Red Hat Source−Navigator™ User's Guide

How to Contact Red Hat

Red Hat Corporate Headquarters

2600 Meridian Parkway
Durham, NC 27713 USA
Telephone (toll free): +1 888 REDHAT 1
Telephone (main line): +1 919 547 0012
FAX: +1 919 547 0024
Website: http://www.redhat.com

Part #: 300−400−1010020−05

Contents

Contents

Part I: Source−Navigator Tutorial

Part II: User's Guide

List of Figures and Tables

Introduction

Red Hat Source−Navigator ™ is a powerful code analysis and comprehention tool that provides a graphic framework for understanding and reengineering large or complex software projects. Source−Navigator's cross−platform nature also makes it an invaluable code porting tool.

Source−Navigator parsers scan through source code, extracting information from existing C, C++, Java, Tcl, **`[incr tcl]`**, FORTRAN, COBOL, and assembly programs and then use this information to build a *project database*. The database represents internal program structures, locations of function declarations, contents of class declarations, and relationships between program components. Source−Navigator graphical browsing tools use this database to query symbols (such as functions and global variables) and the relationships between them.

In addition to the languages supported in the standard distribution, you can use the Source−Navigator Software Development Kit (SDK) to add new parsers and extend Source−Navigator functionality to other languages. For more information, refer to Introduction in the ***Programmer's Reference Guide***.

For information on licensing and redistribution terms, see GNU General Public License.

Managing Projects

A Source−Navigator project is an entity containing references to source code files. A project describes where files are located and how to operate on them. Once a project is defined, developers can:

- identify, locate, access, modify, and analyze program components, including symbol definitions and usage of classes in object−oriented languages.
- prevent parallel modifications on the same code and manage different versions of sources by way of an interface to external version control systems such as ClearCase and CVS.
- create views of pertinent information while hiding information not relevant for the current view.

Navigational Tools

All Source−Navigator tools are organized around the project database that holds all project−specific symbols. The name and location of the project files, symbols extracted from the source code, and the relationships between symbols all reside in the project database. File structure may not be pertinent when visualizing code relationships, so Source−Navigator allows you to view and understand software structure regardless of which file contains what information.

The Symbol Browser, the Editor, and the Cross−Referencer are the basic tools for working with source code in a project.

- The Symbol Browser operates on the project database and helps you learn the existing program structure. The Symbol Browser list can be sorted and filtered in different ways, showing you how and where symbols are used. The Symbol Browser answers the question "What is in this project?"

  See Symbol Browser for more information.

- The Editor, the main Source−Navigator window, combines navigating with browsing. It allows you to navigate through actual program text and shows the location of that text in the source code. The Editor highlights the basic syntax of all supported programming languages and updates the project database

when files are modified and saved. This means that the project database is always up−to−date with your changes during the development cycle. The Editor answers the question "How is this project structured?"

See Editor for more information.

- The Cross−Referencer shows, for a given symbol, all the other symbols it refers to, and all the symbols that refer to it. These are known as *Refers−to* and *Referred−by* relationships, respectively. The Cross−Referencer answers the question "How do the parts of this project work together?"

  See Cross−Reference Browser for more information.

About this Guide

This document serves as a reference to Source−Navigator menus, tools, and functionality.

Document Conventions

This documentation uses the following general conventions:

Italic Font

Indicates a new term that will be defined in the text and items called out for special emphasis.

Bold Font

Represents menus, window names, and tool buttons.

Bold Italic Font

Denotes book titles, both hardcopy and electronic.

`Plain Typewriter Font`

Denotes code fragments, command lines, contents of files, and command names; also indicates directory, file, and project names where they appear in body text.

`Italic Typewriter Font`

Represents a variable for which an actual value should be substituted.

Menu names and their submenus are separated by an arrow (−>). For example, File −> Open means from the File menu, select Open from its submenu.

Paths are written in UNIX notation (forward slashes) throughout; **`.../bin`** means the directory Source−Navigator is installed into, subdirectory **`bin`**.

Mouse Conventions

The following are conventions for using the mouse with Source−Navigator:

Click	Place the cursor on a specified object and press the left mouse button. Double−click means to click the left mouse button twice in rapid succession without moving the mouse. The term "click" alone always means left−click.

Ctrl+click	Depress and hold the **Ctrl** key, simultaneously clicking with the left mouse button.
Shift+click	Depress and hold the **Shift** key, simultaneously clicking with the left mouse button.
Right+click	Place the cursor on a specific object and click the right mouse button.
Select Text	Click and drag cursor through text (or code) to be selected. Selected text is highlighted.
Select Entries	Clicking a line selects it. To indicate the end of the selection, hold down the **Shift** key and click the desired line.

Keyboard Conventions

You can use the keyboard to activate many of the functions displayed on the toolbar and in the menus.

- Holding down the Alt key while pressing S is represented as Alt+S.
- To open a command in the menu bar, press the Alt key plus the first letter of the menu. If a letter in the menu item is underlined, press the underlined letter to open the submenu.

Source−Navigator Tutorial

Source−Navigator scans your source code and loads the extracted information into a project database. This database stores all information about file names, symbol elements, and symbol relationships (functions and global variables are examples of symbols). Source−Navigator provides you with different browsers (graphical views) into the project database.

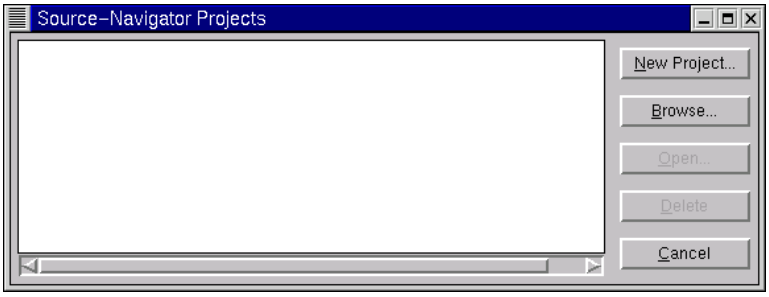
Creating a New Project

To create a new project, follow these steps:

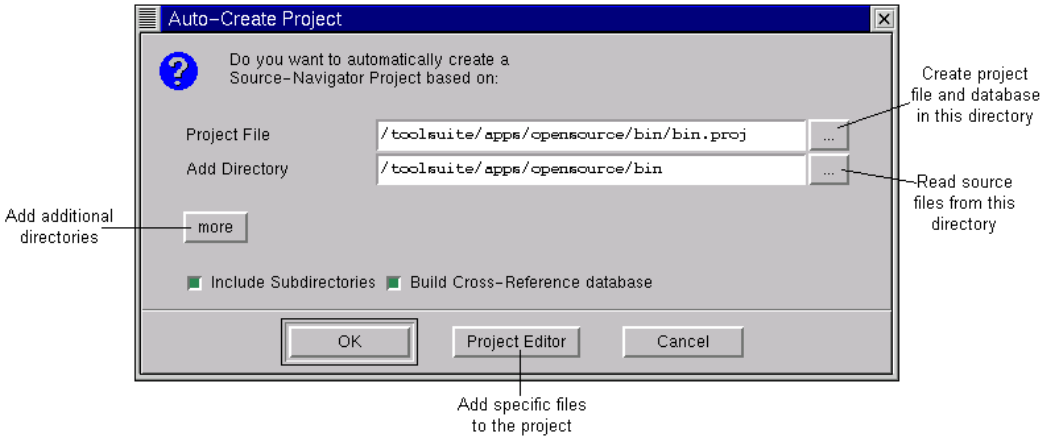
1. Create a directory in which to store your project files.

2. Launch Source−Navigator.
   In UNIX, at the command prompt, type **`snavigator`**
   In Windows, open the directory where Source−Navigator is located and double−click the snavigator.exe icon.

3. Source−Navigator launches and the Projects window appears.

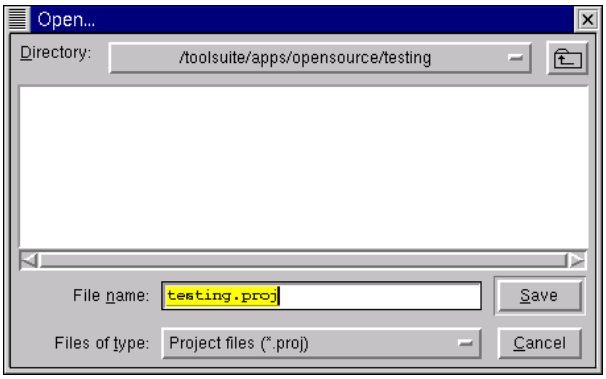


4. Click the New Project button.
   The Auto−Create Project dialog appears.



5.

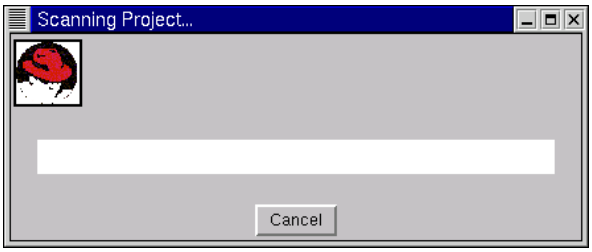
Click the "..." button next to the Project File text box. The Open dialog appears. Navigate to the project directory.
Enter the name of the project.



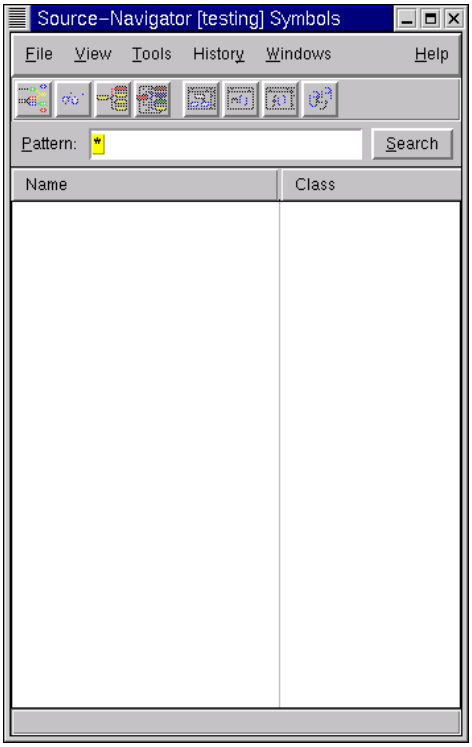
Click the Save button. The Open dialog closes.

6. If the Add Directory text box does not point to the project directory, use the "..." button to navigate to it. Click OK when done.

7. The Scanning Project progress bar appears:



8. Once your project is created, the Symbol Browser appears:

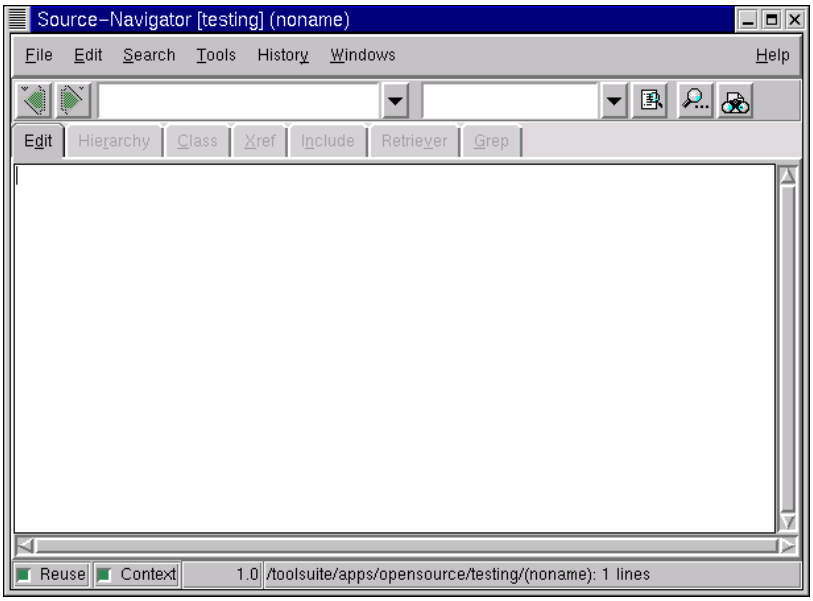


The Symbol Browser is empty because there are not any files in

the project.

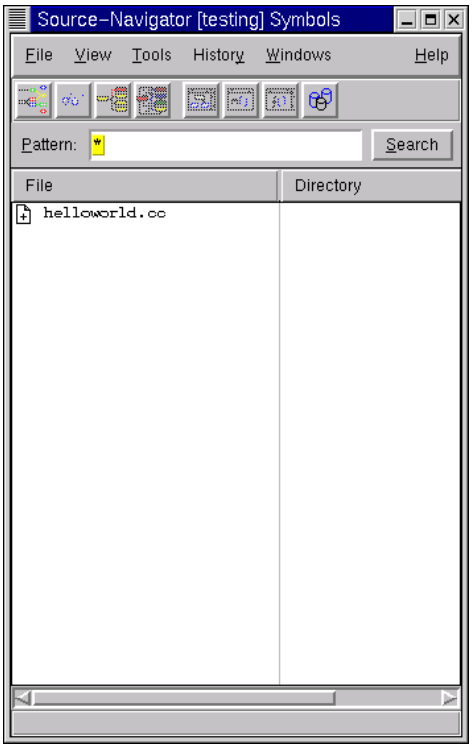
9. From the Windows menu, select Add View −> Editor to create a new file.

10. The Editor window appears.



11. Enter your code and save the file. Saving the file using either Save or Save as updates the Symbol Browser. Fast Save does not update the Symbol Browser window.

12. Click the Symbol Browser window to see the newly created symbol for the selected directory.

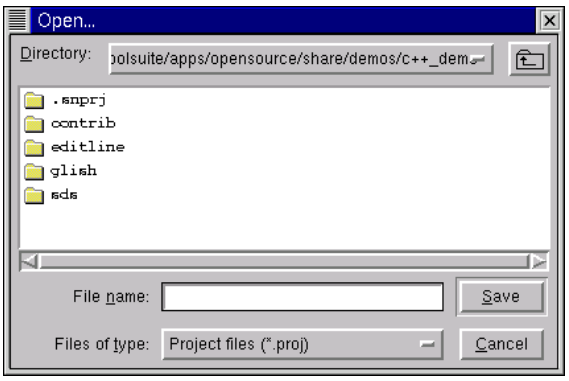


Opening a New Project

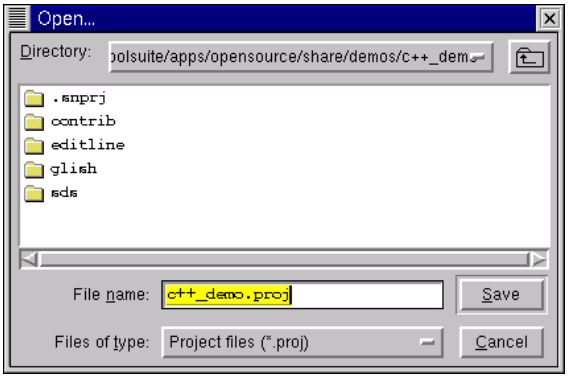
In this tutorial, existing source code is used to create the **c++_demo** project.

1. In the Symbol Browser, from the File menu, select New Project.
   The Auto−Create Project dialog appears.

2. Click the "..." button next to the Project File field. The Open dialog appears. Navigate to the **c++_demo** directory. The path is **.../share/demos**.



3. Enter **c++_demo.proj** to create the C++ demo project.

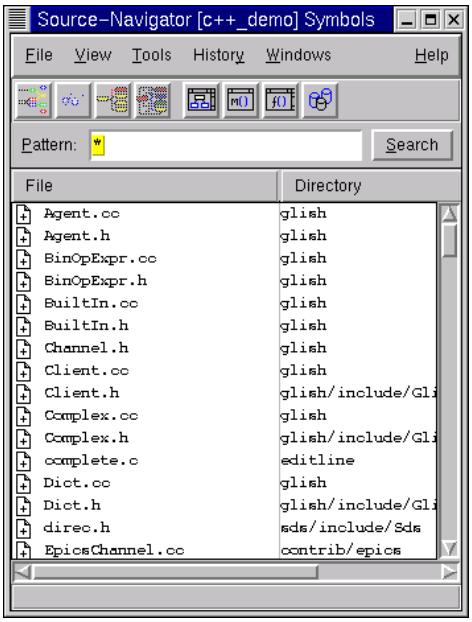


4. Click the Save button. The Open dialog closes.
   If the Add Directory text box does not point to the **c++_demo** directory, use the "..." button to navigate to it. The path is **.../share/demos**.

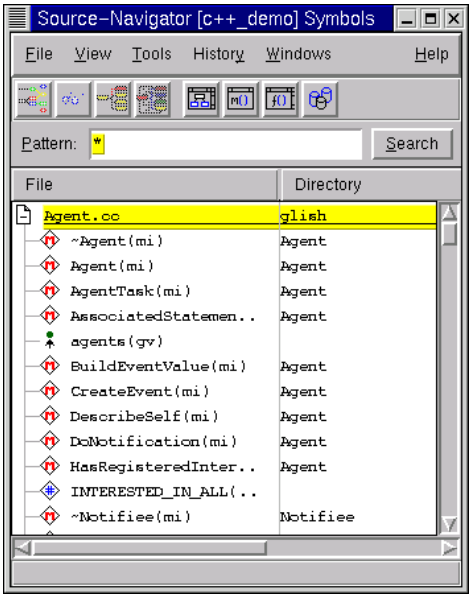
5. Click OK to generate the **c++_demo** project.

Using the Symbol Browser

After you create the demo project, the Symbol Browser window opens. The Symbol Browser provides a view of the symbols within the project.



1. Click on the icon to the left of the file or symbol name to expand the list view to a tree view of the file or symbol.

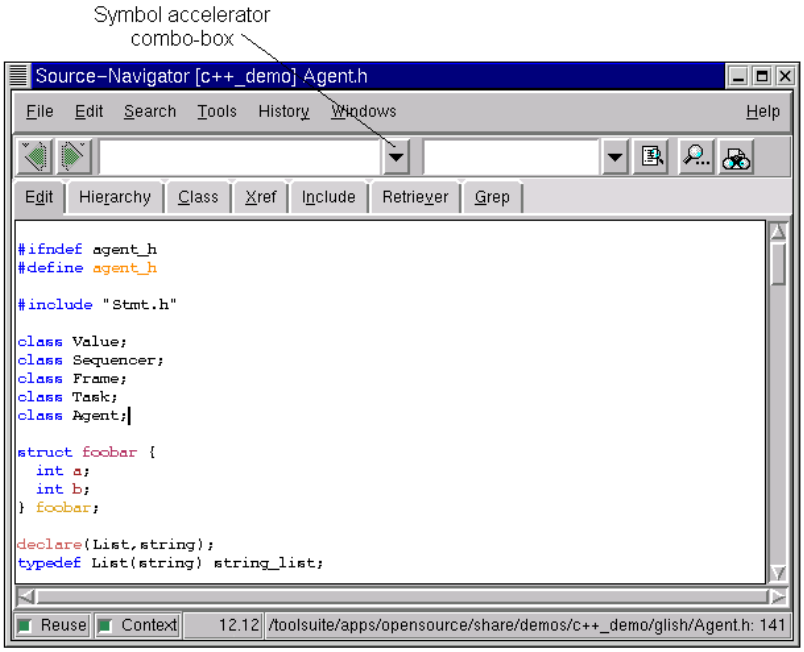


2. Double−click on the file or symbol name **`Agent.h`** to open the Editor to the location of the selected file or symbol in the source code.

For more information on the Symbol Browser, see Symbol Browser.

Using the Editor

Items in the project database are hyperlinked to the Editor. When you double−click on a symbol anywhere within Source−Navigator, the Editor opens to the location of that selected symbol in your source code.

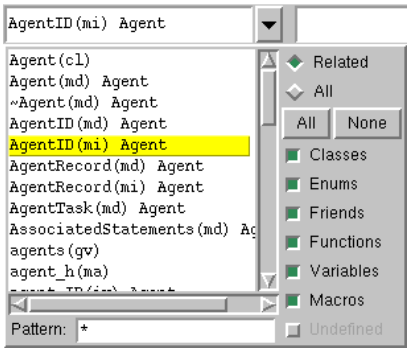


When you save modified source files, the project database is updated and changes are reflected in all of the Source−Navigator tools.

For more information on the Editor, see Editor.

Using the Symbol Accelerator

The Symbol Accelerator combo−box in the toolbar allows you to quickly navigate through the code. When the Editor is open, the Symbol Accelerator lists all of the symbols either within the open file or within the entire project.



When other browsers are open, the Symbol Accelerator lists the components relative to that browser. For instance, in the Class Browser, it lists only classes in the file or in the project.

Notice that the Symbol Accelerator text field in the toolbar displays the symbol that is referenced as the cursor moves through the file.

Using the Cross−Reference Browser

The Cross−Reference Browser (also called Xref) helps you to understand complex source code by showing the *Refers−to* and *Referred−by* relationships between symbols in the project. You can traverse the tree, and expand or collapse the elements within the tree.

Cross−Reference Browser relationships

Refers−to
shown by blue arrows

Referred−by
shown by red arrows

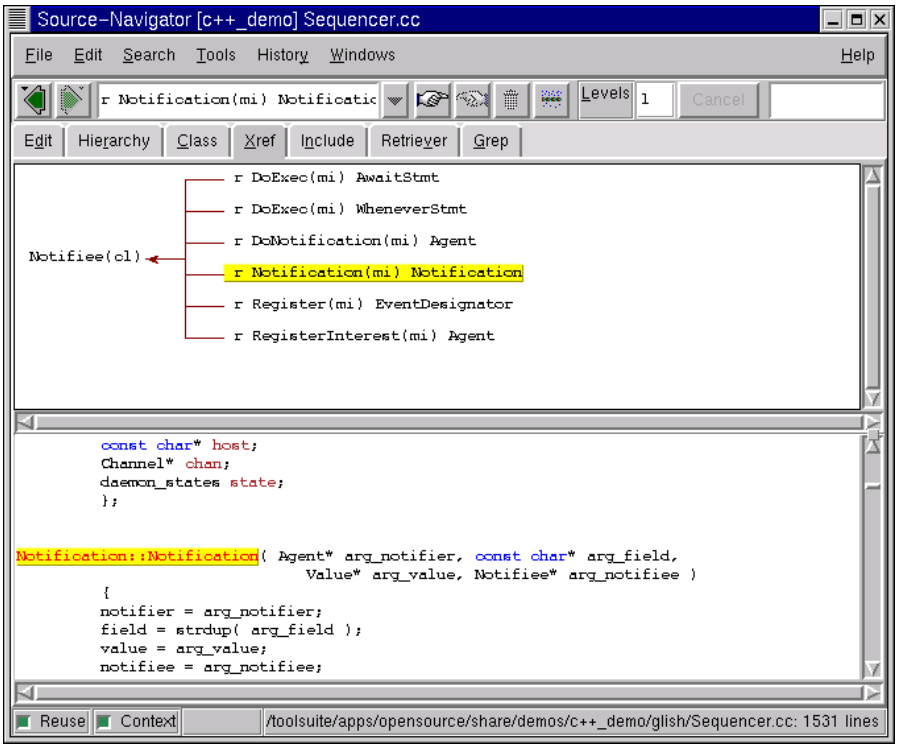
1. From the Symbol Browser, open the **`Agent.h`** file.

2. Highlight the class **`Notifiee`**.

3. To activate the Cross−Reference Browser, select the Xref tab in the Editor window.

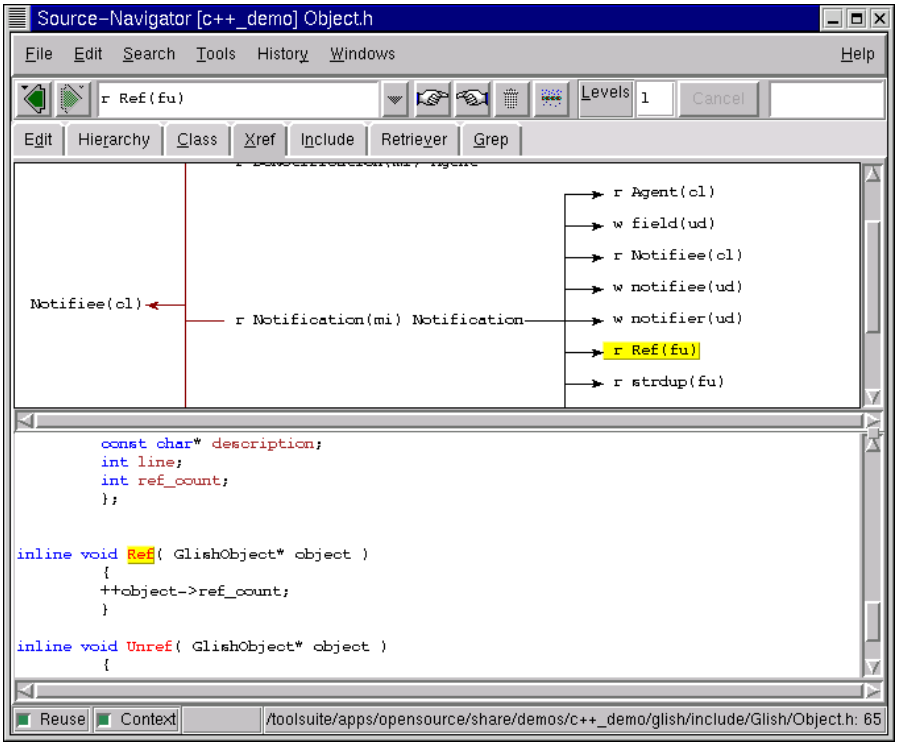
4. Add an Editor view to the Cross−Reference Browser window. From the Windows menu, select Add View −> Editor.

5. Click on the function **`Notification(mi) Notification`**.



6. Click the *Refers−to* button. The Cross−Reference Browser view lists the functions that reference **`Notification(mi) Notification`**.

7. Click on **`Ref(fu)`** and notice that the *Refers−to* and *Referred−by* buttons in the toolbar become active. Click the *Referred−by* button.



