject.udb
und export -dependencies arch myArch csv output.csv myProject.udb
und export -dependencies -col refs -format short file csv out.csv myDB.udb
```

For more information, use the following command:

```
und help export
```

### Analyzing a Project

Use the `und analyze` command to run (or rerun) the project analysis.

When you analyze a project, you have several options. You may re-analyze all files with the -all option (the default), only files that have changed with the -changed option, or a list of files with the -files option. For example:

```
und analyze myProject.udb
und analyze -files @someFile.txt
und -db myProject.udb analyze -rescan -changed
und analyze -files file1.cpp file2.cpp myProject.udb
und -db myProject.udb -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 database, leaving only the project definition. This significantly shrinks the udb file size, which you may want to do before sharing the file or backing it up. Running the analyze command will repopulate the project. For example:

```
und purge myProject.udb
```

For more information, use the following command:

```
und help analyze
```

### Generating Reports

Use the `und report` command to generate reports for the project. This command uses the current report settings, which can be viewed by using the `und list` command (see page 323), and changed using the settings command (see page 324). For example:

```
und list -reports settings myProject.udb
und report myProject.udb
```

### Generating Metrics

Use the `und metrics` command to generate metrics reports for the project. You can generate project metrics (the default), architecture metrics, and the HTML metrics report. For example:

```
und metrics myProject.udb
und metrics -arch myArch myProject.udb
und metrics -html arch1 arch2 c:\temp myProject.udb
```

This command uses the current metrics settings, which can be viewed by using the `und list` command (see page 323), and changed using the settings command (see page 324). For example:

```
und list -metrics settings myProject.udb
```

For more information, use the following command:

```
und help metrics
```

### Using CodeCheck

Use the `und codecheck` command to run the CodeCheck tool on the project and print the log to the screen. You need to provide the name of a CodeCheck configuration file and an output directory for the reports. For example:

```
und codecheck config.ini C:\temp myProject.udb
```

You can create a CodeCheck configuration file as described in *Using CodeCheck Configurations* on page 292.

Options are provided to specify which files to run the CodeCheck configuration on, whether to show ignored violations, whether to flatten the directory tree, and whether to generate HTML output in addition to the default CSV output. You can also export the list of checks performed and the ignored violations to a file without running the CodeCheck configuration.

For example, the following command runs the specified configuration file on the files listed in filelist.txt and generates both the HTML and CSV versions of the results:

```
und codecheck -html -files filelist.txt config.ini C:\temp myProject.udb
```

For more information, use the following command:

```
und help codecheck
```

### Running Perl Scripts

Use the `und uperl` command to run Perl scripts from the command line. For example, the following command would run the myScript.pl file with the arg1 space and arg2 arguments passed to Perl:

```
und uperl myScript.pl -quiet "arg1 space" arg2 myProject.udb
```

For more information, use the following command:

```
und help uperl
```

Note that the und uperl command does not support any graphical uperl commands, such as $ent->draw.

## Creating a List of Files

Where a command accepts a @lisfile.txt for an option, the file must contain one item per line. Full or relative paths may be used. Relative paths are relative to the current directory. A # sign in the first column of a line in the file indicates a comment. If an item has a definition, for example a macro definition, the macro name and its value must be separated by an = sign. For example, *DEBUG=true*.

On Unix here are a couple ways to create such a file:

- Use the 'ls' command, as in:

```
ls *.c *.h > my_project.txt
```

- Use the 'find' command to recurse subdirectories, as in:

```
find . -name "*.c *.h" -print > my_project.txt
```

In a Windows command shell:

- Use the dir command with the /b option:

```
dir /b *.c *.h > my_project.txt
```

- Use the /s option to recurse subdirectories, as in:

```
dir /b /s *.c *.h > my_project.txt
```

# Using the understand Command Line

The *Understand* GUI is launched by the “understand” executable. Normally, you launch this using the shortcuts provided by the installation. If you like, you can modify this using the following command-line syntax.

```
understand [file_1 ... file_n] [-options]
```

Any filenames listed on the command like are opened along with The *Understand* GUI. For example:

```
understand source.c source.h -db myproject.udb
```