Source−Navigator shows the source syntax of **`Ref(fu)`** in the Editor pane, and shows the functions that reference **`Ref(fu)`**.

For more information on the Cross−Reference Browser, see Cross−Reference Browser.

Using the Class Browser

The Source−Navigator Class Browser provides a view of classes and their attributes. Using the Class Browser, you can analyze the members of a selected class based on inheritance tree, scope, member attribute, member type, and more.

Imagine a situation where you are investigating the class structures and attributes of classes within your project, but want to simultaneously view the declaration or implementation of the classes in the source code.

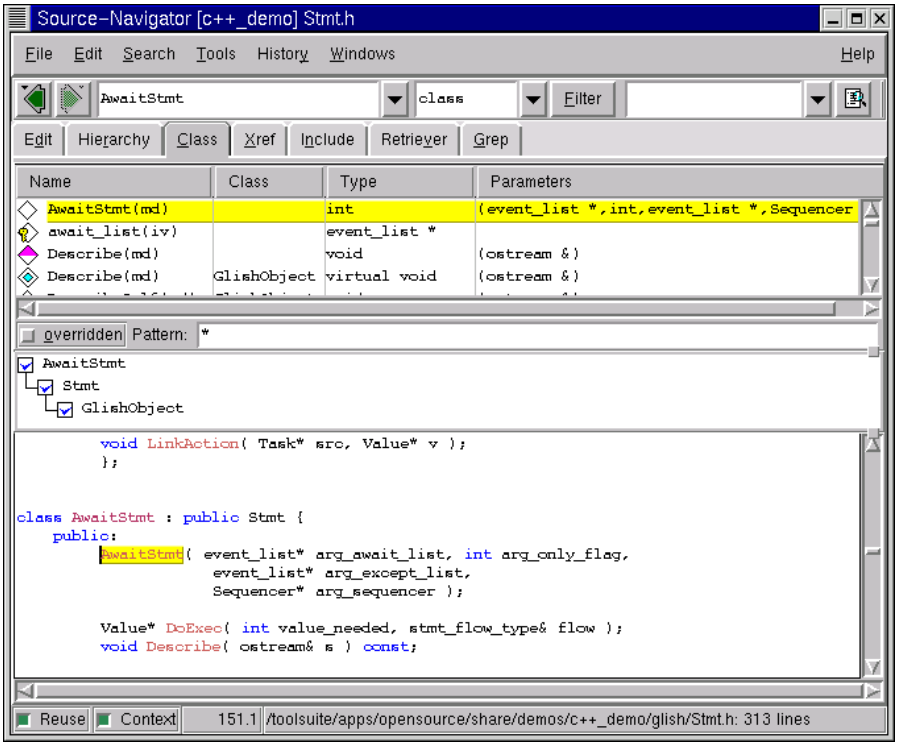
1. Ensure that **`Agent.h`** is open. If it is not, select it from the Symbol Browser.

2. To activate the Class Browser, select the Class tab in the Editor window.

3. Add an Editor to the Class Browser. From the Windows menu, select Add View −> Editor.

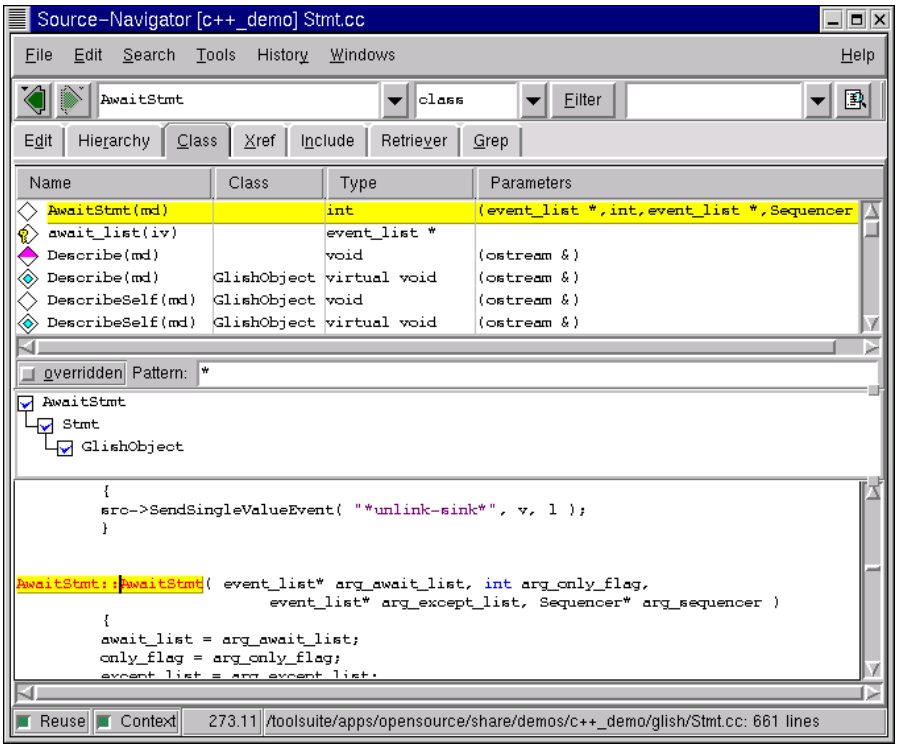
4. In the Symbol Accelerator, all class components for the entire project are listed. From this list of components, select **`AwaitStmt`**.

5. In the Class Browser, click **`AwaitStmt(md)`** in the list view.



Notice that the Editor view on the bottom takes you to the Declaration of the **`AwaitStmt`**.

6. From the Search menu, select Find Implementation.



Notice that the Editor view updates to show the Implementation.

Note

Declaration is set by default. You can set Source−Navigator to open to the Implementation instead. For more information, see Class/Hierarchy Preferences.

For more information on the Class Browser, see Class Browser.

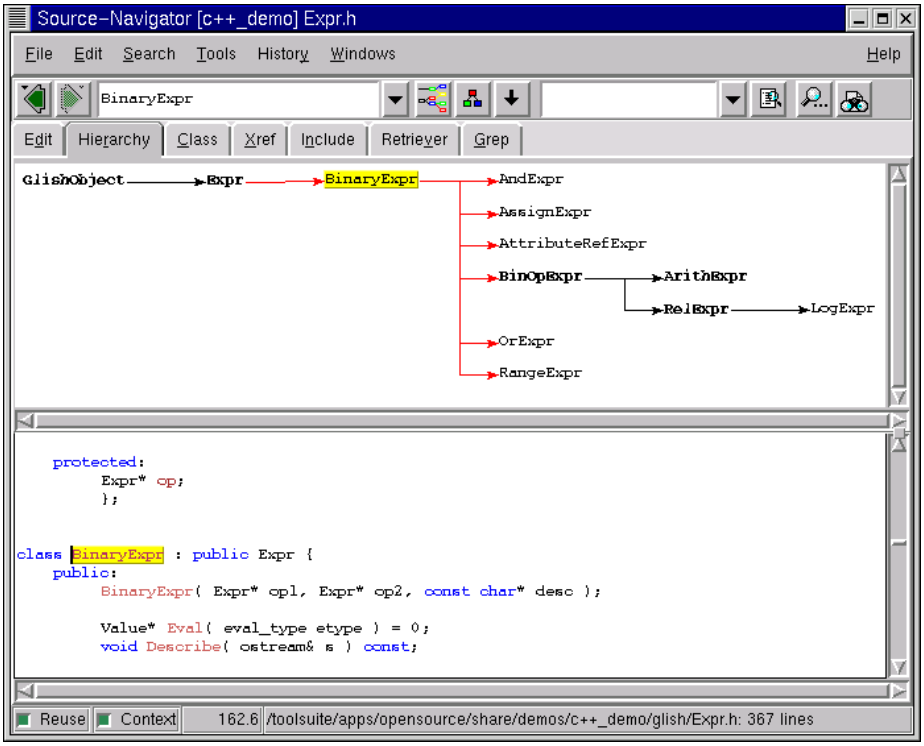
Using the Hierarchy Browser

The Hierarchy Browser (also known as the Class Hierarchy Browser) helps you understand the inheritance relationship for a particular class.

1. To activate the Hierarchy Browser, select the Hierarchy tab in the Editor window.

2. Add an Editor to the Hierarchy Browser. From the Windows menu, select Add View −> Editor.

3. From the Symbol Accelerator, select the class **`BinaryExpr`**.



Source−Navigator draws an inheritance tree from **`GlishObject`**, the base class, to **`LogExpr`**, a derived class.

By clicking on any of the classes in the tree, Source−Navigator shows the class definition in the source code in the Editor.

For more information on the Hierarchy Browser, see Hierarchy Browser.

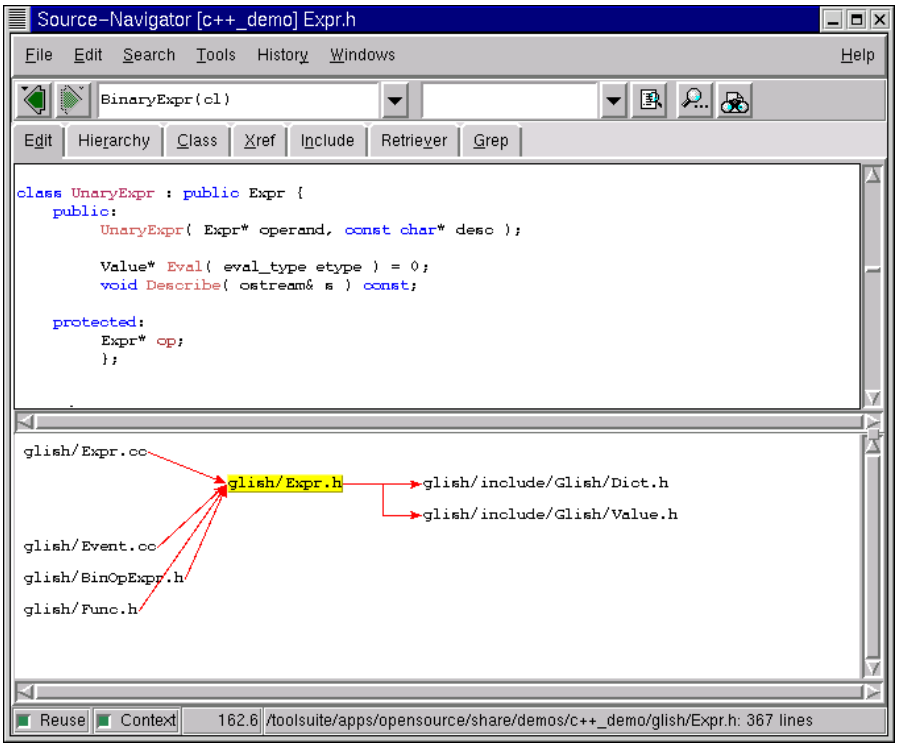
Using the Include Browser

Some programming languages allow files to include source code from other files. The Include Browser allows you to see these *Includes* and *Included by* relationships.

1. Click the Edit tab.

2. Add an Include Browser to the Editor. From the Windows menu, select Add View −> Include.

The Include Browser allows you to maintain awareness of where you are in the project based on pathnames and included files.



This complex view will maintain your visibility while you move between files.

3. Go to the History −> Edit menu.

4. Select another view from the Editor, such as the previous **`Notifiee`** view.

5. Notice that the Editor opens a view to that source file and the integrated Include Browser shows information relative to the "including" or "included" files.

For more information on the Include Browser, see Include Browser.

Closing the Demo Project

To close a project, from the File menu, choose Close Project.

This tutorial introduced you to some of Source−Navigator's browsers for more information on these and other useful tools, see any chapters in Part II of this User's Guide.

With the user interface concepts conveyed in this tutorial, you are now ready to begin exploring your own software projects.

Contents Previous Next
Contents Previous Next

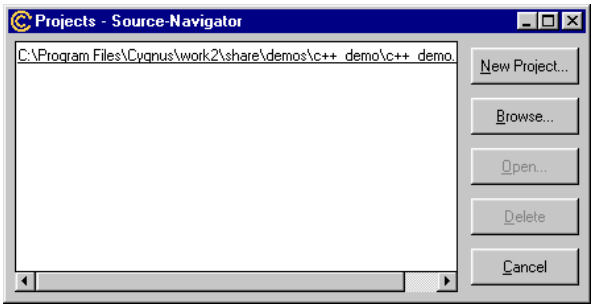
Using the Project Editor

You learned how to create basic Source−Navigator projects in the Source−Navigator Tutorial section. Now, you will learn to use the Project Editor to fine−tune your projects.

Project Editor Details

The Project Editor enables you to add, hide, and unload files from your project. You can also use it to create and change views. To open the Project Editor, from the File menu, select Project Editor.

Project Editor Window



Selecting an option listed next to Display project files as: changes the layout of the files and directories in the Project Editor window; Tree is the default layout.

The Project File text box contains a file path; the directory that your project file is located in is your *project directory*. From now on, every non−absolute path will be relative to this directory. If you would like to choose a different path, click the "..." button. You can change the path only when creating a project.

Note

If the source code included in the project is in a read−only directory, you will not be able to create the project and corresponding database files in the same directory as your source files. You must choose a directory where you have write permission for the project file and the database files.

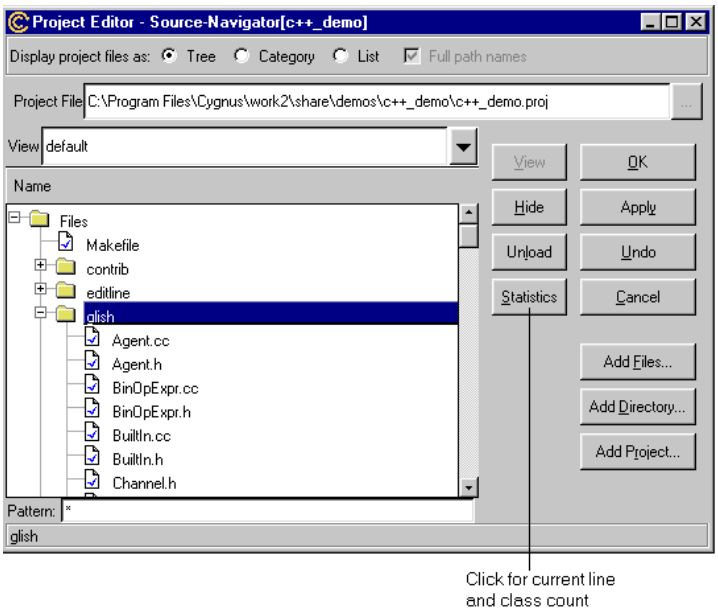
Adding Files to a Project

To add source files to a project, click the Add Files button. The Open dialog appears:

Click on Files of type: at the bottom of the dialog to bring up a menu from which you can filter files based on the file extensions shown in the menu; All files (*) is chosen by default. Select the source files to be added to the project and click the Open button.

Adding Directories to a Project

The Add Directory button allows you to add entire directories and their contents to your project.



Select the directory you want to add and click OK. You can add as many directories as you wish, but you must add them one at a time.

Adding Another Project to a Project

The Add Project button allows you to import another Source−Navigator project and all of the files it includes into your current project.

This is the same Open dialog as when you clicked on the Add Files button, except that Project files (*.proj) is chosen by default from the Files of type menu. To see all files in the directory, choose All files (*). Click on the project from which you want to import files.

Using Views

Views show a partial set of project files. For example, one view might be set to show only database−specific files from a project, while another one shows the GUI components in the project. You may hide specific files in one view and show these hidden files in other views.

Creating views

Enter the name of the new view in the View text box and click OK. This creates the new view and closes the Project Editor. The next time you open the Project Editor, this view will be the default view in the View selection box.

Selecting another view

To select an existing view, click the down arrow to the right of the View text box. A pulldown list of views appears. Select the view you wish to see.

Hiding Files from a View

The Hide button allows you to hide files from the current view, but does not delete them from the project or the file system. Hide files or directories by clicking on a file or directory in the file system tree view of the Project Editor, and click the Hide button.

A new folder named Hidden appears and the hidden files and directories are placed into this folder. To make the hidden items visible again, select the hidden file or directory and click the View button.

Unloading Files from a Project

The Unload button allows you to remove files from a project, but not from your hard drive. To remove files from a project, select the files from the Project Editor window and click the Unload button.

Statistics for a Project

The Statistics button allows you to view the number of symbols (such as classes, methods, and functions) in a file. In the Name text box, select a file and click the Statistics button. The statistics for the file are displayed. Click Close to close the Statistics window.

Closing the Project Editor

When you're finished working with the Project Editor, click OK.

Closing Projects

To close a project, from the File menu, select Close Project. Closing a project:

- closes all tools used while working on the project or a project view.
- closes all database files for the project.
- brings up the Projects window, if no other projects are open.

Deleting Projects

To delete a project while using the Symbol Browser, from the File menu, select Delete Current Project. To delete a project from another browser, from the File menu, select Project −> Delete Current Project. The project file and other project database files are deleted. All source files and directories remain unchanged.

Importing Directories into a Project

As an alternative to using the Project Editor to select source code directories for the new project, you can collect directory names into a separate file and import this file into Source−Navigator. You can also use the **`find`** command to generate this list.

To import directories, start Source−Navigator with the **`import`** option. At the command line, type:

```
snavigator −import filename
```

The project directory is the current working directory, and the file **`filename`** must contain directory names or filenames, one per line (the Include Browser uses the directory names). A sample file for import follows:

```
chk.h
chk.c
hello.h
/usr/tuxedo/src/main.c
/usr/tuxedo/src/read.c
/home/scribbles/tcp/call.c
```

```
/usr/local/include
/tmp
```

For more information on using Source−Navigator from the command line, see Command Line Options.

General Source−Navigator Features

This chapter describes the most commonly used features of Source−Navigator.

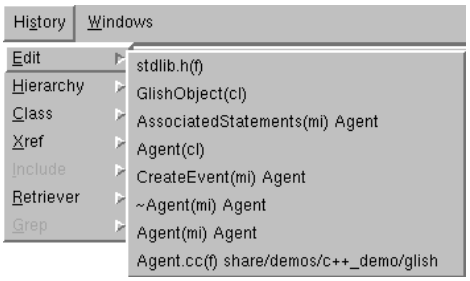
Menus

The File, Edit, Search, and Tools menus appear in all windows, however they are context−sensitive: the options available from the menus change depending upon which tool you're using. The History and Windows menus, however, are more general and offer the same options within each tool.

History Menu

The History menu enables you to repeat queries and restore previous states of each browser.

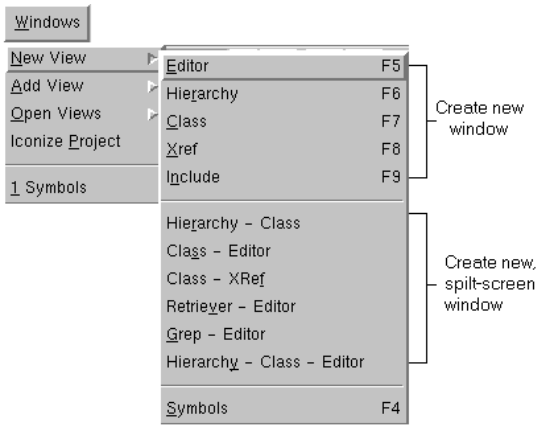
History Menu



Windows Menu

The Windows menu contains the tools available in Source−Navigator.

Windows Menu



To add a new tool in a view, from the Windows menu, select New View. To add a tool to your current window, from the Window menu, select Add View. To change to a currently running instance of a tool, from the Windows menu, select Open Views.

General Window Features

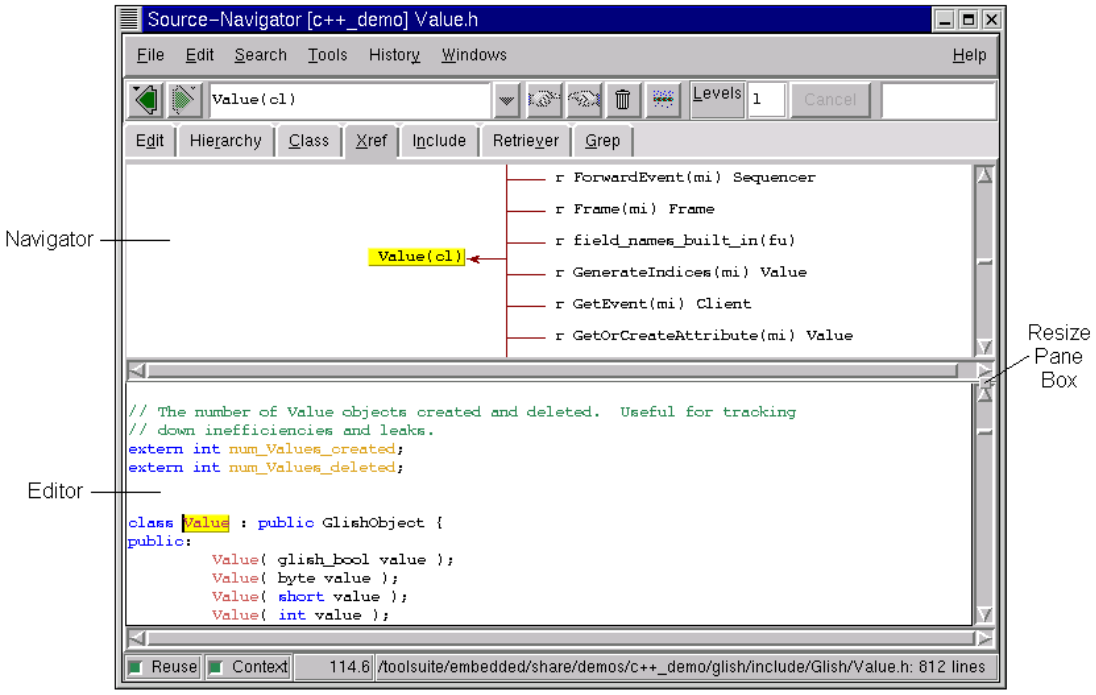
You may reuse Source−Navigator windows and adjust the window column size. The status line shows information such as the number of lines of source code in the Editor window and the current directory.

Adding a Browser to an Existing Window

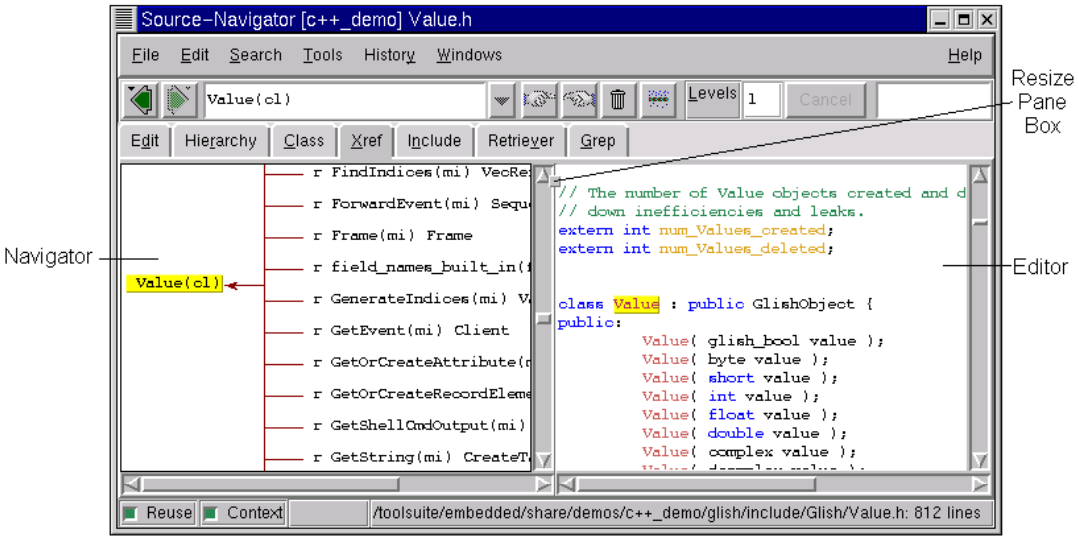
If you find yourself often using two Source−Navigator tools simultaneously, you can combine them into one window. This keeps you from constantly switching between windows, and allows you to see both tools at the same time.

For instance, if you have a class open in the Cross−Reference Browser and you want to bring up an Editor to see the same class, but you want to continue to see the cross−references, from the Windows menu select Add View −> Editor, and an Editor will appear in the same window. Depending upon your settings in the Preferences dialog (see Window), the new Editor appears either at the bottom of the window (Vertical) or on the right side of the window (Horizontal).

Vertical Windows



Horizontal Windows



As you click on symbols in the Cross−Reference Browser, those symbols appear in the Editor.

Reusing Windows

Selecting the Reuse button from a window's status line, overwrites the window's contents. With Reuse deselected, a new window is created to display the new view.

For example, with Reuse deselected, double−clicking a symbol to display its symbol definition opens a new Editor window. Your previous Editor window remains unchanged. Reuse is selected by default.

Preserving Context Between Windows

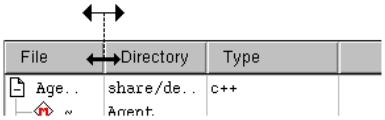
Selecting the Context button from a window's status line preserves the context of a selected symbol when you switch between tools.

If Context is deselected, the new tool opens to the default or empty condition.

For example, if a particular class is selected in the Editor, the same class is displayed when you change to the Class Browser. Context is selected by default. When Context is not selected, the Class Browser opens with whatever was previously displayed.

Adjusting Window Column Size

The column dividers allow you to adjust the width of columns on the screen. You can adjust the size of a column by moving your cursor over the divider until the left−right arrows appear, then clicking and dragging the column divider to the desired position.



Using Filters

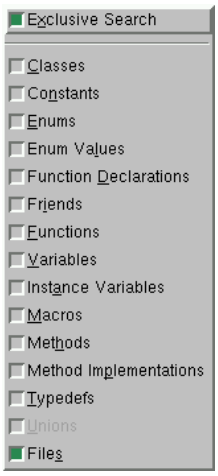
Source−Navigator provides several ways to filter the symbols displayed in the Symbol Browser. You can:

- use the List Filter buttons (see List Filter buttons).
- use the Symbol Selectors from the View menu.
- use Column Filters (see Column Filters).
- use the Symbol Accelerator combo−box (see Symbol Accelerator Combo−box).

Symbol Selectors

Although the List Filter buttons allow you to search by classes, methods, functions, and project files, the Symbol Selector provides a more complete list of search choices. In the Symbol Browser window, from the View menu, select Symbol Selectors.

Symbol Selectors Menu



If the Exclusive Search box is selected, you may choose only one symbol type to search for; if it's deselected, you may combine symbol types for more complex searches.

Pattern Box

In large projects, thousands of symbols may be listed in any list view. To list a subset, or to restrict a symbol list, use the Pattern search filed. When text is typed in this field, the list updates to show only the symbols or components matching that pattern. For instance, when Variables is selected in the Symbol Browser or Symbol Accelerator, typing ***gv*** in the Pattern: field shows all global variables in the file or project. See Symbol and Type Abbreviations for a list of pattern abbreviations.

The Pattern text box allows you to search in a tool for a particular pattern. Patterns are not case sensitive.

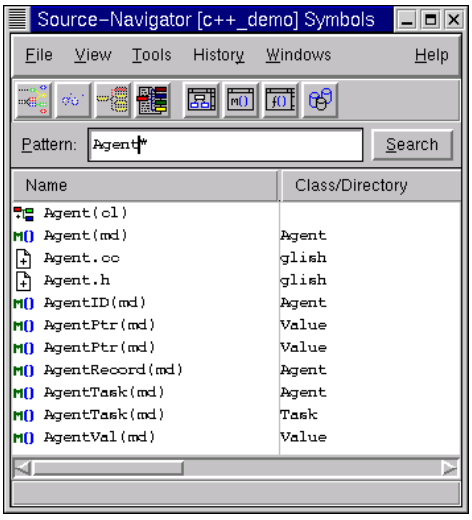
Pattern Interpretation of Special Characters

Special Character	Interpretation in Filter or Pattern Boxes
*****	Matches any sequence of zero or more characters.
?	Matches a single character.
[chars]	Matches any single character in **chars**. If **chars** contains a sequence of the form **c-x**, any character between **c** and **x** inclusive will match.
\?	Matches the **?** character exactly, avoiding special interpretation of the character. Also applies to the following characters: **[**, **]**, *****, **?**, and ****.

For example, ***.[hc]** matches all strings with ***.C**, ***.H**, ***.c**, and ***.h** extensions; **[0-9]*** matches all symbols beginning with a number.

With **Agent*** entered into the Pattern text box, and Classes, Methods, and Files chosen from the View menu, you would see the results shown in Symbol Browser Showing Filter Results.

Symbol Browser Showing Filter Results

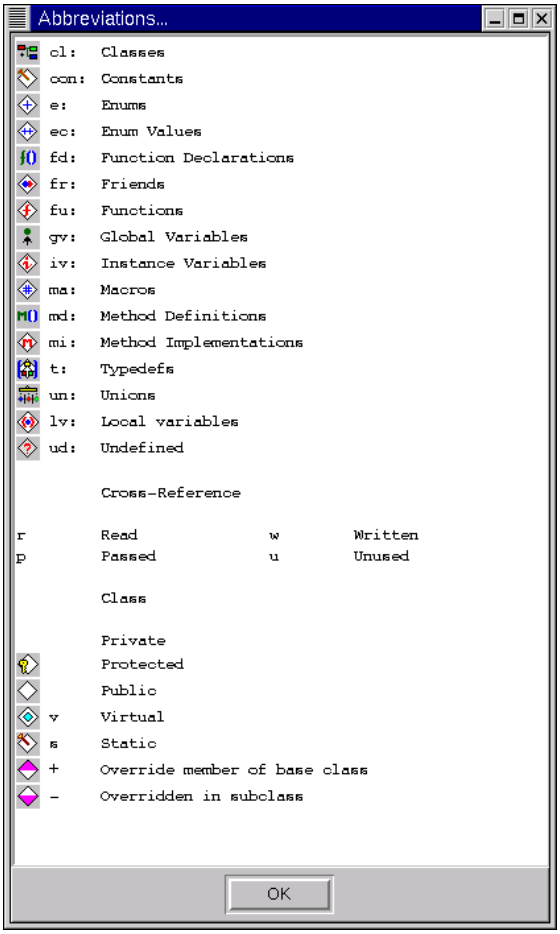


To clear the text box for another search, type Ctrl+U.

Symbol and Type Abbreviations

Source−Navigator uses the following abbreviations, which are accessible from the Help menu by selecting Abbreviations.

Abbreviations Panel

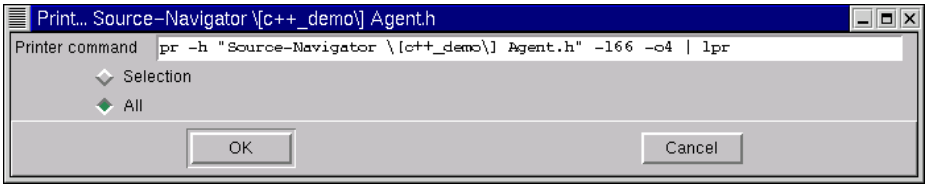


Printing from Source−Navigator

To print the contents of Source−Navigator browsers, such as the Editor or Cross−Reference Browser, from the File menu, select Print. The print dialog box varies with different printers, printer drivers, and platforms.

Print Dialog (UNIX)

UNIX Print Dialog



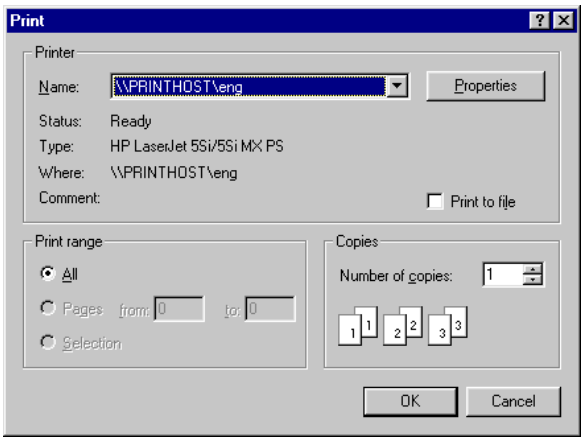
The default command line shown in the Printer command text box is set from the Others tab of the Project Preferences dialog (see Others tab).

Choose Selection to print only the highlighted portion of the file (if you do not highlight any part of the file, the entire file is printed).

Choose All to print the entire file.

Print Dialog (Windows)

Windows Print Dialog



Printer

Name

Choose the name of the desired printer.

Print to file

Check this box to print to a file.

Note

If you choose Print to file, a dialog box appears after clicking OK. Type the desired filename (alone or with a path to a specific directory) in the Output File Name text box and click OK.

If you enter the filename without a path, your file is saved in the same directory as the project file.

Print range

All

Prints the contents of the entire file.

Selection

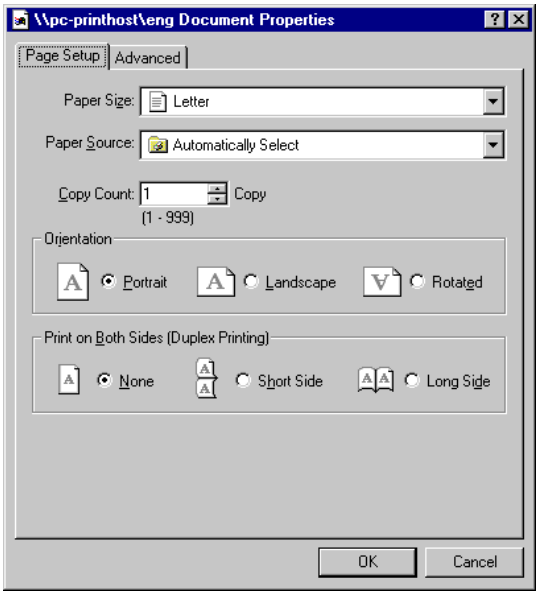
Prints only the highlighted portion of the file. If you do not highlight any part of the file, the entire file is printed.

Copies

Number of copies

Choose the number of copies to print.

Windows Document Properties Dialog



The options in this dialog box depend on which printer you have selected, but usually include paper size, orientation, and duplex printing. For more information on these options, see your printer documentation.

Contents Previous Next
Contents Previous Next

Customizing Source−Navigator

This chapter describes how to customize Source−Navigator to reflect your preferences. For additional information on changing the start−up and runtime behaviors, see the Customization chapter in the ***Programmer's Reference Guide***.

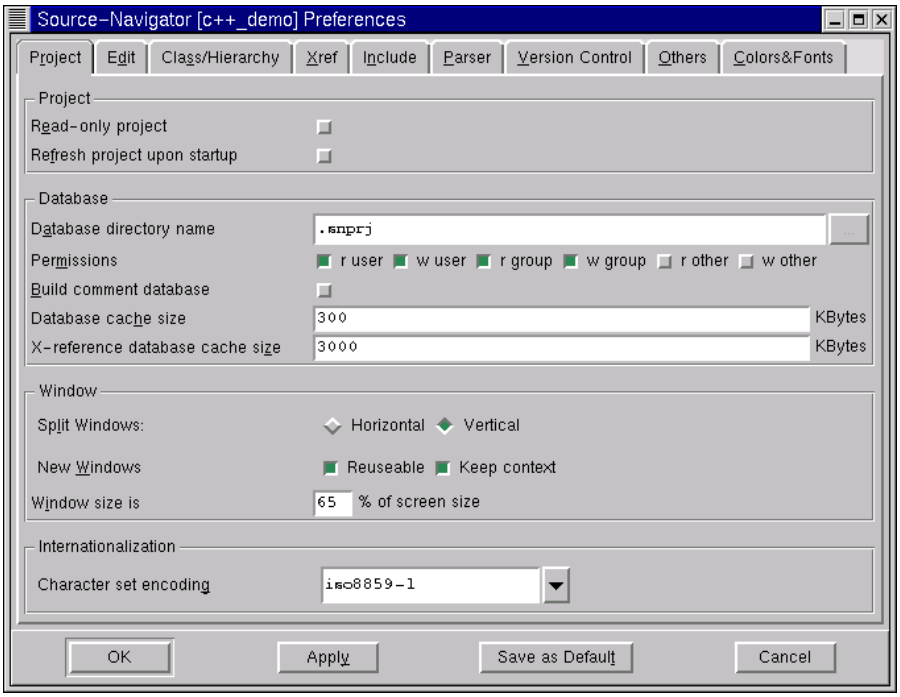
Preferences Dialog

Use the Preferences dialog to specify project−specific parameters. In the Symbol Browser, from the File menu, select Project −> Project Preferences and enter your changes in the dialog. Click OK to update the project with the current settings.

Note

Default values may be transferred from the previously created project, some options take effect only in new windows, and some options only take effect the next time the project is opened.

Project Tab of the Preferences Dialog



General Project Preferences

Descriptions of the general Preferences tabs (Project, Parser, Others, and Colors & Fonts) are included in this section. The following tabs are discussed in the relevant chapters:

Edit see Editor Preferences.
Class/Hierarchy see Class/Hierarchy Preferences.
Xref see Cross−Reference Preferences.
Include see Include Preferences. Version Control see Version Control Preferences.

Project tab

Project

Read−only project

Select this if the project should be read−only. Default is off.

Refresh project upon startup

Select this when files are likely to be changed by other developers, or when you want to be sure that your database is in sync with your sources when you start.

Default is off. Changing this to on may cause delays when opening your project. For large source bases that are relatively stable, or where network latency is a problem, set this to off and periodically, from the Tools menu, select Refresh Project to resync the database with the source.

Database

Database directory name

Source−Navigator creates all database files under this directory. If the directory already exists, you will need read and write permissions for it. If you're creating a new directory, you will need permission to create it.

The filesystem for this directory must contain free disk space equal to the size of the source base without cross−references, and up to about ten times the size of the source code if you choose to generate cross−references. If you don't have the necessary permissions (for example, if it is a read−only file system), or if there is insufficient free disk space in your first choice of location, you may create your project directory in another location on your network by entering a directory name with its absolute path.

This option can be changed only when creating a project.

Permissions

These buttons control the read−write permissions for your project: the first set controls access for the creator of the file, the second set for the group, and the third for "others" (everyone else on the network).

Build comment database

Select this to store comment strings in the database. Default is off.

Database cache size

Caches improve performance by using memory (fast) instead of disk (slow) where possible. Larger cache sizes increase the likelihood that data will be found in memory rather than on disk, though overallocating caches has the opposite effect. The operating system will swap the cache to disk and the system will get dramatically slower. The cache defaults are generous for most projects; don't increase them without a reason.

This option can be changed only when creating a project.

Enter the database cache size (in kilobytes) or accept the default value. Source−Navigator creates the project database (in the background) with the specified cache size. Increasing this amount speeds up project creation and data access, but requires that more memory be allocated to Source−Navigator.

The recommended maximum is the amount of free RAM divided by 16, up to a maximum of 4 MB. The total of this amount plus the amount allocated to the cross−reference database cache should not exceed one quarter of the total memory.

For more details about this parameter, see dbopen in the Database API chapter of the ***Programmer's Reference Guide***.

X−reference (Cross−Reference) database cache size

Enter the Cross−Reference database cache size in kilobytes or accept the default value. Source−Navigator creates the project cross−reference database with this cache size. Increasing this amount speeds up cross−reference creation and data access but requires that more memory be allocated to Source−Navigator.

This option can be changed only when creating a project.

The recommended maximum size is the amount of system memory divided by 32, up to 8 MB. The total of this amount plus the amount allocated to the database cache should not exceed one quarter of the total memory.

For more details about this parameter, see dbopen in the Database API chapter of the ***Programmer's Reference Guide***.

Window

Split Windows

These buttons control where new views appear when you add a view to an existing window (from the Windows menu, select Add View).

Select Horizontal to have new views appear to the right of the current pane; select Vertical to have new views appear below the current pane.

New Windows

Selecting Reusable causes new information to appear in the current window; deselecting it causes a new window to appear when you choose a new symbol, tool, or view.

Selecting Keep Context causes new tool windows to be opened to the same symbol context as the current window; deselecting it causes a new tool to be empty when opened.

Window size is

This setting controls the size of newly−created Source−Navigator windows. You may also resize the windows after they are created.

Internationalization

Character set encoding

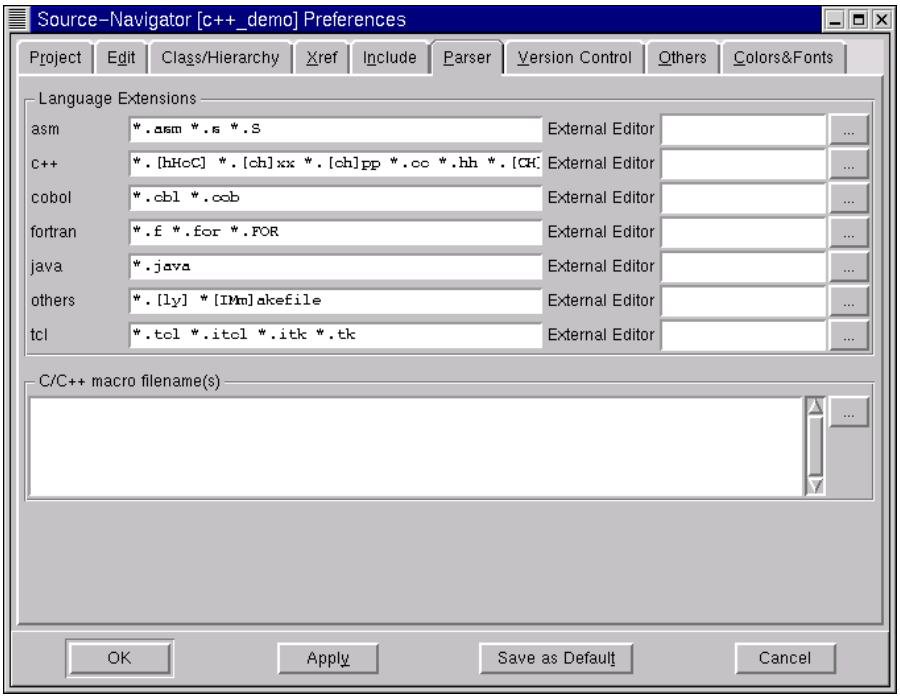
This combo−box allows you to choose the character set encoding for your project to match the character set of your source files.

The default character set for English, German, and most other European languages is ISO8859−1. For Japanese, the default character set is Shift−JIS.

Parser tab

Source−Navigator uses plug−in parsers to parse multiple programming languages. Choosing the appropriate parser is based on file extensions.

Parser Tab of the Preferences Dialog



Source−Navigator is pre−configured for the most commonly used file types; these can be changed and new parsers can be added. For more information on adding parsers, see the Adding Parsers chapter of the ***Programmer's Reference Guide***.

File Types and Associated Filename Extensions

Parse source files as	File Extensions
PowerPC 601 assembly	**`*.asm *.s *.S`**
C/C++	**`*.[hHcC] *.[ch]xx *.[ch]pp *.cc *.hh *.[CH]XX *.[CH]PP *.CC *.HH`**
Cobol	**`*.cbl *.cob`**
FORTRAN	**`*.f *.for *.FOR`**
Java	**`*.java`**
others	**`*.[ly] *[IMm]akefile`**

Tcl **[incr Tcl]**

***.tcl *.itcl *.itk *.tk**

After each of the languages is an External Editor text box; you may type in the executable (shell) command for an external editor, or you may click the "..." button to browse. If you leave the text box blank, Source−Navigator uses its built−in editor.

Macro processing

Macros make the task of source code analysis more complicated, and there is no single right way to handle them. By default, Source−Navigator treats them as opaque symbols and, aside from recording where they are defined, it ignores them completely. This behavior not only makes parsing substantially faster than true compilation, but it also preserves the layer of source code abstraction that is presented in the Editor. This behavior is particularly useful when maintaining software that must run on multiple platforms, and you would like to see all of the impacts that a change might have, regardless of a macro's platform definition.

For some projects and/or tasks, this layer of abstraction is a barrier to code comprehension. For these cases, you can direct Source−Navigator to define and expand macros in one of four ways:

- **define**
  define is used to insert a symbol into Source−Navigator's preprocessor namespace. If you **define FOO** (or **#define FOO**, the leading # is optional), then conditionals that test **#ifdef/#ifndef** are scanned according to the **#ifdef/#ifndef** test. **FOO** will be replaced with the empty string.

  define can also be used to give a symbol a value (such as **define FOO BAR**). This will not only inject the symbol into the namespace, but will cause Source−Navigator to scan **BAR** whenever it sees the **FOO** macro. This is particularly useful for **DEFUN** or **PROTO** macros that are used to bridge between K&R and ANSI C but which otherwise are not interesting at the source code comprehension level. It is also useful when macros test numerical results, such as **#if (X >= Y) or #if X**.

  define macros can take arguments, just as in C and C++, and they can expand recursively. For example, **FOO** might expand to **BAR (5)** which might then expand to **mumble (5, 5, 0)**; in this case Source−Navigator would only see **mumble (5, 5, 0)**.

- **replace**
  replace is just like **define**, except that the symbol is not injected into the namespace. Thus, if you want to expand macros, but not have conditional code compiled away, use **replace**.

  The macro processor does not support **include**, token concatenation, ANSI stringification, or other pre−processor directives. These may be implemented in a future release.

- **delete**
  Source−Navigator lets you use more than one macro file in a project. You can specify a macro in one file and then delete it from the preprocessor namespace using **delete** in a second macro file.

- **undef**
  undef doesn't do any substitution, it just affects the evaluation of **#if**, **#ifdef**, and **#ifndef** statements.

Defining and using macro files

As explained above, Source−Navigator parses, but does not interpret, macro definitions in your project files. It only interprets macro definitions from files you specify explicitly in the parser preferences of your project. Because

multiple files may be specified, you may want to organize your macro files according to global, per−user, and per−project divisions. This order is important, because Source−Navigator uses the last encountered definition for the macro.

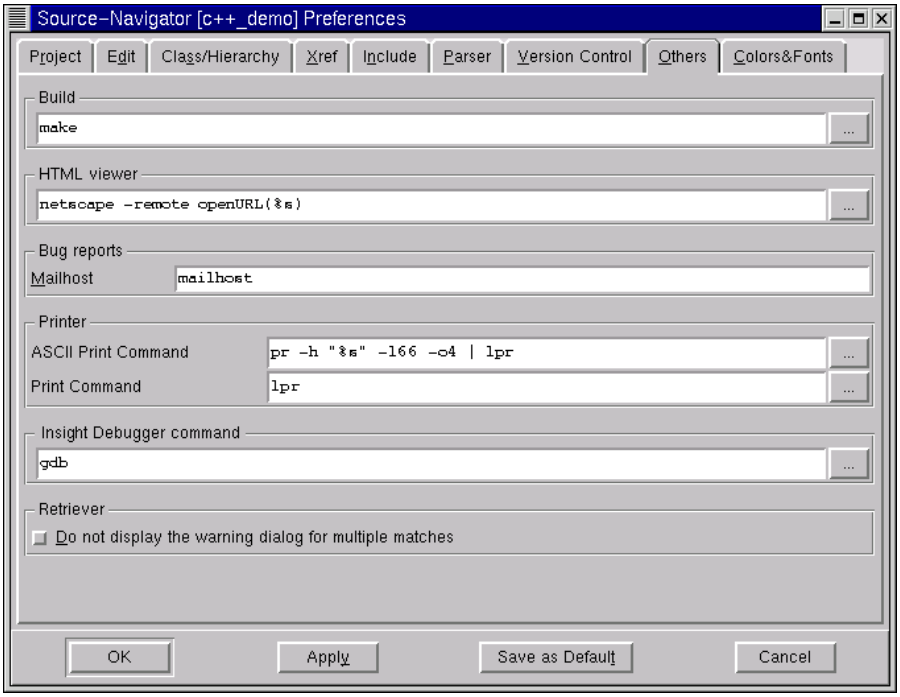
Macro files are ASCII files and every non−blank line is a macro directive. Leading blanks and # characters are stripped, and if the first character of a line is an apostrophe ('), the line is treated as a comment. The macro file can contain continuation lines, for example:

```
define ABC\
   \
   5
```

Otherwise, the directives are interpreted as documented above.

Others tab

Others Tab of the Preferences Dialog



Build

Enter the executable (shell) command used to start your **make** system.

HTML viewer (UNIX only)

Enter the HTML viewer to display online help. From Help, select Online Manuals to open the viewer, which must be included in your path.

Bug reports

Mailhost

Source−Navigator supports sending bug report emails by SMTP. Enter the name of your SMTP mail server in this field.

Printer (UNIX only)

ASCII Print Command

Enter the command you use when printing a source file from the command line.

Print Command

Enter the print command appropriate for your system. For example, Linux−based systems use **`lpr`**.

On Windows, Source−Navigator uses the Registry to decide how to print.

Insight Debugger command

Enter the executable (shell) command to start Insight.

Retriever

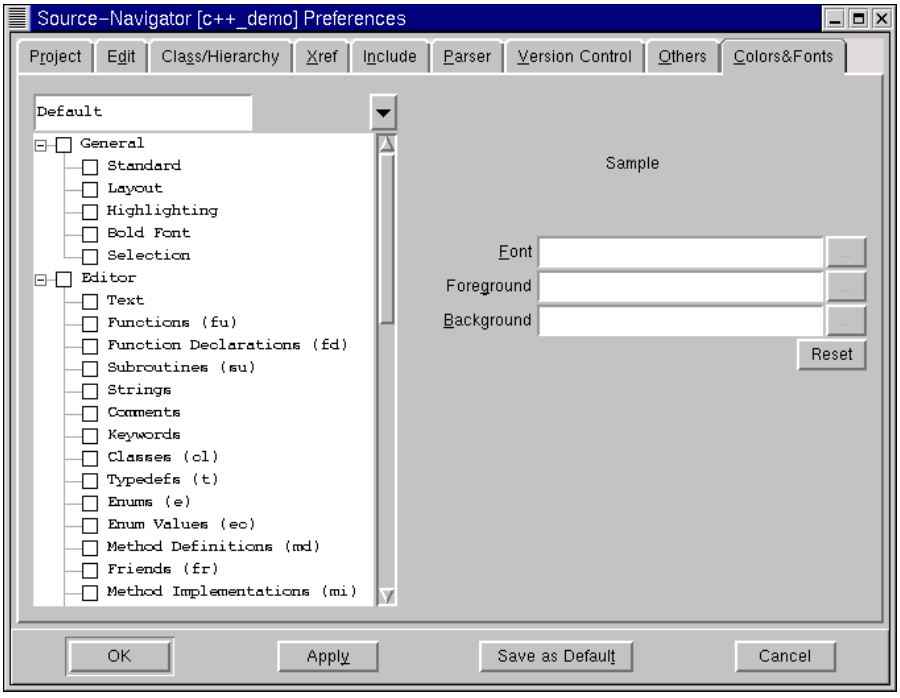
Do not display the warning dialog for multiple matches

Select this if you do not want to be warned when the Retriever finds more than one symbol with the name you are searching for. See Retriever for more information about the Retriever.

Colors & Fonts tab

Source−Navigator assigns a different default color to each component of your source code. To customize these colors or choose different fonts, click the Colors & Fonts tab of the Preferences dialog.

Colors & Fonts Tab of the Preferences Dialog

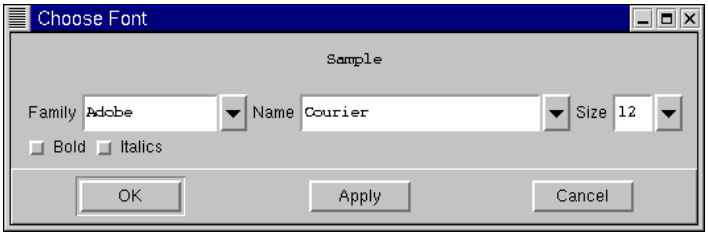


From the list of components, choose the item you want to change. The current settings appear in the Font, Foreground, and Background text boxes, and the word "Sample" displays the text with these settings.