The available command-line options (also called command-line switches) are as follows:

| Option | Discussion |
|---|---|
| -contextmenu *filename* [-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 *path* | Set the current working directory to "path". This takes precedence over the last working directory for a project loaded with -db or -lastproject. |
| -db *filename* | Open the project specified by the filename. |
| -diff *left_path right_path* | Compare the two specified files or folders as with the Tools > Compare command within *Understand*. |
| -existing | Detects any running instance of *Understand* and sends the command line to that instance. |
| -importusertools *importfile*.ini | 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 *Understand*. If you use the operating system to open a file with an extension that opens *Understand*, by default that file opens in any existing instance. You can use this command-line option to force a new instance to open. |
| -noproject | Ignore all project load requests on startup. (This also clears the "Open Last Project" application setting.) |
| -no_splashscreen | Use this option to skip the splash screen when *Understand* starts up. This setting is stored until you change it in the Tools > Options dialog. |
| -quiet_startup | Use this option to disable all dialogs and splash screens shown during startup. |
| -SlowConnect | Allow for a longer timeout period when communicating with the license server. |

| Option | Discussion |
|---|---|
| -visit *filename* [*line# column#*] | 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 *Understand* to block while waiting for the other instance to finish the given command. |

# Using Buildspy to Build Understand Projects

Buildspy is a tool that allows gcc/g++ users to create an *Understand* project during a build. Buildspy gets lists of files, includes, and macros from the compiler. This can save time and improve project accuracy.

To use Buildspy, follow these steps:

1. Change the compiler command from `gcc/g++` to `gccwrapper/g++wrapper` in your makefile or build system.
2. Either add the <*SciTools*>/bin/<*platform*>/buildspy directory to your PATH definition or use the full path to the gccwrapper/g++wrapper executables in your makefile or build system. On Linux, this might be the /SciTools/bin/linux32/buildspy directory. On Windows, this might be the C:\Program Files\SciTools\bin\pc-win64\buildspy directory.
3. Perform a `make clean` or equivalent command. (This step is optional; Buildspy can be run incrementally to update only the files it is run on.)
4. From the directory where your make file is located, run a command similar to the following:

   ```
   buildspy -db path/name.udb -cmd <compile_command>
   ```

   For example:

   ```
   buildspy -db ~/Documents/MyProject.udb -cmd make
   ```
5. When the build has finished running, open the *Understand* project that was created and choose **Project > Analyze All Files**.

The `buildspy` command sends information from gccwrapper/g++wrapper to Buildspy, which allows it to build a complete *Understand* project. The wrappers then call the corresponding compiler.

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.

# Chapter 15 Quick Reference

This chapter lists of commands provided by *Understand*. These lists provide cross references to information about these commands in this manual.

Since new versions of *Understand* are provided frequently, these lists are subject to change.

This chapter contains the following sections:.

# File Menu

The **File** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| New > Project | page 35 |
| New > File | page 181 |
| Open > Project | page 21 |
| Open > File | page 181 |
| Close *project_name* | page 21 |
| Save *project_name* As | page 39 |
| Export to Image File | page 276 |
| Save Configuration | page 292 |
| Save *filename* | page 170 |
| Save *filename* As | page 170 |
| Save All | page 170 |
| Page Setup | page 190 |
| Print *filename* | page 190 |
| Print Entity Graph | page 278 |
| Recent Files | page 99 |
| Recent Files > Clear Menu | page 99 |
| Recent Projects | page 99 |
| Recent Projects > Clear Menu | page 99 |
| Exit | page 51 |

# Edit Menu

The **Edit** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Undo | page 169 |
| Redo | page 169 |
| Cut | page 168 |
| Copy | page 168 |
| Copy Image to Clipboard | page 276 |
| Paste | page 168 |
| Select All | page 168 |
| Comment Selection | page 179 |
| Uncomment Selection | page 179 |
| Change Case | page 179 |
| Toggle Overtype | page 180 |
| Bookmarks | page 181 |

# Search Menu

The **Search** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Find | page 163 |
| Find Previous | page 163 |
| Find & Replace | page 163 |
| Go to Line | page 167 |
| Go to Matching Brace | page 178 |
| Instant Search | page 148 |
| Find in Files | page 150 |
| Replace in Files | page 153 |
| Show Find in Files Results | page 152 |
| Find Entity | page 155 |