Changing Fonts

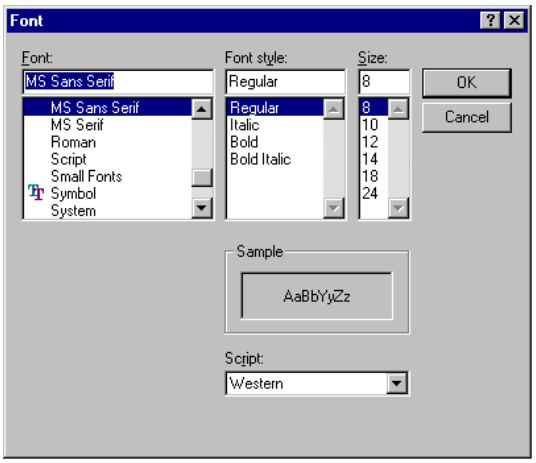
Changing the font assigned to a component is done differently, depending upon your platform.

Under UNIX, click the "..." button at the end of the Font text entry box. The Choose Font dialog appears:



Choose the Family, Name, and Size of font that you would like, as well as Bold or Italics, if you want to use those properties. Click the Apply button to see the effect of your changes and, when you are satisfied, click OK. Your changes take effect the next time you start Source−Navigator.

Under Windows, click the "..." button at the end of the Font text entry box. The Font window appears.

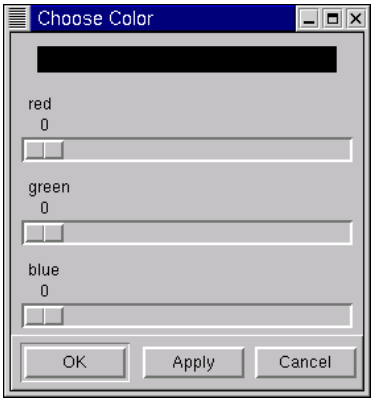


Choose the Font, Font style, and Size of the font to use. Your changes appear in the Sample box. When you are satisfied, click OK. Your changes take effect the next time you access the item you changed.

Changing Colors

You may want to change the Foreground or Background colors from their default settings if they do not show up well on your laptop or CRT display, or if your code colorization conventions do not match Source−Navigator's default colors.

To choose a different color for any component, you must set new RGB (Red−Green−Blue) values. Click the Foreground or Background "..." button. The Choose Color dialog appears.



Move the red, green, and blue sliders until the color you want appears in the box at the top of the dialog. Click the Apply button to see how your text or background looks with the new color. If you are satisfied, click OK. Your changes take effect the next time you start Source−Navigator.

Contents Previous Next
Contents Previous Next

Symbol Browser

The Symbol Browser window is displayed when you first create or open a project. It is the main window for navigating between Source−Navigator symbols. The Symbol Browser shows high−level information about the project, such as files, definitions, functions, variables, or classes/methods.

Symbol Browser Window

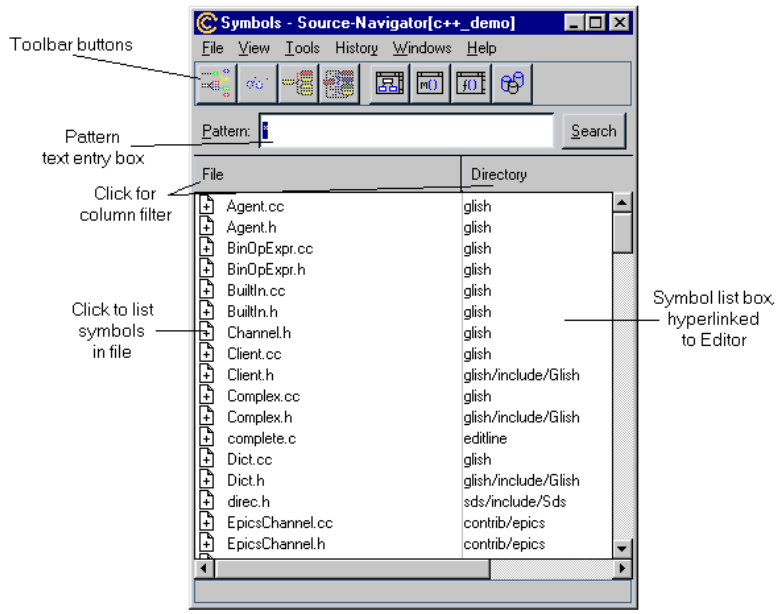
Using the Symbol Browser

Each symbol in the Symbol list box is hyperlinked to the Editor. Double−clicking on a symbol starts the Editor , which displays the symbol in its context in the source file. Depending on your settings in the Edit tab of the Preferences dialog, the cursor is positioned on the location where the symbol is declared or defined. For information on the Editor, see Editor.

Toolbar Buttons

Tool icons are located in the Symbol Browser under the main menu bar. To find out what a tool does, move the cursor over it; a tooltip appears describing the tool. The toolbar is shown in its default configuration in The Default Toolbar.

The Default Toolbar

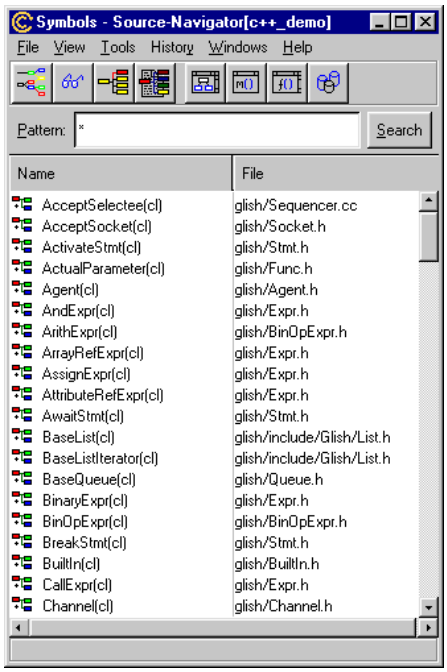


Browser buttons

With the toolbar, you can access many of the browsers you will use most often. However, you can add or remove toolbar buttons and menus. For more information, see the Customization chapter in the ***Programmer's Reference Guide***.

Toolbar buttons provide quick access to the following Source−Navigator functions:

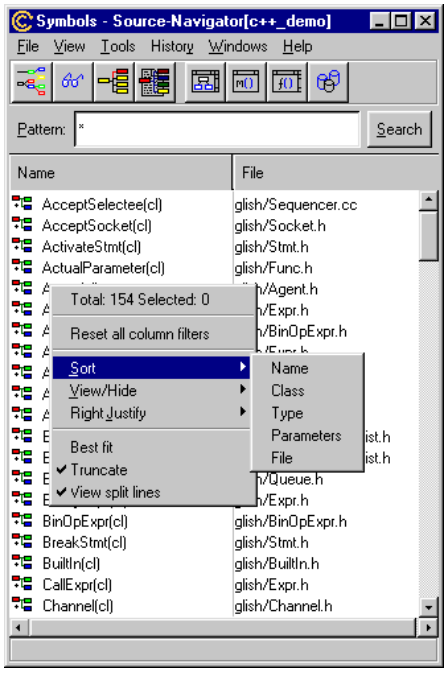
Hierarchy Browser button



Select this button to start the Hierarchy Browser. For more information about this tool, see Using the Hierarchy Browser.

Class Browser button

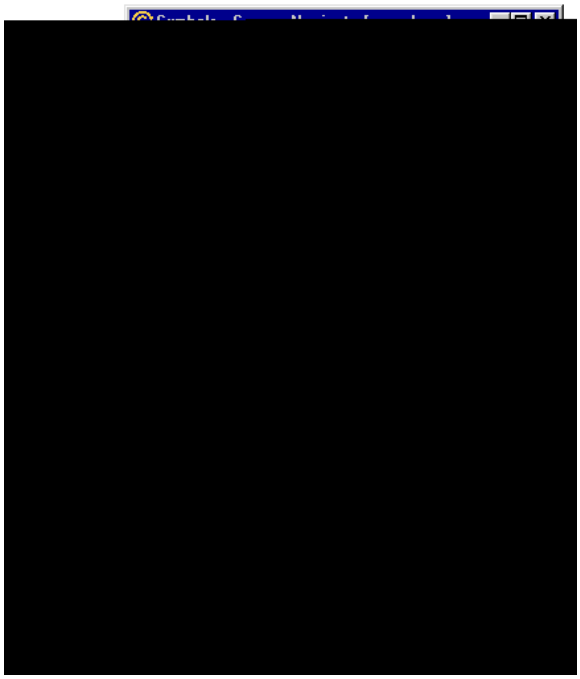
This button starts the Class Browser. For more information about this tool, see Class Browser.



Cross−Reference Browser button

Select this button to start the Cross−Reference Browser. For more information about this tool, see Cross−Reference Browser.

Include Browser button



Select this button to start the Include Browser. For more information about this tool, see Include Browser.

List Filter buttons

The List Filter searches for specific types of symbols in the symbol database. Use the List Filter buttons to restrict searches by symbol type in the Symbol Browser.

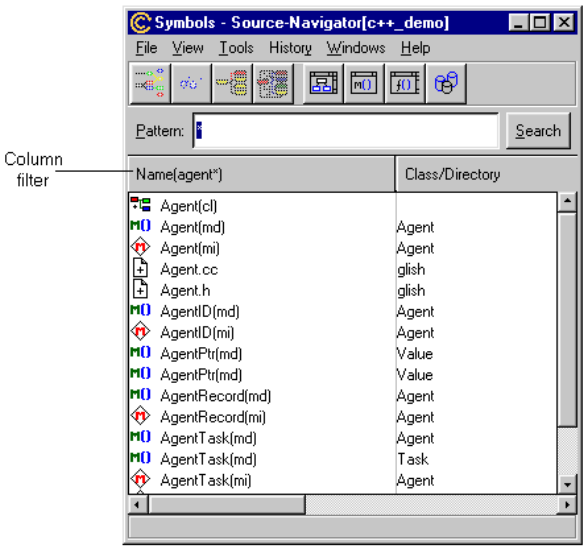
Filter Toolbar Buttons

For details on how to filter for other symbols, see Using Filters.

Symbol Filters

The Symbol list box displays different kinds of project symbols, depending on which type of symbol you select from the Symbol−type selector (accessed through the View menu). If Exclusive Search is selected, then only one symbol type is displayed at a time. If Exclusive Search is deselected, then clicking additional symbol−type selectors adds those symbols to the existing contents of the Symbol list box.

Symbol Browser with Exclusive Search and Classes Selected



To find out the number of symbols matching your selection, right−click on the symbol list. From the popup menu you may sort the symbol list by column and hide columns to simplify the display.

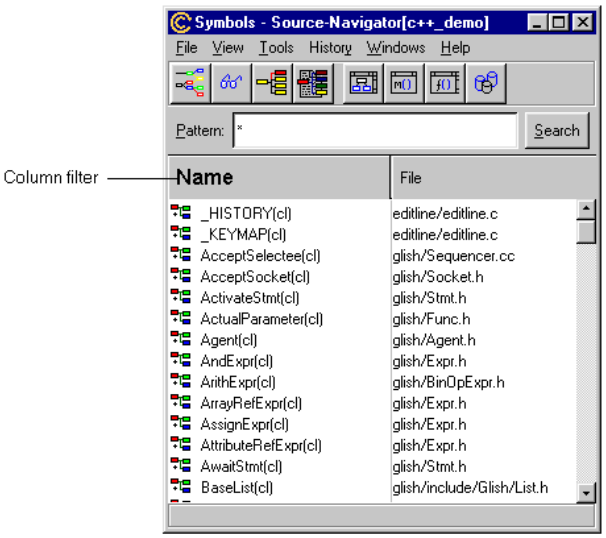
Symbol Browser Right Button Menu

For faster searching you can use the toolbar buttons to browse project files, functions, methods, and classes. For more information on the toolbar see Toolbar Buttons.

Column Filters

Column filters override all other filter preferences, and allow you to constrain the contents of a list view by filtering by a certain pattern. They are available in any window where information is presented in columns, such as the Symbol Browser, Class Browser, and Retriever windows.

Right−click on the column header and it is replaced by a column filter box.



Type in a pattern and press the Enter key. The list displays all symbols matching your pattern.

If you left−click on the column header, it sorts the data by that column.

Contents Previous Next
Contents Previous Next

Editor

Source−Navigator allows you to edit source files with its built−in editor, or with an external editor that you select in the Edit tab of the Preferences dialog (see Editor Preferences). For instructions on how to use Emacs as your external editor, see Using Emacs as your Editor. If you intend to use another external editor, see the Customization chapter in the ***Programmer's Reference Guide***.

The Editor Window

To open the Editor, double−click a symbol in the Symbol Browser (or other Source−Navigator browser window). Symbols are hyperlinked to the Editor, which displays the contents of a project source file and allows you to edit it.

When you save a modified source file, the project database is updated and changes are reflected in all of the Source−Navigator tools. Standard mouse operations are supported in the Editor:

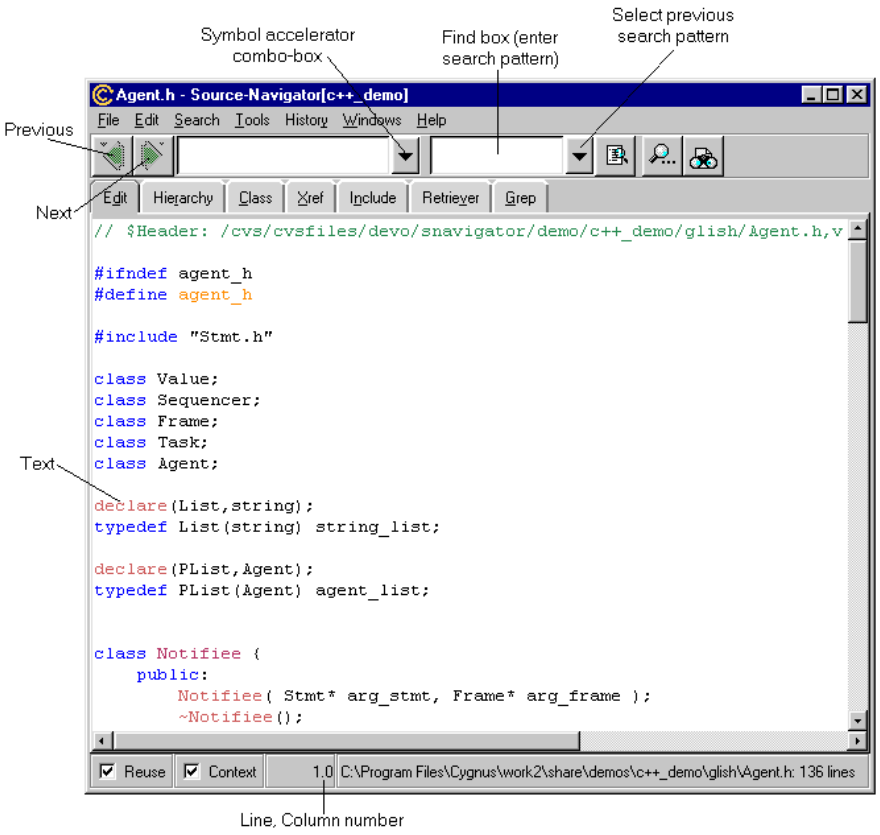
- click on the text and new text will be inserted to the right of the cursor.
- clicking and dragging selects text so that operations such as Cut, Copy, and Paste can be performed.
- clicking and dragging in the scroll bars scrolls the file appropriately.
- double−clicking on words selects the entire word; triple−clicking selects the entire line.

Editor Window

Symbol Accelerator Combo−box

The Symbol Accelerator combo−box in the toolbar allows you to quickly navigate through your source code.

Symbol Accelerator Combo−box



When the Editor is open this combo−box lists all of the symbols in the open file. When the All button is clicked, all of the symbols within the entire project are displayed. When other tools are open, the Symbol Accelerator combo−box lists the components relative to the tool.

For instance, in the Class Browser, it lists only classes in the file. When the All button is clicked, all of the classes in the project are displayed.

Notice that the Symbol Accelerator text field in the toolbar displays the component that is referenced as the Editor's cursor moves through the file.

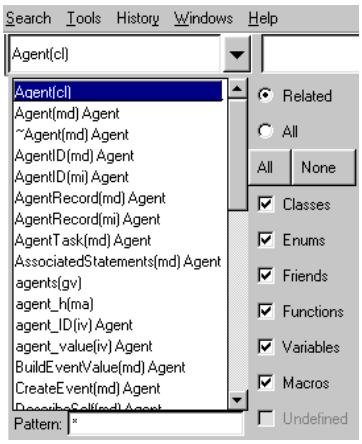
Find Box

You can use the Find box in the toolbar to search for text. Type text into the text box and press the Enter key; the next instance of the text is found. To find a previously used pattern, click the Find box down arrow to see a list of previous patterns. Select one of the patterns with the mouse and the next instance of that pattern is found.

Pattern Searching

Use these buttons to search the window or the project database for a symbol.

Pattern Searching Toolbar Buttons



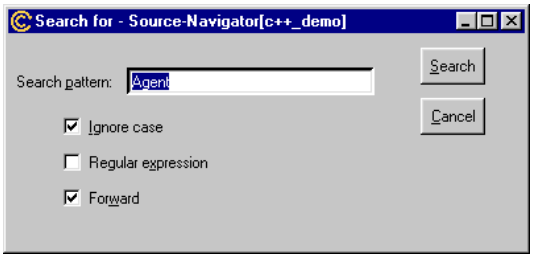
The Extended Toolbar

The extended toolbar provides buttons to manage your files and text. To add this toolbar to Source−Navigator, from the File menu, select Project Preferences. Select the Edit tab and select the Extended Toolbar Buttons box.

File and Text Management Toolbar Buttons

View History

Source−Navigator provides complex information in a number of ways. As you navigate through a project, you may want to return to the view of a relationship that you previously investigated. Source−Navigator stores a view history of your journey through the project. The left and right arrows in the toolbar act like Back and Forward buttons in popular Web browsers. You can also right−click on one of these buttons to get a list of previously visited locations within the project.



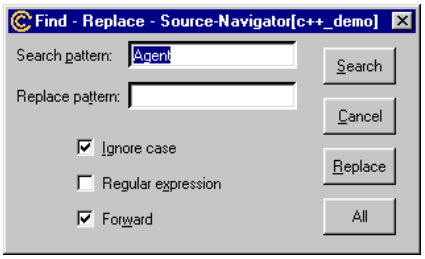
For more detailed history, the History menu lists previous views on a per−tool basis. This lets you jump directly to the view you want, rather than paging through previous views.

Search Menu

The Search menu is context−sensitive; different options are available depending upon the tool you are using.

Find dialog

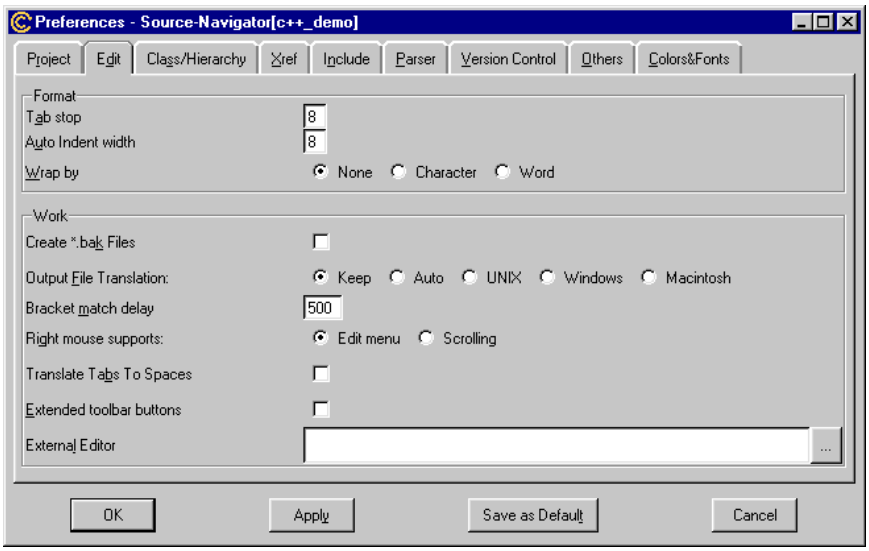
To find a specific string or pattern in the text file, from the Search menu, select Find.



Type your string or pattern into the text box and click the Search button; you can click the Search button multiple times to find more instances of the string. Deselect Ignore case if you want a case−sensitive search. When Regular expression is selected, the pattern is treated as a regular expression, and clicking the Search button finds the next match for the regular expression. Deselect Forward if you wish to search backwards.

Replace dialog

To search and replace a specific string or pattern in the text file, from the Search menu, select Replace.



The Replace dialog is similar to the Find dialog. The Ignore case, Regular expression, and Forward options behave the same way. Search finds the next instance of the pattern. Replace replaces the current pattern with the pattern in the Replace pattern text box. Select All to replace all occurrences of the pattern.

Find Declaration, Implementation

If a symbol is selected in the Editor, selecting Find Declaration switches to the location of the declaration of the symbol, and Find Implementation switches to the location of the implementation of the symbol.

Note

You can also toggle between Declaration and Implementation by using the keyboard shortcut commands Ctrl+Shift+D and Ctrl+Shift+I, respectively.

Grep

Grep activates the Grep tool, which allows you to search for text in all project files. Select the expression you want to search for and from the Search menu select Grep. Source−Navigator automatically searches for your text in all project files. For more information on Grep, see Grep.

Go To menu

When you want to go to a specific line in the file, from the Search menu, select Go To −> Go To Line. This allows you to type in the line number you wish to go to. Set Mark allows you to set a place in the file you wish to come back to later. Go To Mark jumps to the last mark you set. Go to Error displays the line that caused the build error.

Editor Preferences

You'll find preference settings for the Editor window in the Edit tab of the Preferences dialog. To find this dialog:

1. In the Symbol Browser, from the File menu, select Project Preferences. In the Editor, from the Edit menu, select View Preferences.
2. Choose the Edit tab.

Edit Tab of the Preferences Dialog

Format

Tab stop

When a tab is inserted, it is this many spaces wide.

Auto Indent width

When the Enter key is pressed, this is the number of spaces inserted at the beginning of the next line. This is also the number of spaces inserted by indent (from the Edit menu, select Indent Text) and deleted by outdent (from the Edit menu, select Outdent Text) when reformatting source code.

Wrap by

This controls where the Editor breaks a line that is longer than the width of the window.

Work

Create `*.bak` Files

If selected, the Editor creates backup files whenever you save a file.

Output File Translation:

End−of−lines of source files may be represented differently on different platforms:

- Keep retains the original file's end−of−line characters; this is set by default.
- Auto saves the file with your current platform's end−of−line characters.
- UNIX, Windows, or Macintosh sets the end−of−line characters to match the platform you choose, regardless of the platform you're working on.

Bracket match delay

Sets the amount of time (in milliseconds) that matching brackets should be highlighted.

Right mouse supports

If Edit menu is selected, you can access some functions, such as Undo, Delete, Cut, Copy, and Paste through a right−mouse pop−up menu.

If Scrolling is selected, you can scroll the text in the Editor using the right−mouse button.

Translate Tabs to Spaces

Selecting this converts all tab characters in the file to the number of spaces shown in the Tab Stop box.

Extended toolbar buttons

Selecting this adds several new tool buttons to the Editor toolbar. For more information, see The Extended Toolbar.

External Editor

Insert the command line for your favorite editor in this text box. See Common editor configurations for command line syntax.

Source−Navigator can set the position of the cursor in the editor if the editor can be configured by command line options. Source−Navigator can perform the following substitutions on the command line before it is executed:

%f file name

%l line number

%c column number

%d project directory

Common editor configurations

vi

To invoke the vi editor, you must enter the following into either the External Editor text box in the Edit preferences tab or at the command line:

```
xterm −T %f −e vi +%l %f
```

The modifications you make and save are stored in the database only after you quit vi.

Emacs

Starting a new Emacs session: To start a new Emacs session whenever you view source code, enter **emacs** or the name of the executable file of Emacs in either the External Editor text box in the Edit preferences tab or at the command line. For example, enter **nemacs** or **xemacs**, without any parameters. The string **emacs** must be found in the command if you want the changes you make and save to be immediately stored in the database (without terminating Emacs).

Using a current Emacs session: For instructions on how to configure Source−Navigator to communicate with a currently running Emacs session, see Using Emacs as your Editor.

For information on customizing your key bindings, see the Customization chapter in the ***Programmer's Reference Guide***.

Using Emacs as your Editor

Source−Navigator supports GNU Emacs 19.34 and XEmacs 19.14; other versions may also work, although these have not been tested.

When you use Emacs as your editor, Source−Navigator displays files in an Emacs window. Whenever Emacs saves a file, Source−Navigator updates the database. Multiple projects can share a single Emacs editing session.

You can use Emacs with Source−Navigator in one of two ways:

- to start a new Emacs process whenever you make an edit request.
- to communicate with an already running Emacs process.

To Start a New Emacs Process

Enter **`emacs`** (or the name of your program with the string **`emacs`**) in the External Editor text box of the Edit preferences tab.

To Communicate with an Already Running Emacs Process

1. Modify your Emacs start−up file so that **`gnuserv`** utility, which is provided in your Emacs distribution, is loaded. This involves adding two lines to your Emacs start−up file (usually **`~/.emacs`**). You need to enter the full path to your Emacs directory:

   ```
   (load "<path to emacs location>/lisp/gnuserv")
   (server−start)
   ```

   See your Emacs documentation for additional information.

2. In the External Editor text box, set your editor to **`gnuclient`**.

   When you start a new Emacs session, Source−Navigator can now request that the running Emacs session bring up files for editing. Source−Navigator also rescans the files when you finish editing.

Note

If you use **`xemacs`**, the **`gnuserv`** package is included; see your XEmacs documentation for instructions on loading it.

Source−Navigator's search function replaces the **`find−tag`** command (**`Meta−period`**) when you search for a symbol. Because the other tag commands are not yet available inside Source−Navigator, you need to use the equivalent **`emacs`** commands, if available.

Hierarchy Browser 

Inheritance, a type of relationship between objects, allows one object to share behavior with one or more other objects. Inheritance provides a basic mechanism for the reuse of code. Sharing with more than one is known as *multiple inheritance*.

The Hierarchy Browser can display the entire class hierarchy, including the superclasses and subclasses of a selected class. This helps you understand class hierarchy trees, which in turn helps you to reuse existing code.

A baseclass is a top−level class in the class hierarchy. It does not inherit from any other class; other classes inherit from it.

A class a is said to be a superclass of class b when class b inherits from a or another class that inherits from a.

A class a is said to be a subclass of class b when class a inherits from b or another class that inherits from b.

If class a is a superclass of class b, then class b is a subclass of class a.

Using the Hierarchy Browser

Start the Hierarchy Browser in one of the following ways:

- from the Windows menu, select New View −> Hierarchy.
- click the Hierarchy toolbar button.
- choose the Hierarchy tab in the Editor.

Hierarchy Browser Window

Tools Menu

The Hierarchy menu, accessed from the Tools menu, contains the following items, which control how the class hierarchies are displayed.

Show Superclasses

Limits the hierarchy to the superclasses (and their subclasses) of the selected class.

Show Subclasses

Limits the hierarchy to the subclasses of the selected class.

Show All

This is the default for this menu.

Display file names

Displays the file names of the superclasses and subclasses.

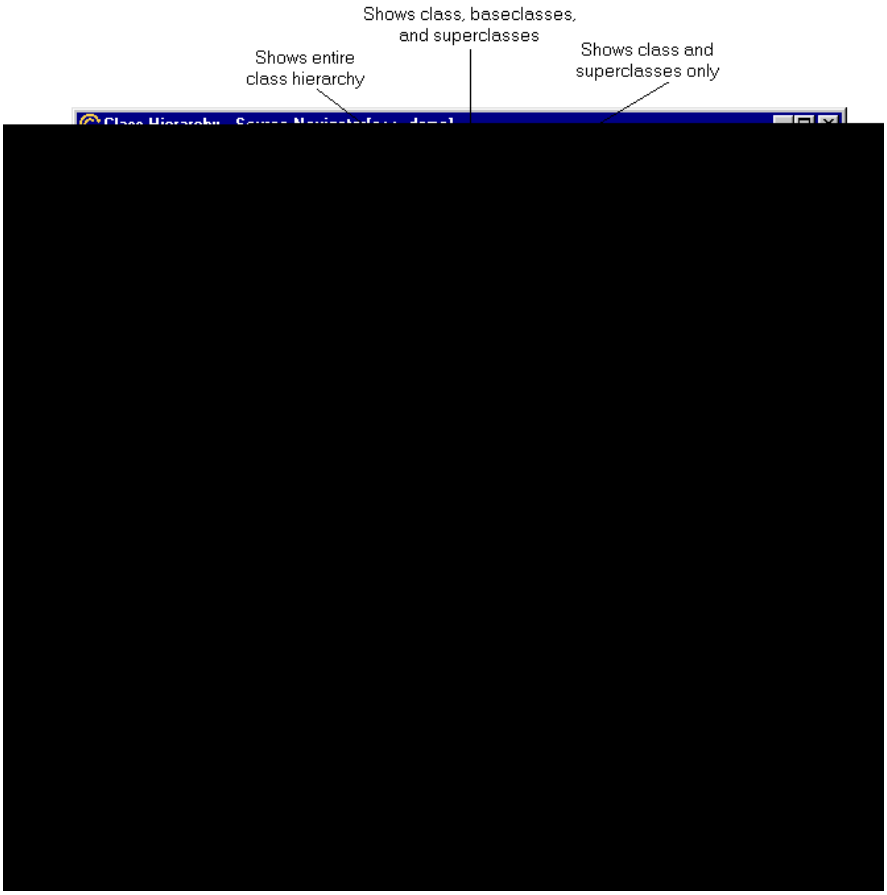
Class/Hierarchy Preferences

Many of the settings that control how the Hierarchy Browser window functions are located in the Class/Hierarchy tab of the Preferences dialog. To find this dialog:

1. In the Symbol Browser, from the File menu, select Project Preferences. In the Hierarchy Browser, from the File menu, select View Preferences.

2. Choose the Class/Hierarchy tab.

Class/Hierarchy Tab of the Preferences Dialog



Class

Go To

Select Declaration if you want the Class Browser to display the prototype of the function; select Implementation if you want to see the actual code.

Orientation

Select Horizontal to have the Hierarchy Browser appear below the Class Browser; select Vertical to have the Hierarchy Browser appear to the right of the Class Browser.

Display members

Select First to cause instance variables to appear before methods in the Class Browser; select Second to cause methods to appear first, with the instance variables after them.

Hierarchy Layout

Display order:

Left to right displays the hierarchy from left to right in the main window; Top to Bottom displays the hierarchy from top to bottom.

Display layout style:

Select Tree to display the hierarchy in tree layout; select ISI to display the hierarchy in ISI layout.

Vertical space:

Enter the number of vertical pixels between symbols in the Hierarchy Browser window.

Horizontal space:

Enter the number horizontal pixels between symbols in the Hierarchy Browser window.

Hierarchy Browser Shortcut Keys

This shortcut key is available for use with the Hierarchy Browser.

Key Combination		***Function***
UNIX	*Windows*	
Meta+B (or Alt+B)	Alt+B	Starts the Class Browser for the marked class.

Class Browser 

For projects developed using object−oriented languages, the Class Browser enables you to browse class hierarchies, access levels, and member types. The Class Browser displays the list of class members of a particular class, based on your selections from the Class/Hierarchy tab of the Preferences dialog (see Class/Hierarchy Preferences).

For traditional languages such as C, COBOL, and FORTRAN, the Class Browser enables you to see the members of structures and common blocks.

Note

Source−Navigator treats structures, classes, and common blocks in the same way. The only difference is that classes have inheritance and the others do not.

Using the Class Browser

Start the Class Browser in one of the following ways:

- double−click on a class.
- select a class and click the Class Browser toolbar button (see Class Browser button).
- from the Windows menu, select New View −> Class.
- choose the Class tab in the Editor.

Class Browser Window

Class Name

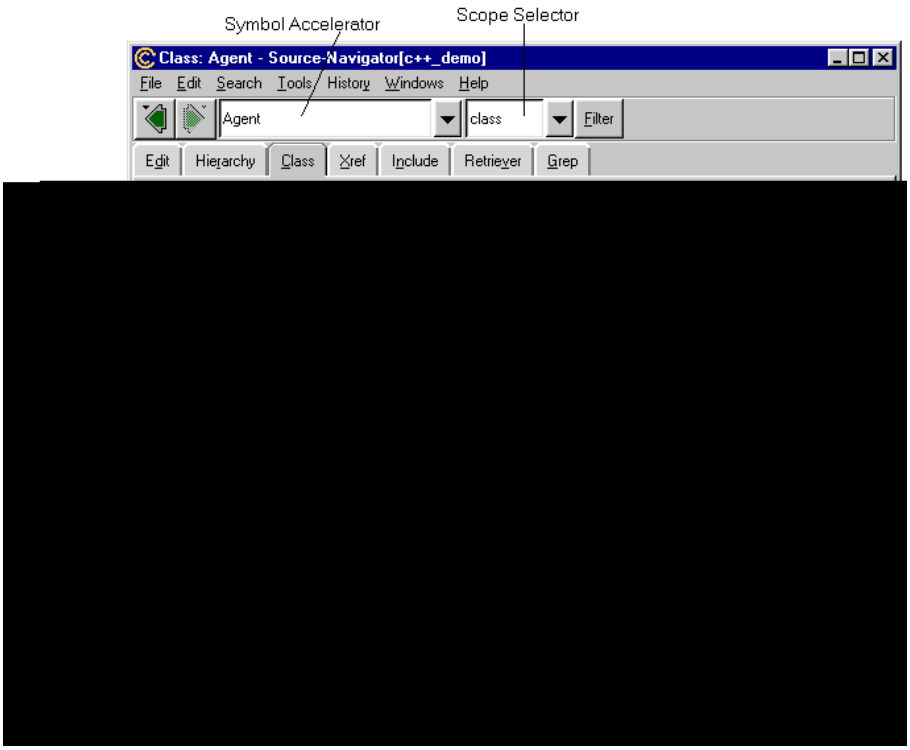
You can enter the class name into the Symbol Accelerator combo−box (emacs−style tab completion is also supported). If you press the Enter key and the name matches a valid class name, the information for the appropriate class is loaded.

Member List

The symbols displayed in the member list are controlled by the pulldown menus and inheritance tree. Access levels and attributes are indicated by icons; for the key to these icons select Abbreviations from the Help menu, or see Abbreviations Panel.

Inheritance Tree

The inheritance tree shows the relationship of the browsed class and its baseclasses.



The check boxes before the class names determine whether or not members of a class are included in the member list. Use the mouse to manipulate these check boxes:

Click

Toggles the check box.

Ctrl+click

Includes only the members of the selected class.

Double−click

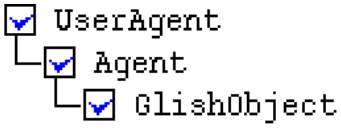
Starts the Editor, which displays the source file.

Right−click

Displays a menu in which you may select one or all classes.

Member List Filter Dialog

Click on the Filter button to bring up the Filter dialog. The symbols displayed by the Member List are included based upon these settings.



All

Sets all selections.

None

Clears all selections.

Methods, Instance Variables, Friends

Shows methods based on their types.

public, private, protected

Shows members based on their access level.

AND

If AND is selected, only functions matching *all* attributes will be shown. If AND is not selected, functions matching any of the attributes will be shown.

static, structor, inline, virtual, pure virtual

Shows members based on their attributes.

overridden

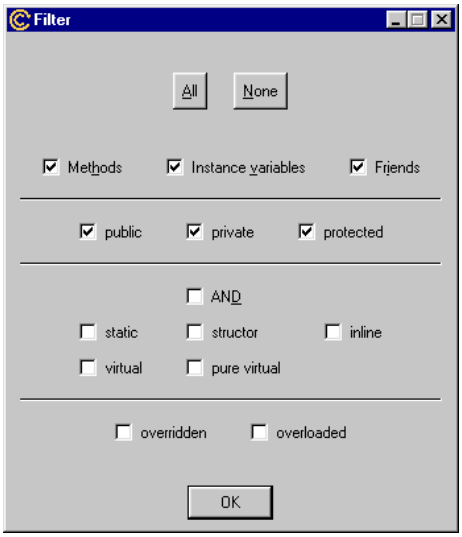
Shows members that are overridden from a base class. You can also display these by selecting the overridden checkbox in the main window.

overloaded

Shows functions that have more than one type signature in the class.

Scope Selector

The Scope Selector menu filters the member list by the accessibility of the members.



subclass

Shows only the members accessible to new subclasses of the currently browsed class. Does not include private members of the currently browsed class or private base classes.

class

Shows only the accessible members of the currently browsed class; private members of base classes are not included.

baseclass

Shows all members, including the private members of the base classes.

Contents Previous Next
Contents Previous Next

Cross−Reference Browser 

The Cross−Reference Browser shows you where elements in your program are used or accessed. It can find every call of a function, or tell you everything a particular function calls. It creates tree diagrams that show essential relationships within the project's symbol database, such as the function call hierarchy tree. You can traverse up and down the hierarchy tree, as well as expand or restrict the tree. You can select items in the hierarchy and display their *Refers−to* and *Referred−by* relationships; these relationships are based on the "point−of−view" of the selected symbol.

A *Refers−to* relationship is one where the selected symbol is used in the context of another symbol, which is in turn *Referred−by* the selected symbol.

Source−Navigator creates the cross−reference database in the background, which enables you to work in other views. During this process, the Cross−Reference tool button is disabled (grayed−out). After the database is built, the Cross−Reference tool can be opened.

Although you can always start the Cross−Reference Browser from the Windows menu by selecting New View −> Xref, you may want to start the Cross−Reference Browser so that it focuses on a specific symbol. To do this, select a symbol in the Symbol Browser or Editor, and then click the Cross−Reference tool button or choose the Xref tab.

The selected symbol becomes the root symbol in the Symbol Accelerator text box at the top left of the Cross−Reference Browser window. The references that *Refer−to* the root symbol are indicated by connecting lines and those that are *Referred−by* are indicated by connecting arrows. You can traverse the hierarchy tree by selecting references and clicking the right−pointing hand tool (*Refers−to*) and left−pointing one (*Referred−by*) as shown in Cross−Reference Browser Window.

Cross−Reference Browser Window

The Remove Subnodes button allows you to remove displayed subnodes from the hierarchy tree view.

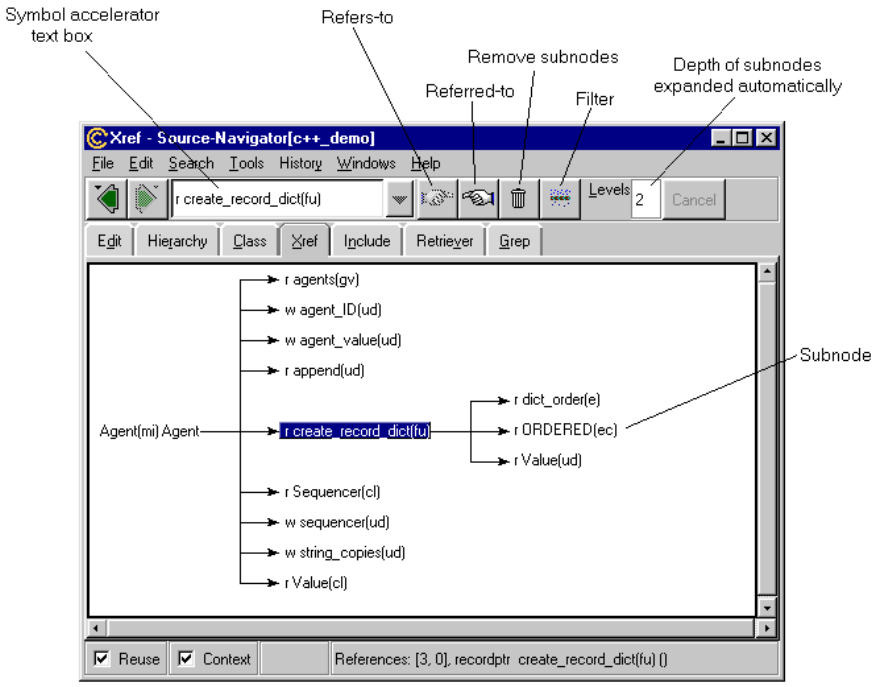
To set the number of subnode levels to display, enter a positive integer in the Levels text entry box.

Double−clicking a symbol in the Cross−Reference Browser window starts the Editor, with the specific symbol in context in the source file. The cross−reference information is stored in the database and is kept current by the Editor.

Cross−Reference Filter

Click the Filter icon to bring up the Filter dialog. The symbols displayed by the Cross−Reference Browser are included based on these settings.

Cross−Reference Filter



All

Sets all non−Access selections.

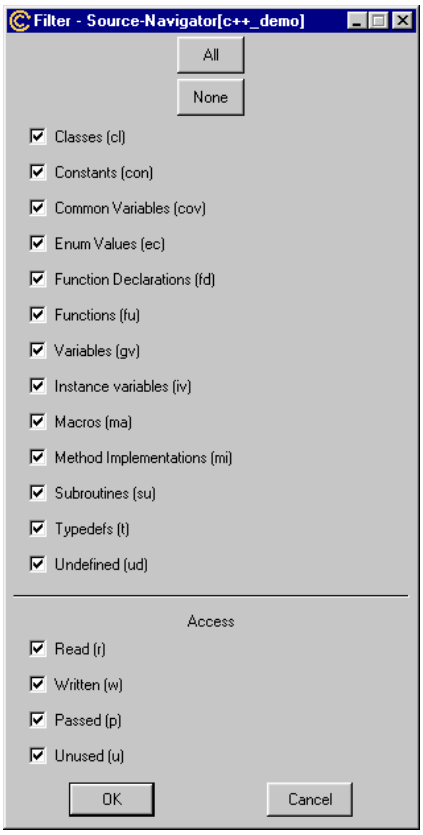
None

Clears all non−Access selections.

Cross−Reference Browser Details

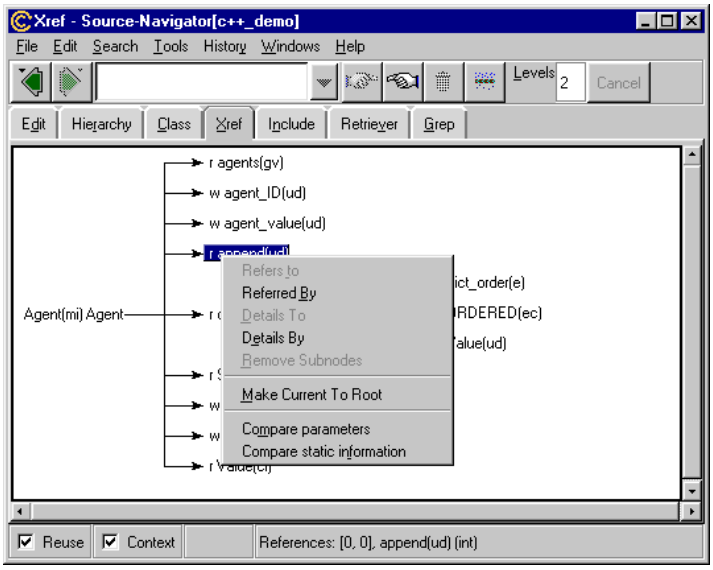
Holding down the right mouse button on a symbol in the Cross−Reference Browser brings up a popup menu that allows you to filter the list of symbols you're working with, as well as to gather new information about the symbols you're interested in.

Cross−Reference Browser, Right Mouse Button Down



Choosing Details By brings up a window that shows where each symbol in the list is referenced.

Cross−Reference Browser Showing Details By Window



Clicking on the column headers allows you to sort by the selected column, either alphabetically, by line number, by class, etc. The Pattern text entry box in the window allows you to use a string to filter the list.

Symbols that occur multiple times are listed; when you click on a symbol and then add the Editor to the window (from the Windows menu, select Add View −> Editor), the Editor shows where that symbol is referenced.

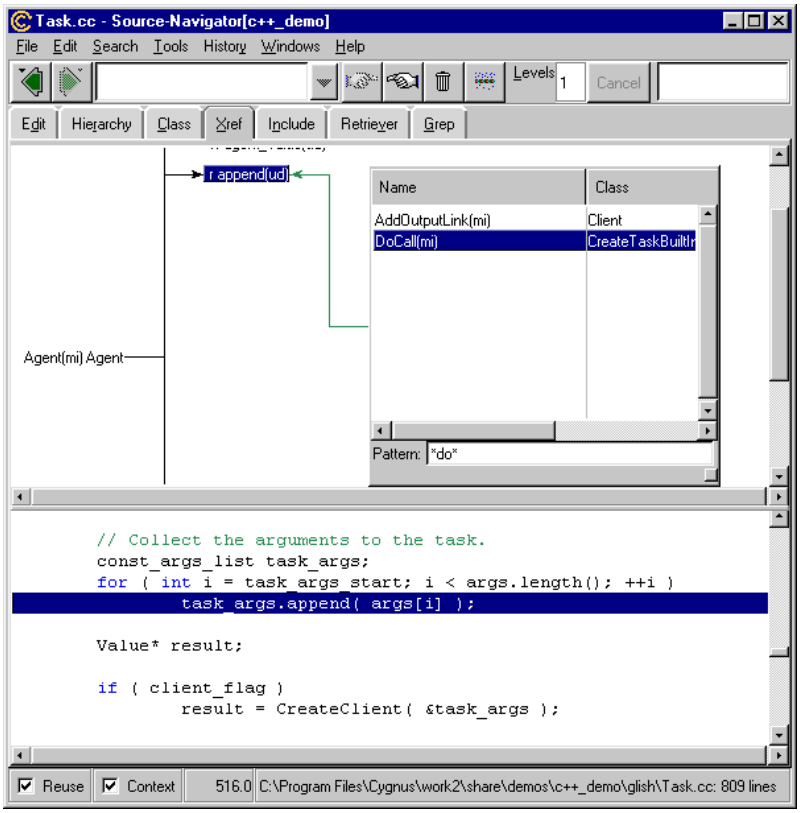
Editor Showing Referencing of Symbol

Cross−Reference Preferences

Preference settings for the Cross−Reference Browser are located in the Xref tab of the Preferences dialog.

1. In the Symbol Browser, from the File menu, select Project Preferences. In the Cross−Reference Browser, from the Edit menu, select View Preferences.
2. Choose the Xref tab.

Cross−Reference Tab of the Preferences Dialog



Cross−referencing

Build Cross−Reference database

Select this if you would like Source−Navigator to build the cross−reference databases for your project. This is on by default.

Generate references to local variables

Select this if you would like cross−reference information for local variables. This is off by default.

Note

Parsing is much slower when cross−referencing local variables. Also, the database size grows considerably when generating local variable cross−references. Make sure that you have an adequate amount of disk space (approximately 10 times the space used by your source code).

Audible alert when complete

Selecting this causes a bell to ring when cross−referencing is complete.

Layout

Compare parameters

Select this to generate cross−references only when the parameter types of the *Refers−to* and *Referred−by* symbols match. Deselecting this allows symbols to be considered matches, regardless of differences in parameters.

Compare static information

Select this to generate cross−references only when both the *Refers−to* and *Referred−by* static attributes match.

Display parameter list

Select this to display parameters with the symbol in the Cross−Reference Browser window.

Display boxes

Select this to surround cross−reference nodes with boxes.

Display order:

Left to right displays the cross−reference hierarchy from left to right; Top to Bottom displays it from top to bottom.

Display layout style:

Select Tree to display cross−references in tree layout; select ISI to display them in ISI layout.

Vertical space:

Enter the number of vertical pixels between symbols in the Cross−Reference Browser window.

Horizontal space:

Enter the number of horizontal pixels between symbols in the Cross−Reference Browser window.

Include Browser

Some programming languages provide a facility for including other source files. In C/C++ this is achieved using the **`#include`** preprocessor directive. The Include Browser lets you display *Includes* and *Included by* relationships simultaneously.

Using the Include Browser

Start the Include Browser in one of the following ways:

- from the Windows menu, select New View −> Include.
- click the Include toolbar button (see Include Browser button).
- choose the Include tab in the Editor.

Include Browser Window

To see further relationships, after selecting a file in the Include Browser window, use the right and left pointing−hand tool icons to show files included by the selected file (black connection arrows) and files that include the selected file (red connection arrows).

To determine the number of levels shown for a query, enter a positive integer in the Levels text entry box.

Reducing Displayed Information

Holding down the right mouse button on a symbol in the Include Browser window brings up a popup menu that allows you to show or hide include information.

Include Browser Window, Right Mouse Button Down

This is useful when the Include Browser displays more information than you need, and you'd like to hide all relationships but the particular one you're interested in.

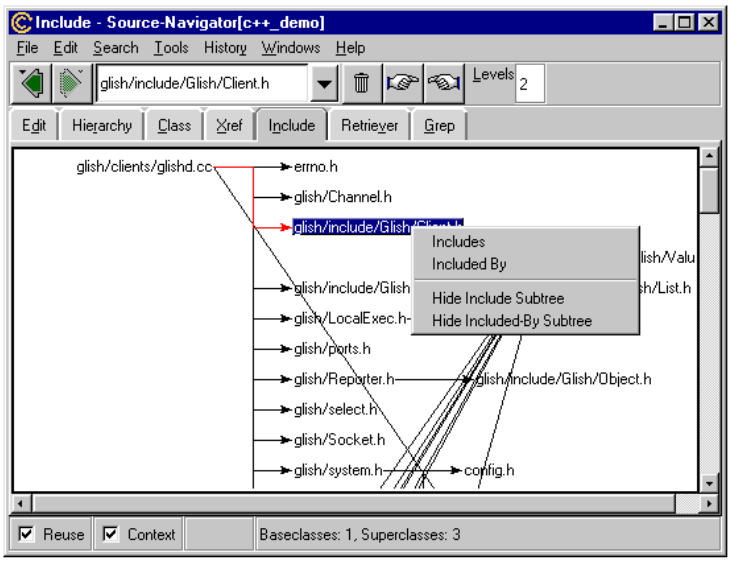
Include Preferences

You'll find preference settings for the Include Browser window in the Include tab of the Preferences dialog. To find this dialog, select one of the following:

1. In the Symbol Browser, from the File menu, select Project Preferences. In the Include Browser, from the Edit menu, select View Preferences.

2. Choose the Include tab.

Include Tab of the Preferences Dialog



Layout

Display order:

Left to right displays the include hierarchy from left to right; Top to Bottom displays it from top to bottom.

Display layout style:

Select Tree to display includes in tree layout; select ISI to display them in ISI layout.

Vertical space:

Enter the number of vertical pixels between symbols in the Include Browser window.

Horizontal space:

Enter the number of horizontal pixels between symbols in the Include Browser window.

Include directories

Locate Headers

Uncheck this box to prevent included files from being parsed.

In the Include Directories box you can choose which directories should be searched for include files. Using the set of directories shown in Include Tab of the Preferences Dialog, for example, if **`stdio.h`** is a reference then Source−Navigator looks first for the file **`/usr/include/stdio.h`**, then for **`./stdio.h`**, and so on down the list. The order of the list is important, because the first file found is the one that will be used by the Include Browser.

Contents Previous Next
Contents Previous Next

Retriever

Retriever allows you to search for text patterns in the names of symbols in the database. To search for text patterns in source files, use the Grep tool in Source−Navigator. For more information about the Grep tool, see Grep.