# View Menu

The **View** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Toolbars | page 162 |
| Browse Mode | page 168 |
| Zoom | page 166 |
| Fold All | page 178 |
| Soft Wrap | page 180 |
| Hide Inactive Lines | page 178 |
| Bookmarks | page 181 |
| Contextual Information | page 164 |
| Entity Filter | page 129 |
| Entity Locator | page 155 |
| Information Browser | page 131 |
| Favorites | page 143 |
| Analysis Log | page 121 |
| Project Browser | page 137 |
| Scope List | page 167 |
| Window Selector | page 160 |
| Previewer | page 177 |
| Dependency Browser | page 140 |

# Project Menu

The **Project** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Configure Project | page 39 |
| Rescan Project Directories | page 46 |
| Analyze Changed Files | page 121 |
| Analyze All Files | page 121 |
| Improve Project Accuracy > Undefined Macros | page 123 |
| Improve Project Accuracy > Missing Includes | page 125 |
| Improve Project Accuracy > More Information | page 122 |
| Project Overview Charts | page 246 |
| Architectures > New Architecture | page 202 |
| Architectures > Browse Architectures | page 193 |
| Architectures > Manage Architectures | page 201 |

# Reports Menu

The **Reports** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Configure Reports | page 208 |
| Generate Reports | page 210 |
| View Reports > HTML | page 211 |
| View Reports > Text | page 211 |
| Dependency > Architecture Dependencies | page 242 |
| Dependency > File Dependencies | page 242 |
| Dependency > Class Dependencies | page 242 |
| Project Interactive Reports | page 32 |

# Metrics Menu

The **Metrics** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Metrics Summary | page 232 |
| Browse Metrics | page 233 |
| Export Metrics | page 235 |
| Project Metric Charts > Code Volume | page 237 |
| Project Metric Charts > File Volume | page 237 |
| Project Metric Charts > Average Complexity | page 237 |
| Project Metric Charts > Sum Complexity | page 237 |
| Configure Metric Charts | page 237 |
| Metrics Treemap | page 240 |

# Graphs Menu

The **Graphs** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Dependency Graphs > By *architecture* | page 196 |
| Dependency Graphs > Load Saved Dependency Graph | page 197 |
| Project Graphs | page 32 |
| Graphs for *selected entity* | page 248 |

# CodeCheck Menu

The **CodeCheck** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Open CodeCheck | page 282 |
| Re-Run Previous Checks | page 292 |
| Analyze Changes and Re-Run Previous Checks | page 292 |
| Standards | page 292 |
| Saved Configurations | page 292 |
| Implement Your Corporate Standard | page 293 |

# Annotations Menu

The **Annotations** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Annotate | page 184 |
| Filter Annotations | page 188 |
| Manage Orphaned Annotations | page 189 |
| Search Annotations | page 187 |
| Annotation Options | page 59 |
| Refresh Annotations | page 59 |
| Display Inline | page 59 |
| Display Hover | page 59 |
| Display Indicator | page 59 |

# Tools Menu

The **Tools** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Run Command | page 317 |
| User Tools > *tool_name* | page 314 |
| User Tools > Configure | page 306 |
| Editor Macros > Record Macro | page 180 |
| Editor Macros > Replay Macro | page 180 |
| Editor Macros > Save Macro | page 180 |
| Editor Macros > Configure Macros | page 114 |
| Scheduler > Scheduled Activities | page 50 |
| Compare > Compare Files/Folders | page 296 |
| Compare > Compare Entities | page 299 |
| Compare > Compare Arbitrary Text | page 300 |
| Options | page 95 |

# Window Menu

The **Window** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Close *current_file* | page 170 |
| Close All Document Windows | page 160 |
| Release Window | page 160 |
| Split Vertically | page 160 |
| Split Horizontally | page 160 |
| Unsplit | page 160 |
| Tile | page 160 |
| Cascade | page 160 |
| Predefined Windows Layouts | page 160 |
| <*open source file list*> | page 160 |
| Windows | page 160 |

# Help Menu

The **Help** menu in *Understand* contains the following commands:

| Command | See |
|---|---|
| Help Content | page 16 |
| Key Bindings | page 104 |
| Example Projects | page 16 |
| PERL API Documentation | page 32 |
| Python API Documentation | page 32 |
| Frequently Asked Questions | page 16 |
| Getting Started | page 21 |
| Licensing | page 14 |
| View SciTools Blog | page 16 |
| Check for Updates | page 21 |
| About Understand | page 16 |

# Index

## Symbols



## 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