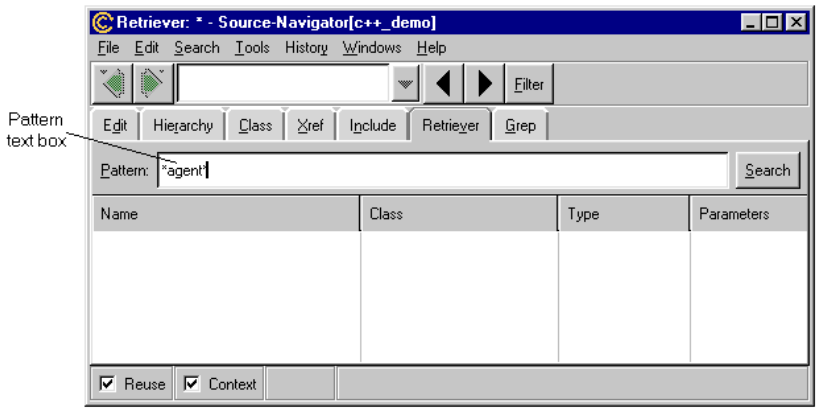
Using the Retriever

To find a symbol, enter its search pattern surrounded by asterisks (for example **`*agent*`**) in the Pattern text box and click Search. The **`*`**, **`?`**, **`[`**, and **`]`** wildcard characters are supported.

Retriever Window

The Retriever displays all the symbols it found containing the pattern being searched.

Retriever Window Showing Search Results

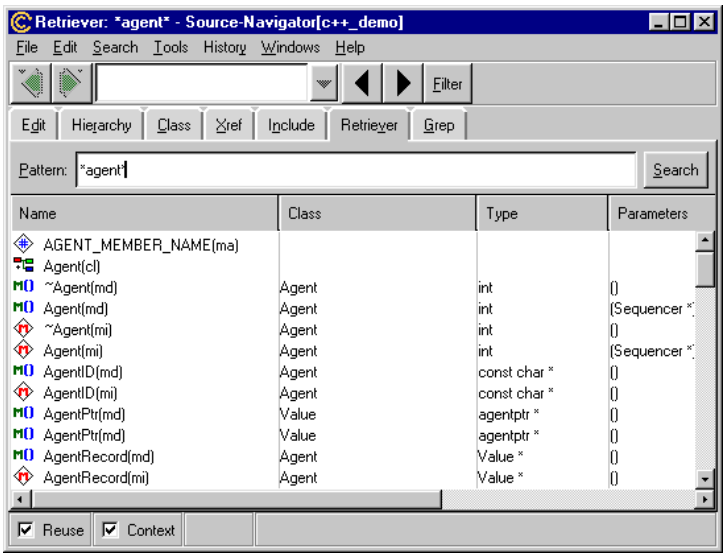


Double−clicking an item in the list opens the Editor showing the symbol in context in the source code.

For more information on reusing windows for multiple searches, see Reusing Windows.

Retriever Filter

Click the Filter button to bring up the Retriever filter. The symbols displayed by Retriever are included or excluded based upon these settings.



All

Selects all items except Unions and Files.

None

Clears all selections.

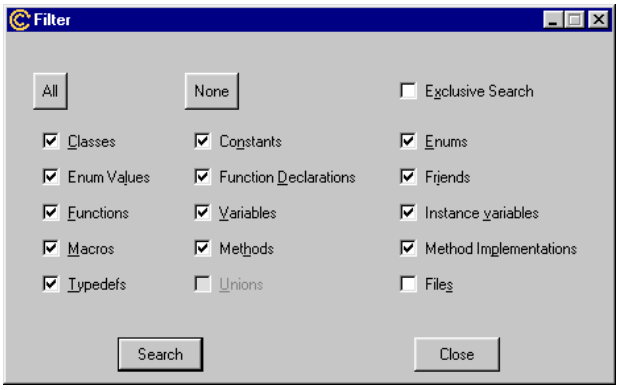
Executive Search

If the Exclusive Search box is selected, you may choose one symbol type to search for; if it's deselected, you may combine symbol types for more complex searches.

Retriever with the Cross−Reference Browser

If you're looking at a unique symbol in the Editor and you click the Xref tab to see its cross−references, the Cross−Reference Browser window appears.

However, if you're looking at a symbol that is not unique (there's more than one symbol with that name in the database), the Retriever displays a message dialog notifying you that it has found multiple matches and requests that you choose the correct one to display.



If you don't want to see this warning in the future, but want to go straight to the Retriever, click the check box in the dialog. This can also be changed in the Others tab of the Preferences dialog (see Others tab).

In the Xref Retriever window, double−click on the symbol for which you want cross−reference information, and the Cross−Reference Browser window appears (see Cross−Reference Browser for more information).

Grep

The Source−Navigator Grep tool allows you to search for text patterns in source files throughout the project. It is more powerful than using **`grep`** at the command line because it:

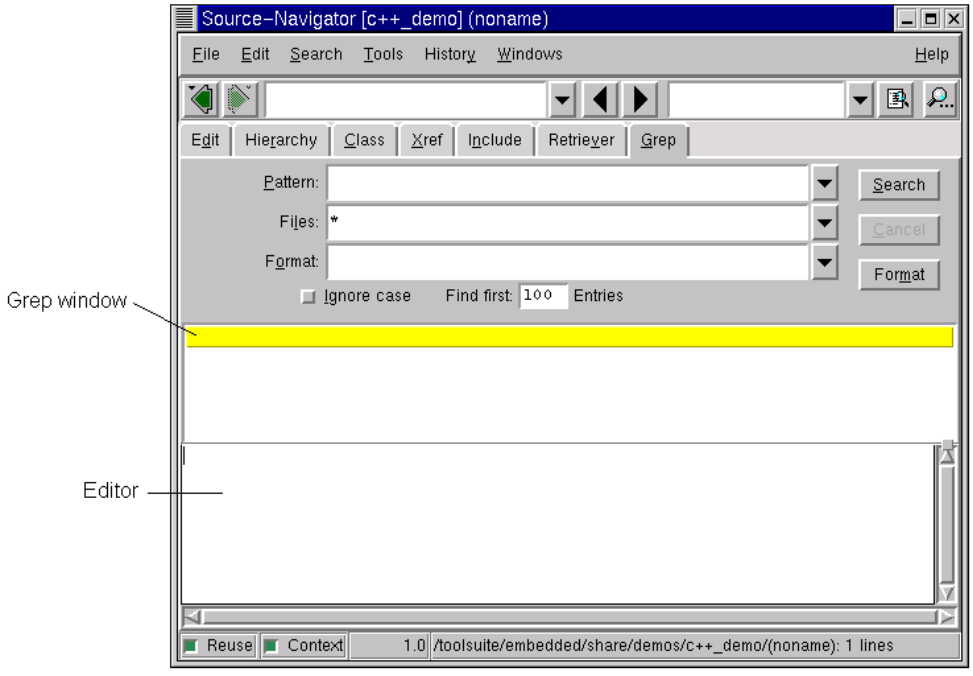
- allows you to search multiple directories.
- searches only the files in your project.
- can be restricted to a subset of the files in your project.
- saves results of multiple searches, so you can review them later.
- provides "one−click" Editor access to the lines of code matching your search.

To search for text patterns in the name of the symbols in the database, use the Retriever tool. For more information, see Retriever.

Using Grep

Start the Grep tool from the Windows menu by selecting New View −> Grep−Editor. This opens a Grep/Editor split window (for more information on split windows, see Adding a Browser to an Existing Window).

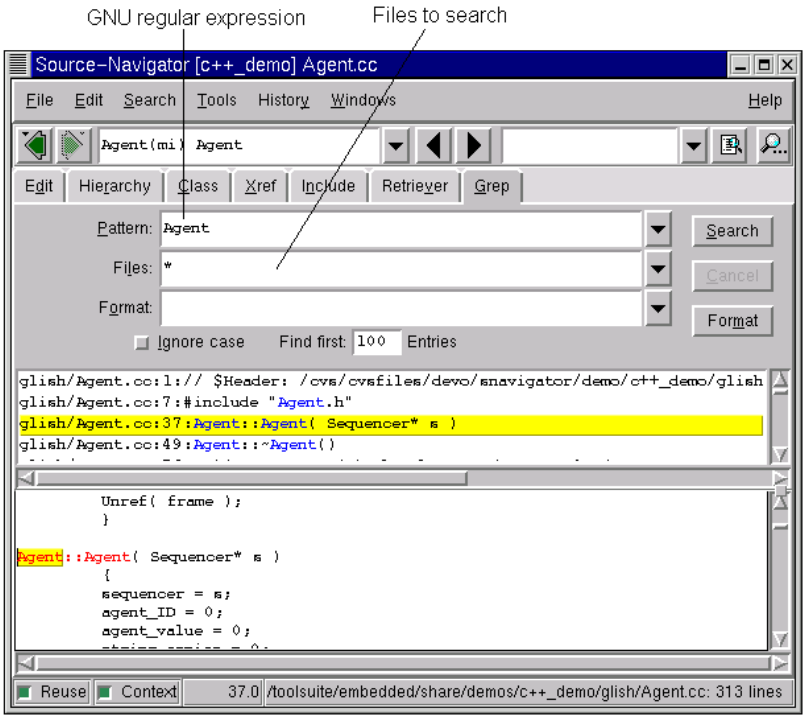
The Grep/Editor Window



In the Pattern text box, enter a regular expression then click the Search icon. For more information on regular expressions, see GNU Regular Expressions.

You can use the Files filter to limit your search; for example, entering ***.c** restricts the search to only C files. For a list of file extensions and their associated languages see File Types and Associated Filename Extensions. Sample Grep Search Results shows the results of a sample Grep search.

Sample Grep Search Results



Clicking on an item in the Grep window displays the appropriate file in the Editor , with the cursor positioned at the selected line.

To step through the Grep results, click the left or right black arrow keys in the toolbar. To filter your Grep results, use the Format combo−box to select an option.

GNU Regular Expressions

A GNU regular expression[1] is a pattern that describes a set of strings. Regular expressions are constructed like arithmetic expressions: various operators combine smaller expressions to form larger expressions.

Ordinary Characters

An ordinary character is a simple regular expression that matches a single character and nothing else. For example, f matches f and no other string. You can concatenate regular expressions together. For example, foo matches foo and no other string.

Special Characters

You can combine regular expressions with regular expression operators, or *metacharacters*, to increase the versatility of regular expressions. Traditional expression characters are enclosed by brackets [and].

For example, [Aa]pple matches either Apple or apple. A range of characters is indicated by a dash −. [a−z] matches any lowercase letter and [0−9] matches any digit string between 0 and 9. You can string groups together, so that [a−zA−Z0−9] matches any single alphanumeric character.

To represent the − dash itself, it must be the last character, directly succeeded by the]. To represent] bracket, it must be the first character after the [or the [^.

Special Characters lists special characters and examples of how to use them.

Special Characters

Symbol

Definition

Example

. (period)

matches any single character except a new line

a.b
matches any three−character string beginning with a and ending with b (such as acb, a6b, a#b)

[...]

matches characters between the brackets.

[af]
matches either one a or one f
[af]*
matches any string composed of just a's or f's or the empty string

[^...]

matches any character except the ones specified

[^a−z]
matches any characters except lower−case letters

^ (caret)

matches the empty string, but only at the beginning of the line

^foo
matches foo and food, but not ofoo

$ (dollar sign)

matches the empty string, but only at the end of the line

fo$
matches a string ending in fo, but not foop

\ (backslash)

escapes special characters (including \)

fo\?
matches fo?

| (pipe)

is used to designate OR

foo|bar
matches either foo or bar

(...)

is a grouping construct

foo(bar)*
matches zero or more instances of bar with foo (such as foo, foobar, and foobarbar)

Note

In the | (pipe) example above, notice that foo|bar is matching either foo or bar. It's not matching o or b, resulting with either fooar or fobar.

Predefined Sets of Characters

Certain named classes of characters are predefined, but can only be used within bracket expressions.

Character Classes

Symbol

Definition

C−locale Equivalent

[:alnum:]

alphanumeric characters

[a−zA−Z0−9]

[:alpha:]

alphabetic characters

[a−zA−Z]

[:blank:]

space and tab characters

[:cntrl:]

control characters

[:digit:]

numeric characters

[0−9]

[:graph:]

characters that are printable and are also visible (a tab is printable, but not visible, while an a is both)

[:lower:]

lower−case alphabetic characters

[a−z]

[:print:]

printable characters (ASCII 32 and above), but not control characters

[:punct:]

punctuation characters

[:space:]

space characters (such as space, tab, newline, and page eject)

[:upper:]

upper−case alphabetic characters

[A−Z]

[:xdigit:]

hexadecimal digit characters

[0−9a−fA−F]

For example, [[:alnum:]] means [0−9A−Za−z] in the C−locale, except the latter form is dependent upon the ASCII character encoding, whereas the former is portable.

Repetition

A regular expression matching a single character may be followed by one of several repetition operator:

Interval Expressions

Symbol

Description

Example

* (asterisk)

post−fix operator that matches an expression 0 or more times

fo*
matches a string starting with f and ending with a repeating o or no o's (such as f, fo, foo)

+ (plus)

post−fix operator that matches an expression at least once

f+o
matches a string starting with one or more f's and ending with an o (such as fo, ffo, and fffo)

? (question mark)

post−fix operator that must match an expression once or not at all

f?o
matches fo or o

{n}

preceding item is matched exactly *n* times

fo{2}
matches foo

{n,}

proceeding item is matched *n* or more times

fo{2,}
matches foo and fooo

{,m}

proceeding item is optional and is matched at most *m* times

fo{,3}
matches f, fo, foo, and fooo

{n,m}

proceeding item is matched at least *n* times and at most *m* times

fo{1,3}
matches fo, foo, and fooo

Escape Sequences

Some characters cannot be included literally in regular expressions. You represent them instead with escape sequences, which are characters beginning with a backslash (\). A backslash is also part of the representation of

unprintable characters such as a tab or newline.

Escape Sequences

Symbol

Description

\\

a literal backslash

\a

alert

\f

formfeed

\n

newline

\r

return

\t

horizontal tab

\v

vertical tab

\?

question mark

\(

left parenthesis

\)

right parenthesis

\[

left bracket

\]

right bracket

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated subexpression. For example:

[a−b] matches either a or b

[d−e] matches either d or e

[a−c][d−e] matches ad, bd, ae, or be

The backreference \n, where n is a single digit, matches the substring previously matched by the nth parenthesized subexpression of the regular expression. For example, \(ab)c\1 matches abbbcabbb, but not abbbcabb.

For additional information on regular expressions, please refer to a reference text such as ***Mastering Regular Expressions***[2]***.***

---

1. Richard Stallman and Karl Berry wrote the GNU regex backtracking matcher. Copyright © 1989, 1991 Free Software Foundation, Inc., 675 Massachusetts Avenue, Cambridge, MA 02139, USA. Return to text

2. Friedl, Jeffrey E. F. 1997. ***Mastering Regular Expressions***. ISBN 1−56592−257−3.

Contents Previous Next
Contents Previous Next

Version Control Systems

Source−Navigator provides a graphical user interface to several external version control systems. Source−Navigator allows you to manage version control tasks on a project level. With version control you can organize your development to manage versions, version history, labels, and related documents.

Version control systems use locks to prevent the same files from being modified by two developers simultaneously. If a lock is used while revising a file, no one can modify the file until it is unlocked. Locking and unlocking can be controlled by checking in or checking out versions.

For more information, see the Integrating with Version Control Systems chapter in the ***Programmer's Reference Guide***.

Using Version Control

The following version control systems have been integrated with Source−Navigator: GNU Revision Control System (RCS), Concurrent Versions System (CVS), and the Source Code Control System (SCCS). It has also been integrated with Rational's ClearCase version 3; other versions may also work, although these have not been tested. When creating a project, you must specify the correct version control system being used to manage the body of source code you wish to analyze. You make your selection in the Version Control tab of the Preferences dialog (see Version Control Preferences).

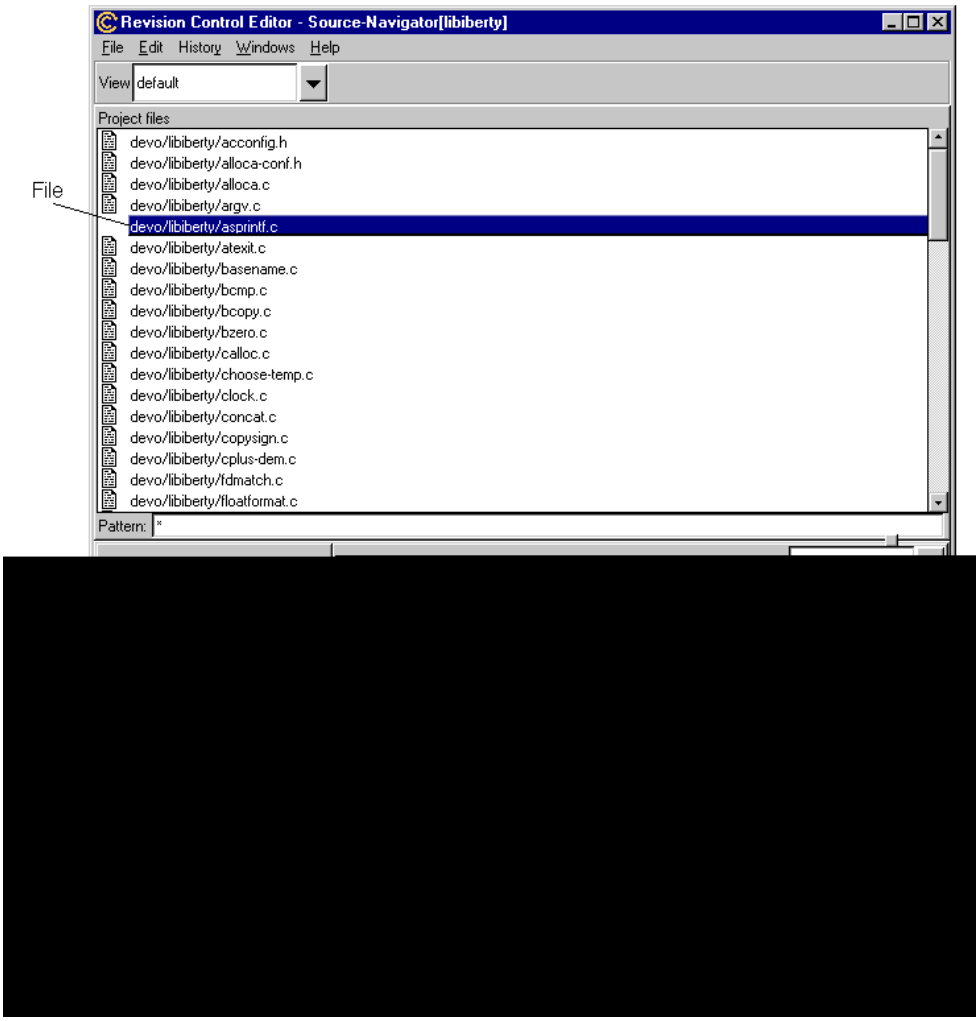
To start version control, from the Tools menu, select Revision Control −> Revision Control Editor (see Revision Version Control Window).

Revision Version Control Window

Checking Out a File

In the Editor, from the Tools menu select Revision Control −> Check Out to check out a file. Versions of a file can be checked out for modification either locked or unlocked. Checking out with With lock selected in the Check Out dialog box prevents other users from checking out the same version in write mode.

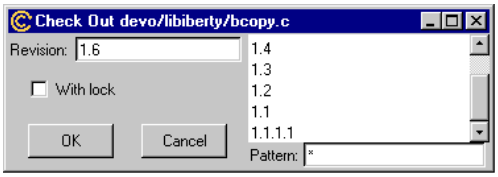
Check Out Dialog Box



Checking In a File

You can check in all or selected project files into your version control system. When you check in, you can enter descriptive text of the changes and a version number. Using the left−mouse button, select one or more files to check in. With these files highlighted, in the Revision Control Editor window, from the Edit menu, select Check In. The Check In dialog is displayed.

Check In Dialog Box



If you check in with With lock selected in this dialog box, others may not check out the same file in write mode.

This is useful if you are continuously working on a particular file, but wish to register checkpoints in your work without giving other developers the opportunity to make modifications to that file.

Discarding Changes to a File

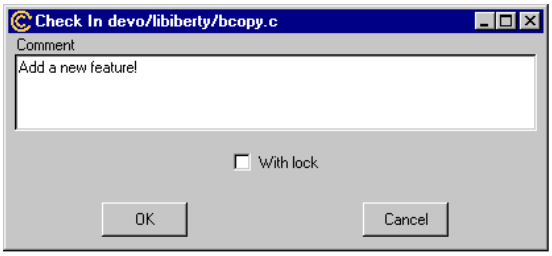
To revert working files to the repository version, in the Revision Control Editor window, from the Edit menu, select Discard Changes.

Show Differences

The Diff tool highlights differences between the current version of a file (the one you have checked out) and another one that you select from the list of available version numbers.

To access this tool, from the Edit menu select Compare Revisions.

Showing Differences



Version Control Preferences

Use the Version Control tab of the Preferences window to tell Source−Navigator what source code control system you're running. You can access this tab from the Symbol Browser from the File menu by selecting Project Preferences, or from the Version Control window from the Edit menu by selecting View Preferences.

Version Control Tab of the Preferences Dialog

Version control system

Select your external version control system for the project.

Ignored directories

These directories will be ignored by the Source−Navigator as it scans for files to parse.

Debugger

Source−Navigator works with the Red Hat Insight™ debugger, which is based on gdb, a popular open−source debugger. To get Insight, go to http://sourceware.cygnus.com/insight.

Launching the Insight Debugger

The following steps are necessary to start Insight from Source−Navigator:

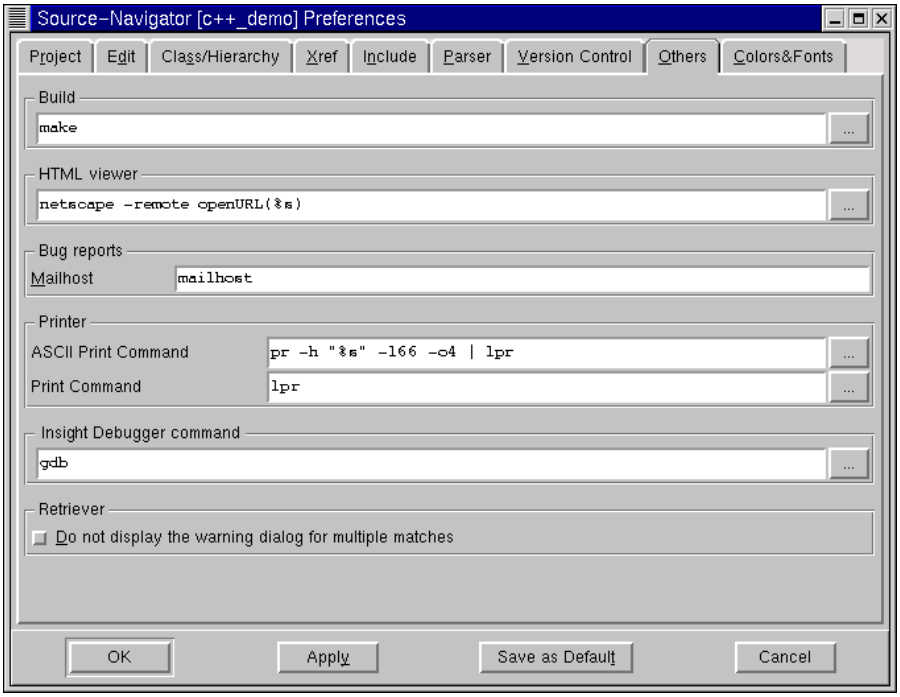
1. Launch Source−Navigator:

   On UNIX, change to the **<install directory>/bin** directory and type **./snavigator** to launch Source−Navigator *without* putting it in the background.
   On Windows, double−click the Source−Navigator icon (or snavigator.exe). It is located in **<install directory>/bin**.

2. Open the project to debug.

3. From the File menu, select Project −> Project Preferences −> Others to set the debugging preferences.

   Others Tab

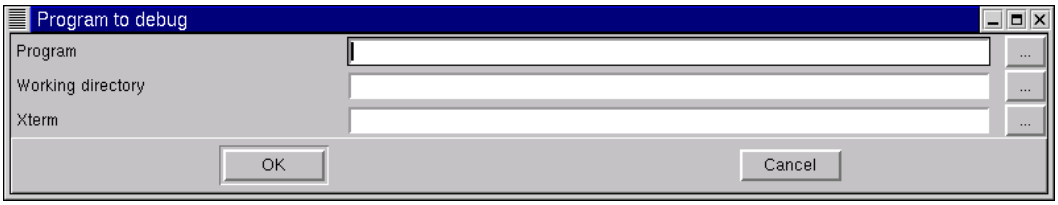


4. Enter **gdb** in the Insight Debugger Command text box.

5. Click OK to close the window.

6. From the Tools menu, select Debugger to launch the Program to Debug dialog box.

Program to Debug Dialog



7. In the Program field, enter the name of the application you want to debug; you can also click the "..." button and browse to the application in the Open dialog box.

   In the Working Directory field, enter (or browse to) a working directory.

   On UNIX, in the Xterm field, enter **`xterm -e`** if you are debugging a console application.

   The application name and command line you entered are saved in the project database, and are used as the default values the next time you launch the debugger from Source−Navigator.

8. Click OK to launch the debugger.

Building Programs

Source−Navigator allows you to build executable programs from the files in your project. Using Source−Navigator, you can compile your code, navigate to any errors, link your code, and, with the Insight debugger, set up a debugging session to debug your code.

Note

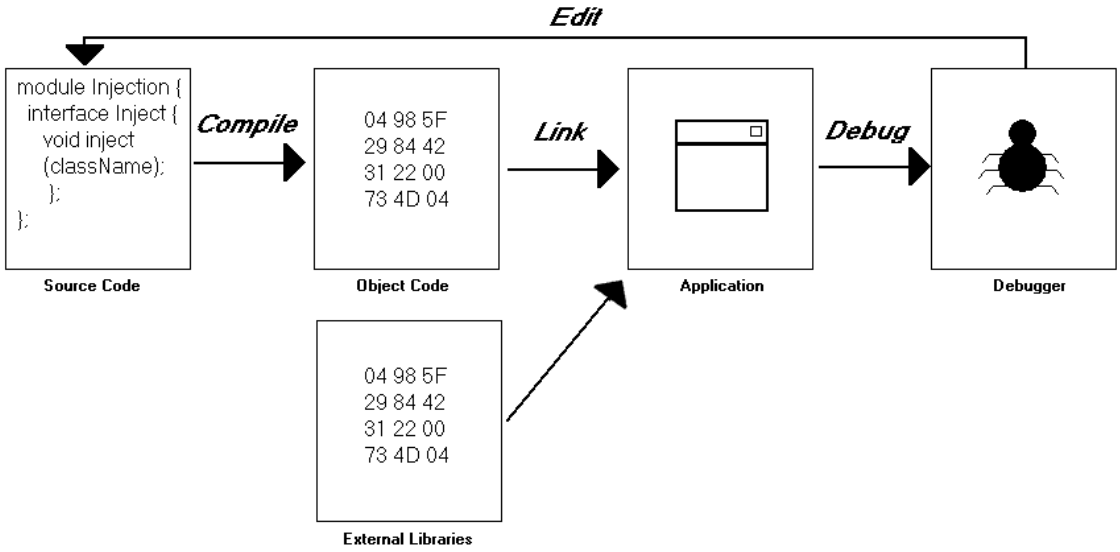
The compiler, **`make`**, and the Insight debugger must be installed on your machine before using these features.

At the end of this chapter, there is a build tutorial for a command line−driven real estate trading game.

The Building Process

The building process compiles and links source files, such as libraries and executable files, to produce an output binary file.

Build Process



There are four steps to building your program:

- editing your code,
- compiling your source into an intermediate format, called object files,
- linking your object files together to produce an executable application, and
- debugging your executable to find any problems.

The minimum requirements for building source code include specifying which source files should be included, the directory in which the build should be stored, linking rules, debugging, and optimization flags, and included paths.

You can edit your code using Source−Navigator. This chapter explains compiling and linking. For information about external stand alone debuggers, see either Debugger or your debugger's documentation.

make

Source−Navigator uses a utility called **make**. **make** combines a set of rules for compiling and linking code with a tracking mechanism for determining which files must be compiled.

Source−Navigator generates a **makefile**, which **make** uses to determine which commands need to be executed in order to build your program.

Build Targets

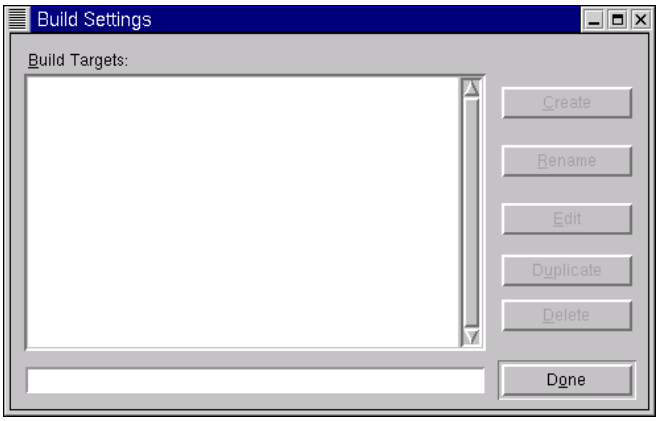
A build target is a conceptual object that contains information needed to compile and link a project. For example, **hello.c** is converted into **hello.o** before **hello** is built. To produce the **hello** executable, the **hello.o** object file is linked with any required libraries.

The first time you select Build Settings, the Build Targets list is empty. After you create a build target, its name appears in the list.

Creating a New Build Target

From the Tools menu, select Build Settings to start the Build Settings dialog.

Build Settings Dialog



Enter the name of the build target in the text entry box and click the Create button. The Edit Target dialog opens.

See Editing a Target.

Modifying Build Targets

Rename a build target by selecting the target, entering a new name, and clicking the Rename button.

Note

Do not highlight the target name in the text box or a new target is created.

To edit an existing build target, select the target and click the Edit button.

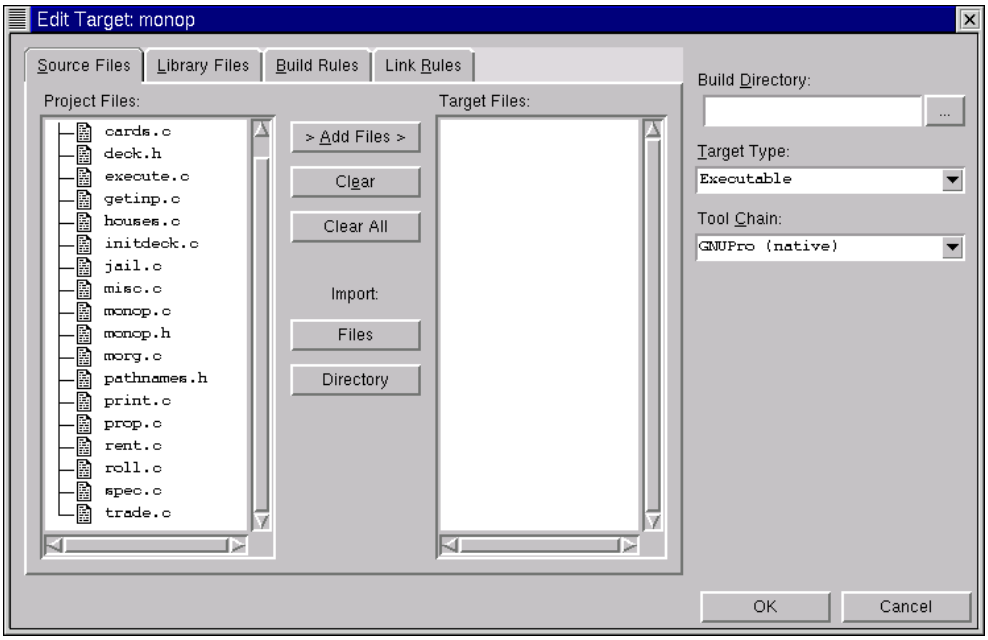
Duplicating a build target is useful when a new target is only slightly different from an existing target. To duplicate an existing build target, select the target in the Build Targets list and click the Duplicate button.

Delete a build target by selecting it in the Build Targets list and click the Delete button. The target is removed from the listing.

Editing a Target

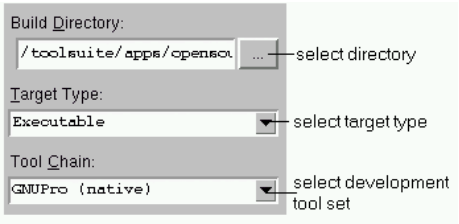
In the Build Targets list, either select the target name and click the Edit button or double−click the target name. The Edit Target dialog opens.

Edit Target Dialog



Edit Target tabs

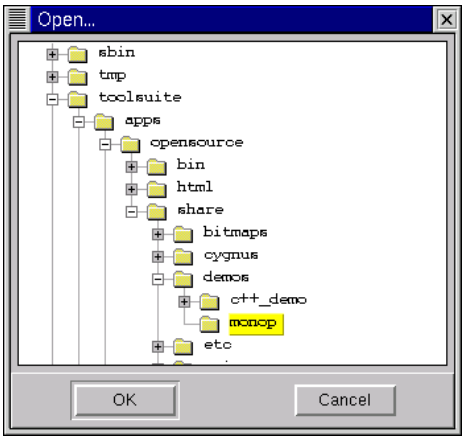
The Build Directory, Target Type, and Tool Chain combo−boxes are common to the tabs accessed in the Edit Target dialog.



These store information about the build target.

Build Directory

This is the directory where all files generated in the build process are initially stored. By default this is blank. If this is left blank, then the project directory is used. To specify the directory, either type the directory path or click the "..." button. If you click the "..." button, the Open dialog opens:



Select the directory in which to store the intermediate files. Click OK to close the dialog. The directory appears in the Build Directory field.

Target Type

Use this combo−box to select the type of build target to create. The options are *Executable* and *Library*.

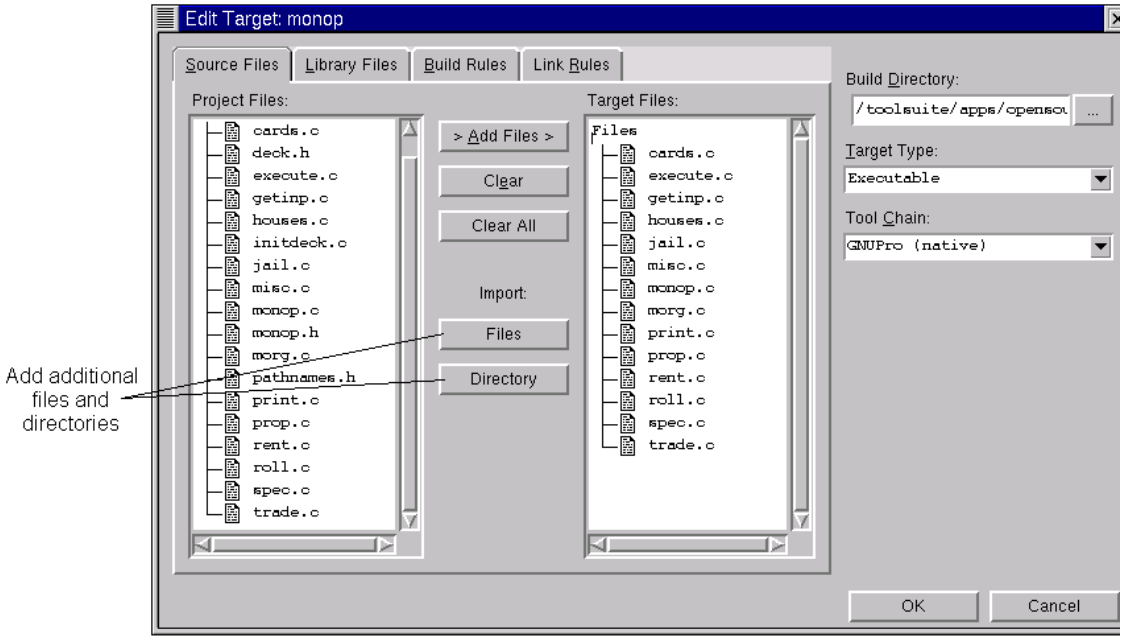
Tool Chain

A *tool chain* is a set of compilers, debuggers, and linkers. The currently available option is *GNUPro (native)*.

Source Files tab

The Source Files tab controls which source files are included in your build target. The Project Files and Target Files sections contain tree information for the selected build target.

Source Files Tab



Adding files

To add files to the Target Files list:

1.

Select the files from the Project Files list.

2. Click the Add Files button.

   The files are copied into the Target Files list.

Removing files

To remove files from the Target Files section, select the files to remove and click the Clear button. To remove all of the files from the Target Files section, click the Clear All button.

Importing files and directories

To import files or directories into your currently active project:

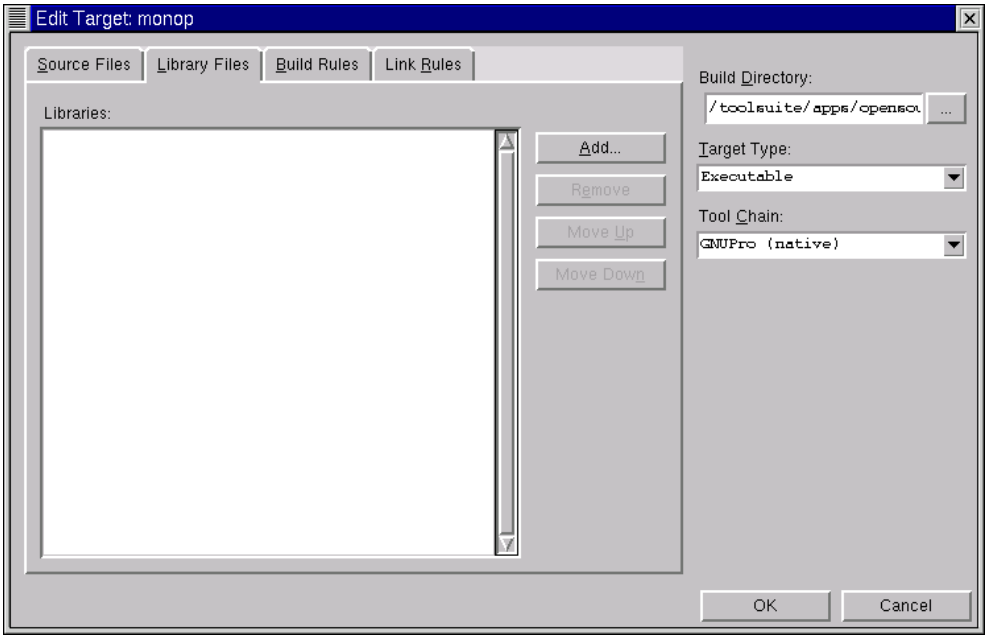
1. Under Import, click either the Files or Directory button.

2. Click the files or directories you wish to add to the project. Click OK when done. The names of the added files or directory appear in the Project Files list.

Library Files tab

Most libraries required for building targets are linked in automatically by the compiler/linker. If you know that your target requires additional libraries, use the Library Files tab to add them to your build.

Library Files Tab



Add additional libraries by clicking the Add button. The Open dialog opens to the last directory you have looked at in this project. After you select a library and click the Open button, it appears in the library list.

To remove a library from the listing, select the library and click the Remove button. The library is removed from the build.

Libraries are linked in the order listed in the dialog. To change the order of libraries, select the library and click

either the Move Up or Move Down button to change its linking order.

Build Rules tab

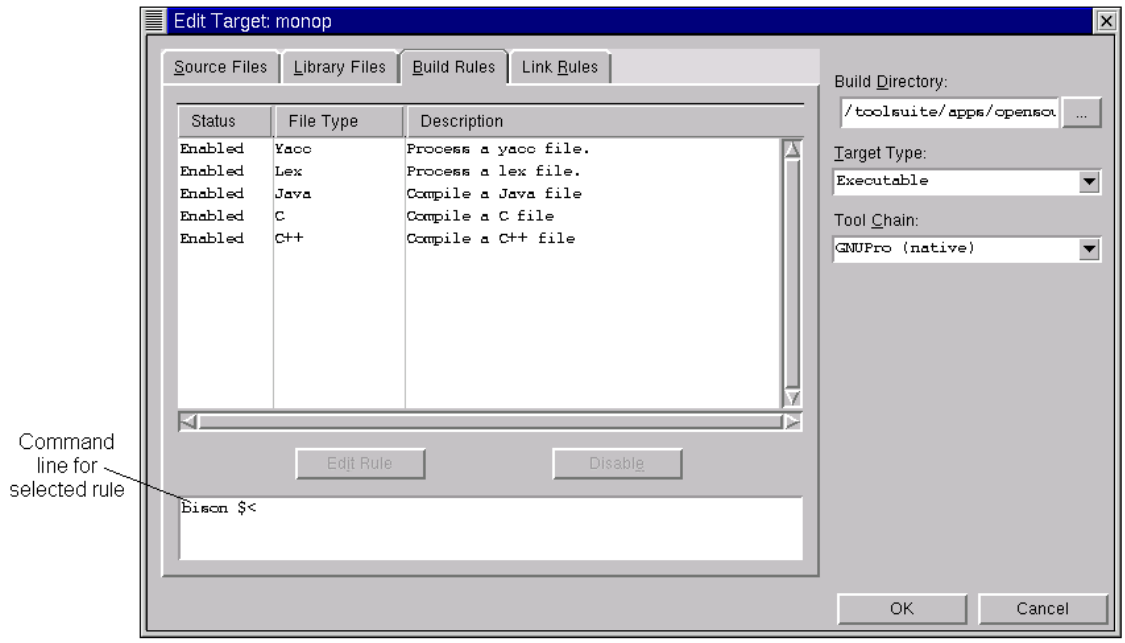
Click the Build Rules tab to configure each rule in the build target.

A rule contains information required to compile files in a project. For example, a rule to compile a C file might contain information about which compiler and flags to use, as well as what the file is called after it's compiled.

This tab lists the rules for the specified build target. Within the Build Rules tab, you can:

- disable and enable rules for the target.
- edit existing rules.

Build Rules Tab



Status

This column shows the currently enabled and disabled rules. To disable a rule, highlight the rule and click the Disable button. To enable a rule, highlight the rule and click the Enable button.

Note

This button changes between Disable and Enable depending upon the state of the rule.

File Type

This column displays the type of file the rule acts upon.

Description

This column displays a description of the rule.

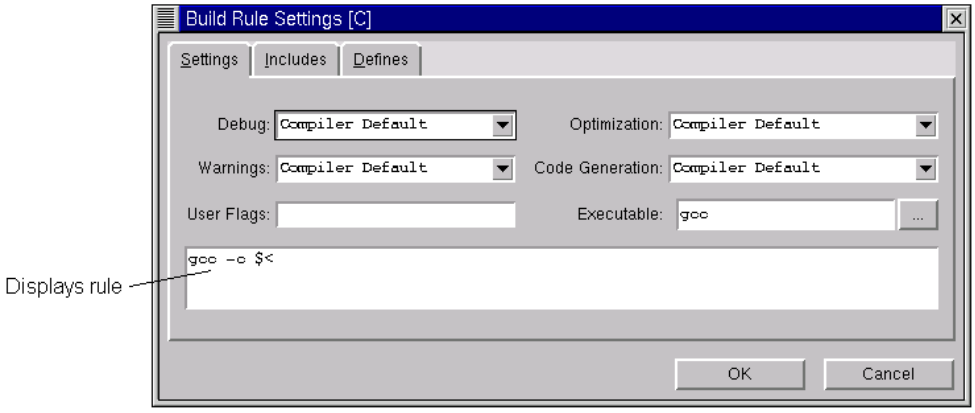
Editing a rule

To edit a rule, select the rule from the rule listing. Click the Edit Rule button. The Build Rule Settings dialog opens. The dialog title bar displays the extension for the files involved.

Settings tab

The Settings tab allows you to change the default settings for the rule.

Build Rule Settings with Options Selected



Debug:

This controls the debug information generated by the compiler.

Warnings:

Controls the level of warnings the compiler generates. A stricter warning ensures fewer problems with future compatibility. Set "Warnings as Errors" to make sure the compile stops any time a warning is generated.

User flags:

Enter flags not covered by the options listed in this screen. To add macros, see Defines tab.

Optimization:

Compiler optimization for the code.

Code Generation:

Processor or code control specific optimizations of settings.

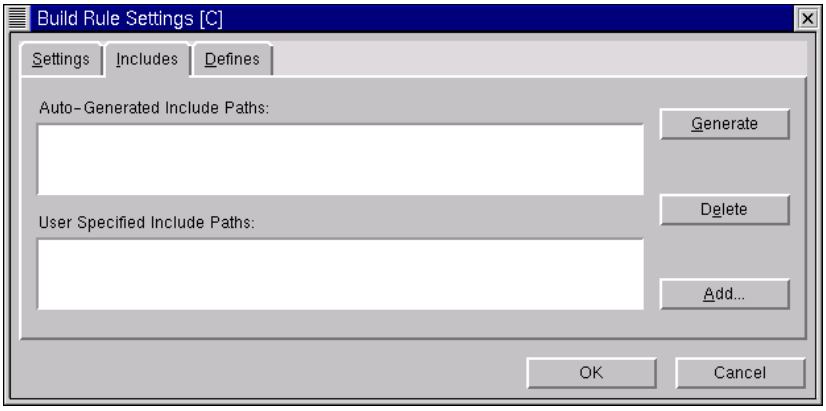
Executable:

Selects the executable to use. To change the tool (such as compiler) location, either enter the location path or click the "..." button to choose the tool binary to use.

Includes tab

Sometimes files include other files. In C this is done with the **`#include`** statement. The Includes tab allows you to change the included paths for the rule.

Includes Tab



The Auto−Generated Include Paths list displays paths generated from the Source−Navigator database.

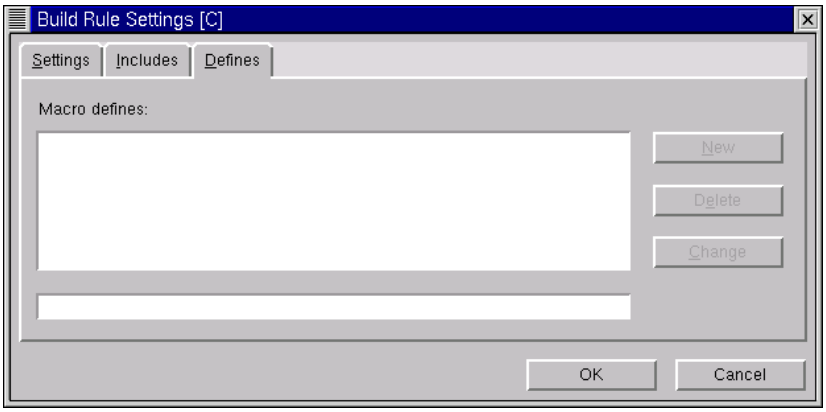
Click the Generate button to generate a list of Source−Navigator included paths. These appear in the Auto−Generated Include Paths list.

To add additional paths, click the Add button. The selected paths appear in the User Specified Include Paths list. You can delete paths by highlighting the path and clicking the Delete button.

Defines tab

The Defines tab enables you to view, edit, and create new macro definitions.

Defines Tab

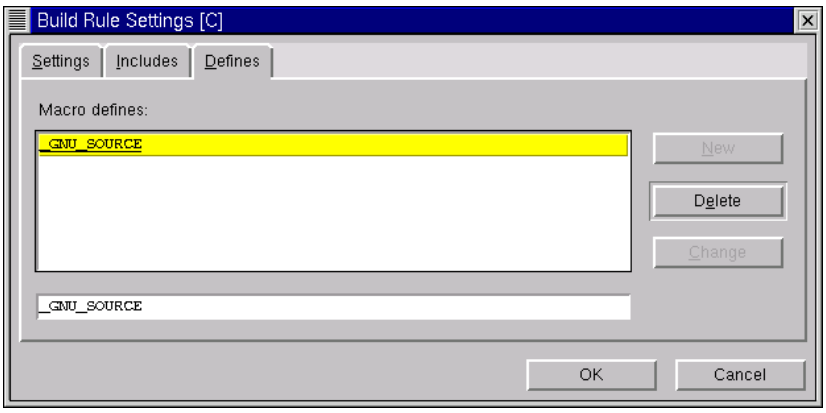


Modifying macro definitions

To create a new macro, enter the name and definition in the text entry box and click the New button.

To change the current macro definition, select the macro from the Macro defines list. The macro appears in the text entry box.

Macro Created and Selected



Make the necessary modifications to the macro.

To create a new macro, click the New button. The new macro appears in the listing.

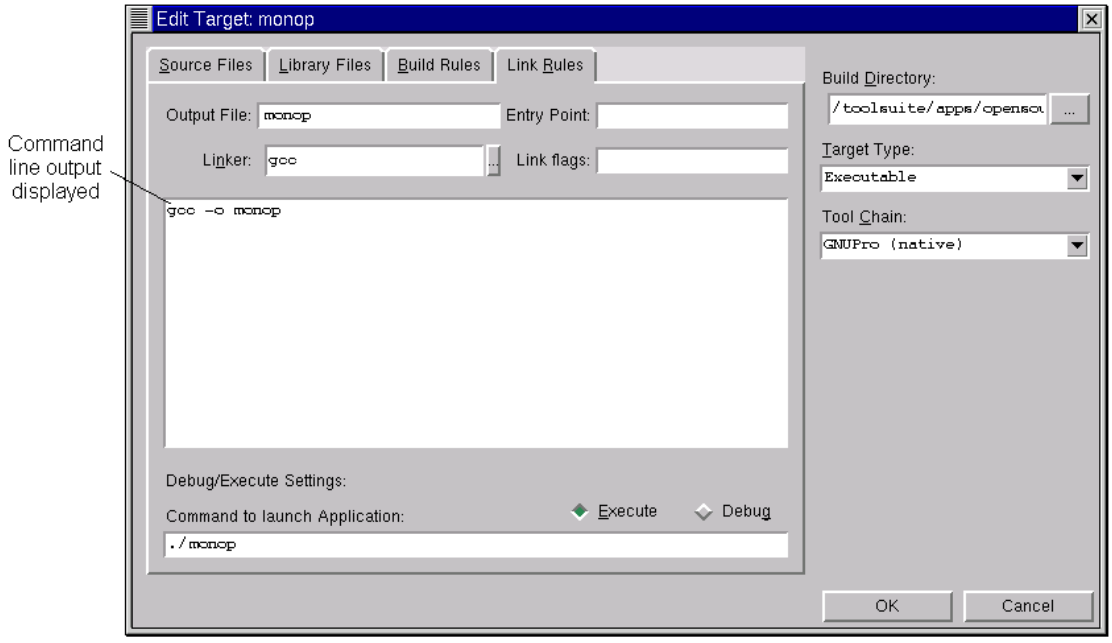
To update the macro, click the Change button. The modified macro appears in the listing.

To delete a macro, select it from the Macro defines list and click the Delete button. The macro is removed from the listing.

Link Rules tab

The Link Rules tab allows you to specify the program to execute and the name of the final output file.

Link Rules Tab of the Build Targets Menu



Output File:

The name of the final output file.

Linker:

Auto−detects the type of project. Click the "..." button to select another linker.

Entry Point:

This is the first function executed for the application. Default is **`main()`**. In Java, you must specify the name of the class that defines **`main()`**.

Link flags:

Displays the full link command line and allows you to add flags, but you cannot edit existing links or flags.

Debug/Execute Settings:

Controls the mode for the rule. Execute allows running the program from the Build window. Debug allows starting the Insight debugger from the Build window. See Debugging the build target for more information on debugging your program.

Command to launch Application:

This field lists the name of the binary to execute or debug. The default name is the file listed in Output File. If you changed the default output file, you also need to change the name here.

On UNIX, enter **`xterm -e`** before the executable name if you are debugging a console application.

Click the OK button to close the Edit Target dialog. Click the Done button to close the Build Settings dialog.

Compiling Build Targets

After you have created and configured a build target, you must build it.

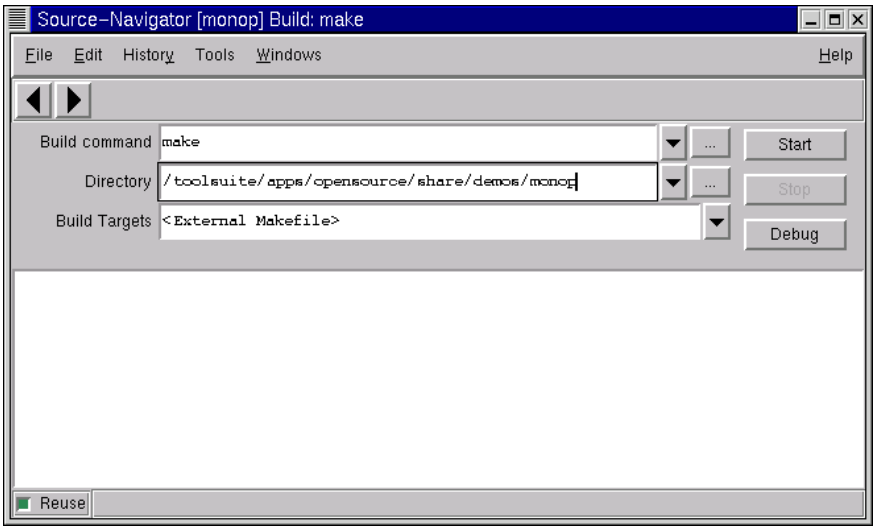
Source−Navigator can generate its own **`makefiles`** or work with one that you supply.

Internal build systems

To build your project using the Source−Navigator build system:

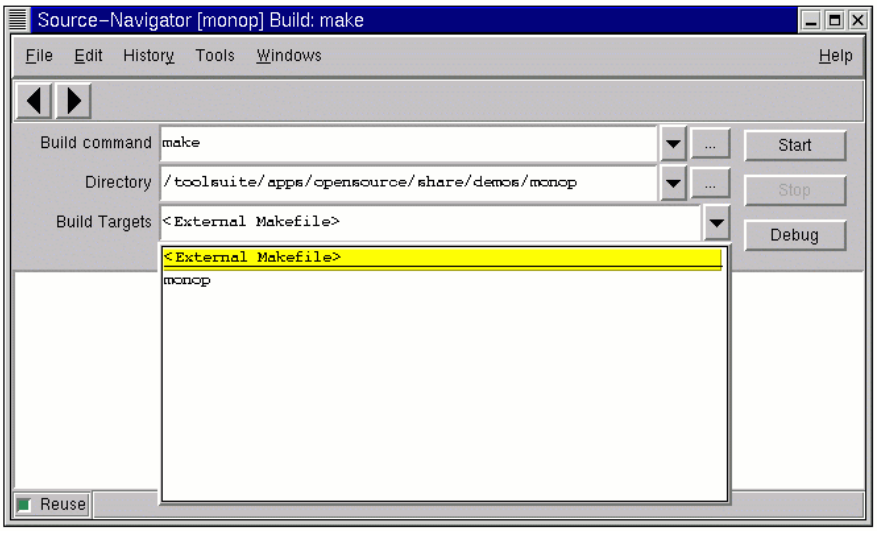
1. From the Tools menu in the Symbol Browser, select Build.

2. The Build window opens.



3.

Select the build target from the Build Target list.



4. Click the Start button to perform the build. If you need to stop the build process for any reason, click the Stop button.

5. When Source−Navigator starts the **`make`** command, output from the **`make`** process is displayed. To step through the results, click the left or right black arrow keys in the toolbar.

6. If errors appear, the build target needs to be modified. See Modifying the Build.

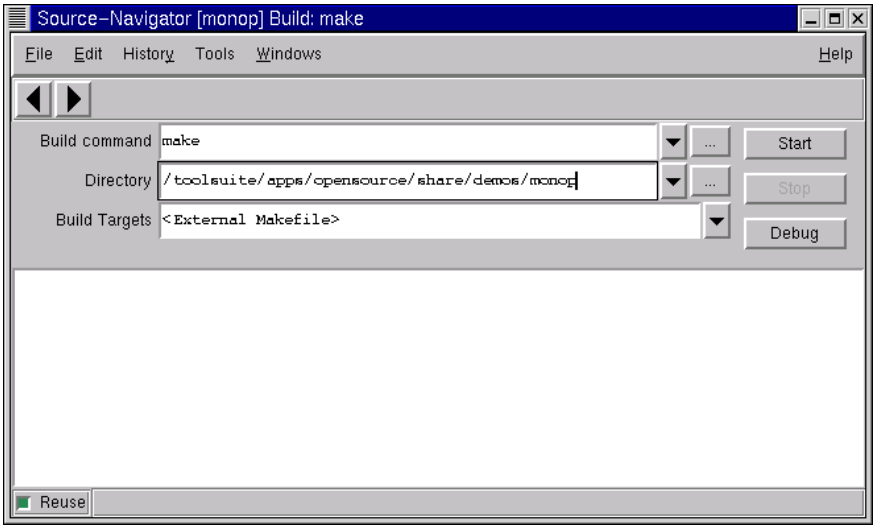
   If no errors appear, the build target is compiled and ready to be executed. Click the Run button to execute the application.

External build systems

To build using your own **`makefile`**:

1. From the Tools menu in the Symbol Browser, select Build.

2. Ensure that Build Targets is set to **`<External Makefile>`**.



3. In the Directory field, select the directory containing the external makefile.

   If you require additional flags or a different **make** program, from the File menu, select Project Preferences and select the Others tab. Enter the additional flags or **make** program into the Build field (see Others Tab of the Preferences Dialog).

4. Click the Start button to perform the build. If you need to stop the build process for any reason, click the Stop button.

5. When Source−Navigator starts the **make** command, output from the **make** process is displayed. To step through the results, click the left or right black arrow keys in the toolbar.

6. If errors appear, the source code needs to be modified.

   If no errors appear, the build target is compiled and ready to be executed. Click the Run button to execute the application.

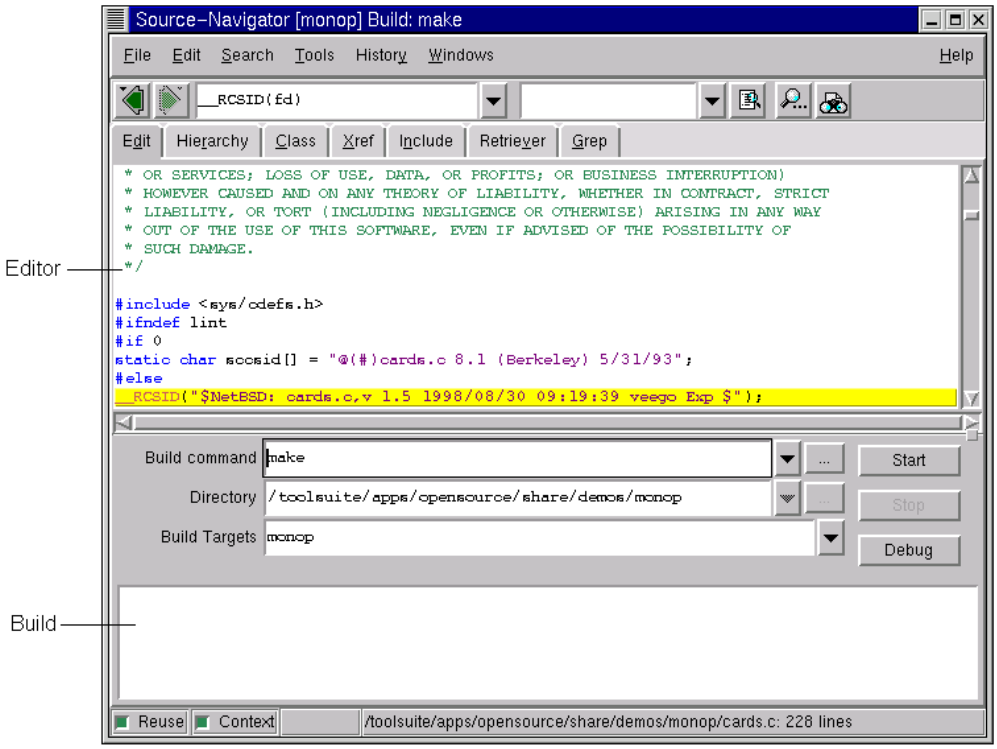
Modifying the Build

If errors appear in the Build window, the source code must be modified before the build is downloaded to the target board.

Double−clicking a line with a compiler error message activates the Editor with the cursor positioned on the line of code where the error appears. Right−clicking in the output window allows you to save the build output to a file.

From the Windows menu, select Add View −> Build to add a Build window to an Editor window.

Editor−Build Window



To save and recompile the modified files, press the Start button. The Fast Save dialog box appears asking if you want to perform a fast save. Click the Yes button to save the changes and rebuild. Click the No button to discard changes and perform a rebuild. Clicking the Cancel button closes the dialog without rebuilding the program.

When the build is completed, Source−Navigator saves the executable using the name specified in the Link Rules tab. If a name is not specified, the file name will be the target name. The compiled file is saved in the build directory.

Build types

The Tools menu in the Build window lists the types of builds you can perform. Before selecting an option from this menu, ensure that a build target has been selected.

Tools Menu

Build

Rebuilds only the modified files. This is the same as clicking the Start button in the Build Targets window.

Force Build

Rebuilds the modified files in the project and ignores errors,

Clean Build

Removes the object files and executables.

Export Makefile

Saves your build target as a **`makefile`** for external use.

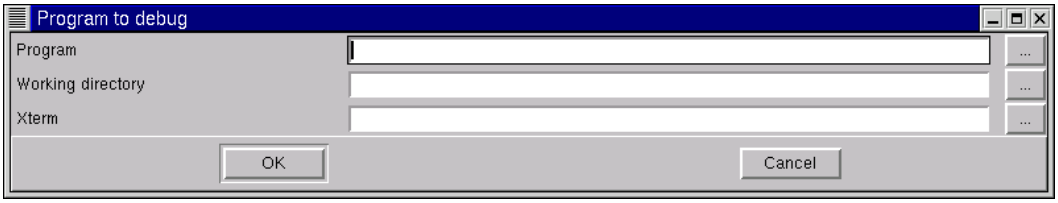
Debugging the build target

If you selected Debug from the Link Rules tab in the Edit Target dialog, you will use Insight or another debugger to debug your code.

1. From the Build window, click the Debug button.

   The Program to Debug dialog appears.

   Program to Debug Dialog Box

2. In the Program field, enter or select the application to debug.

   In the Working Directory field, enter or select the path of the project directory.

   On UNIX, in the Xterm field, enter **`xterm`** to have a separate debugger output console window appear. If this field is left blank, the Source−Navigator output console, specified in the Others preference tab (see Others tab), is used.

3. Click OK to debug the build target.

   The debugger launches.

   For more information on the debugger, see Debugger or your debugger's documentation.

4. Return to Source−Navigator to fix any errors and repeat the build−download−debug cycle.

Changing the build target from Debug to Execute

After debugging and fixing your build target, you must change the build target from debug to execute in order to run the application.

1. From the Symbol Browser window, select Tools −> Build Settings.

2. Double−click the build target to modify from the Build Targets list box.

3. Click the Link Rules tab.

4. In the Debug/Execute Settings, select Execute.

5.

Click the OK button to close the Edit Target dialog.

6. Click the Done button to close the Build Settings dialog.

7. Recompile and execute your build target (see Compiling Build Targets).

Executing the application

Once the application compiles without errors, from the Build window, click the Run button to execute the application.

Build Tutorial

This build tutorial is based on the **monop** demo, located in the **demos** directory. **monop** is a command line based game that you can build, edit, and play. In this tutorial, you will create two build targets and use many build and debugging features of Source−Navigator.

Note

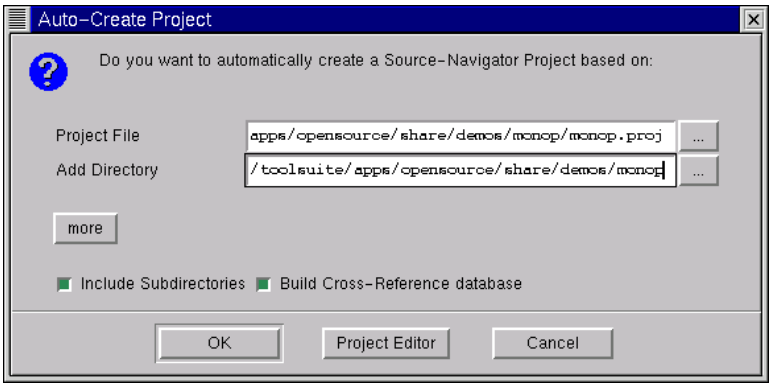
The monop demo is written in C. A Java demo is located in the **.../share/demos/java** folder. A C++ demo is located in the **.../share/demos/c++** folder. To learn more about these demos, read the Readme file located in each folder.

Creating the Project

1. In the Symbol Browser, from the File menu, select New Project.

2. Create a new project called **monop.proj**.

3. Under Add Directory, click the "..." button to select the **demos/monop** folder.



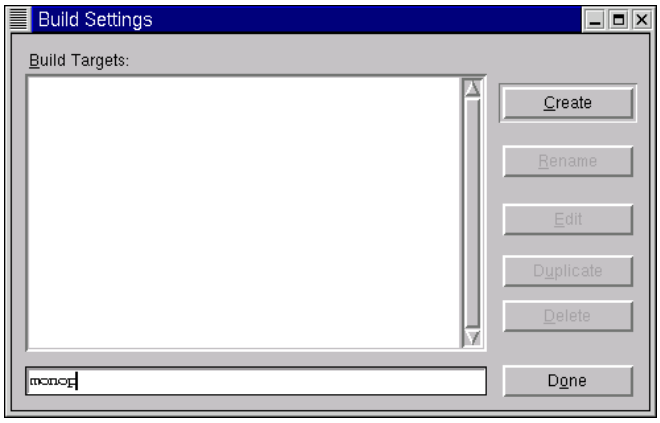
4. Click OK to create the project

Creating the monop Target

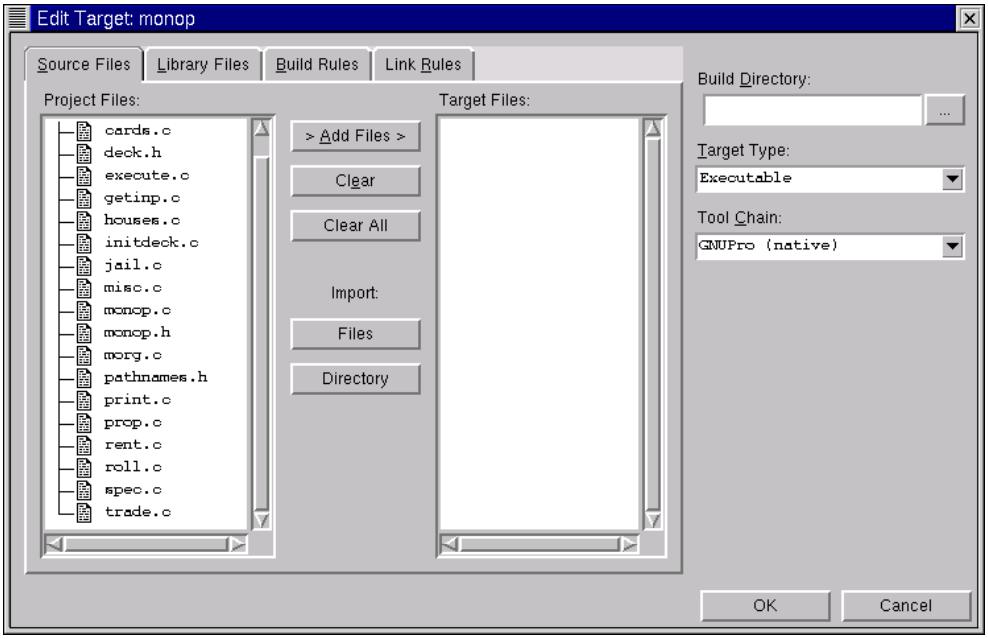
1. From the Tools menu, select Build Settings. The Build Settings dialog opens.

2.

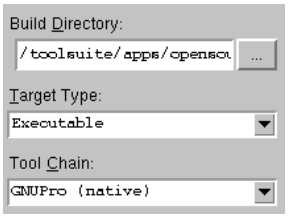
Enter **monop** as the name of the build target.



3. Click the Create button. The Edit Target dialog opens.

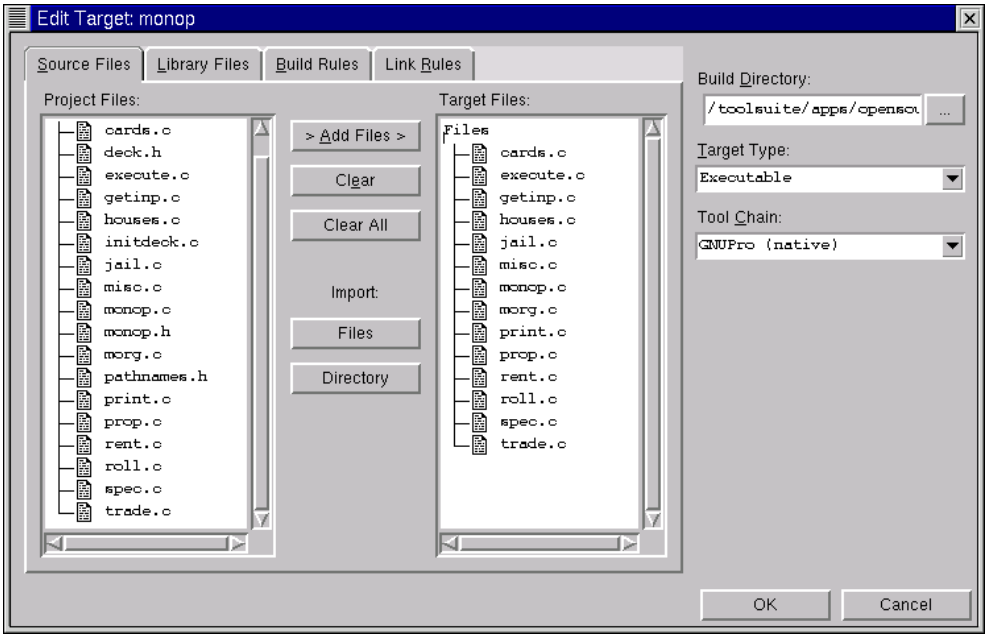


4. In the Build Directory field, click the "..." button and select the build directory for the **monop** project.



5. From Project Files, select the **cards.c**, **execute.c**, **getinp.c**, **houses.c**, **jail.c**, **misc.c**, **monop.c**, **morg.c**, **print.c**, **prop.c**, **rent.c**, **roll.c**, **spec.c**, and **trade.c** files.

6. Click the Add Files button to copy the files to the Target Files list.



Note

To execute the program correctly in UNIX, click the Link Rules tab. Enter **`xterm -e ./monop`** in the Command to launch Application field. Click OK to close the Link Rules dialog.

7. Click OK to close the Edit Target dialog.

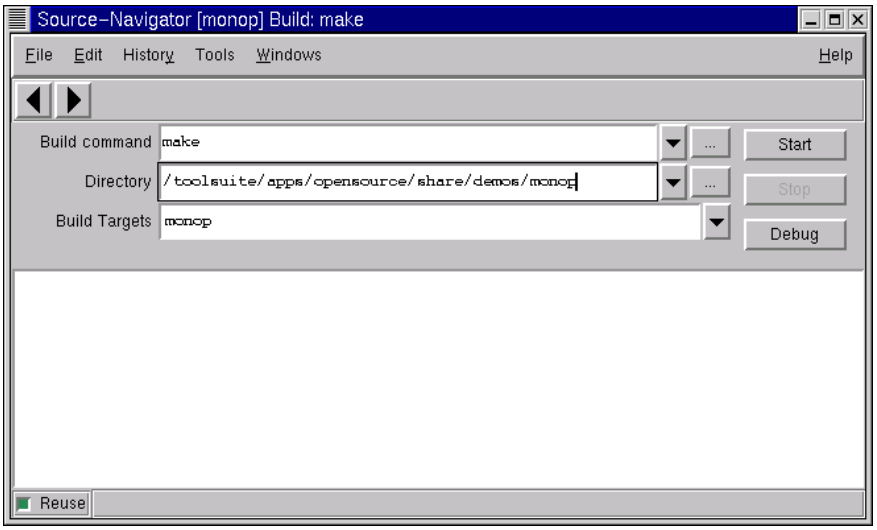
8. Click the Done button to close the Build Settings dialog.

The build target is created. Now you need to compile the program.

Debugging the monop build target

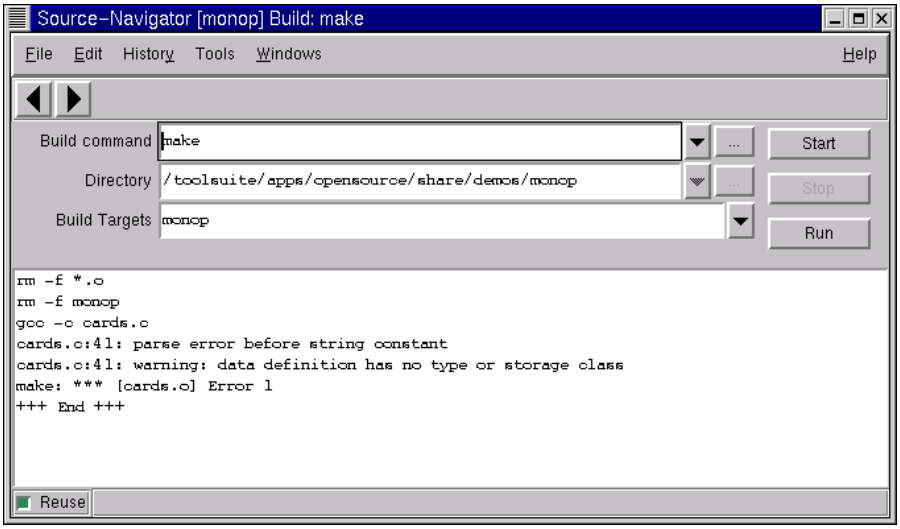
1. From the Tools menu, select Build. The Build dialog opens.

2. From the Build Targets field, select **`monop`**.



3.

Click the Start button.



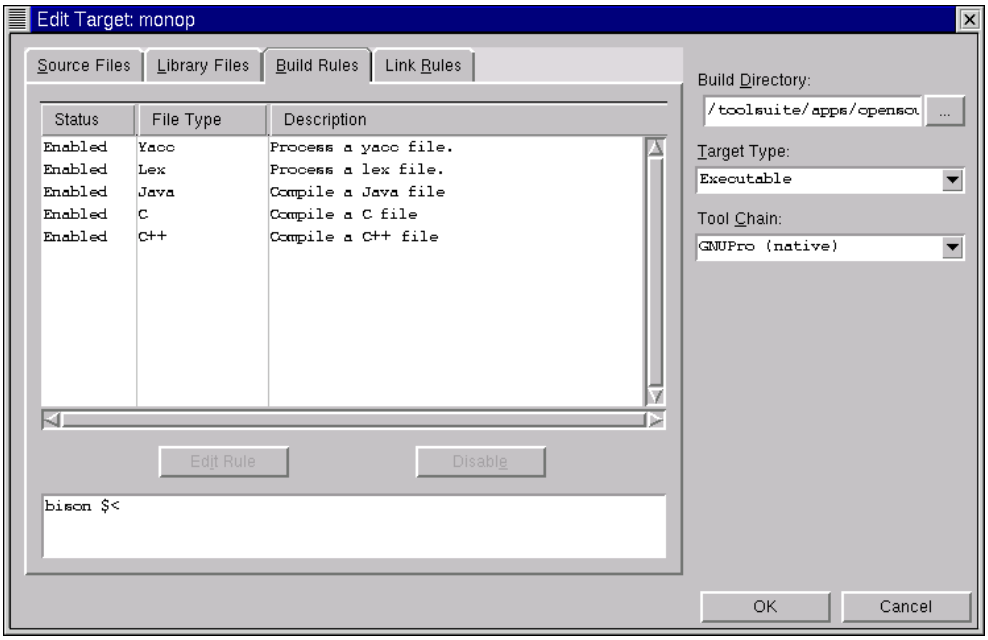
Source−Navigator generates some errors from the build. The **lint** macro must be defined.

Creating the lint macro

1. From the Tools menu, select Build Settings. The Build Settings dialog appears.

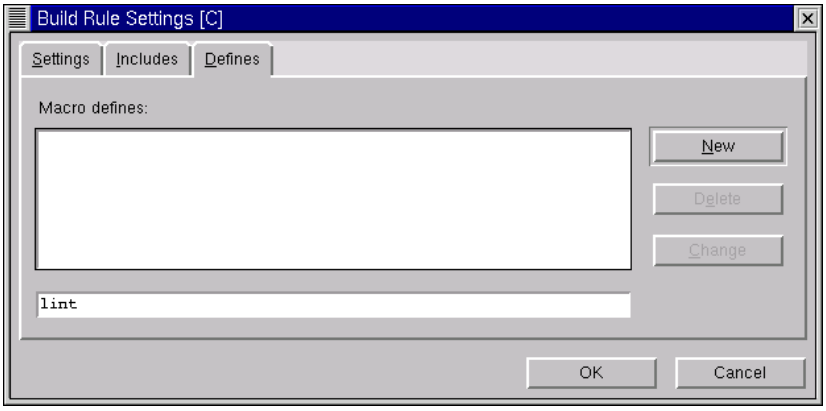
2. Double−click the **monop** build target. The Edit Target dialog opens.

3. Click the Build Rules tab.



4. Because **monop** is written in C, double−click the C rule. The Build Rule Settings dialog opens.

5. Click the Defines tab. Enter **lint** in the text entry box.



6. Click the New button to create the macro.

7. Click OK to close the Build Rules Settings dialog.

8. Click OK to close the Edit Target dialog.

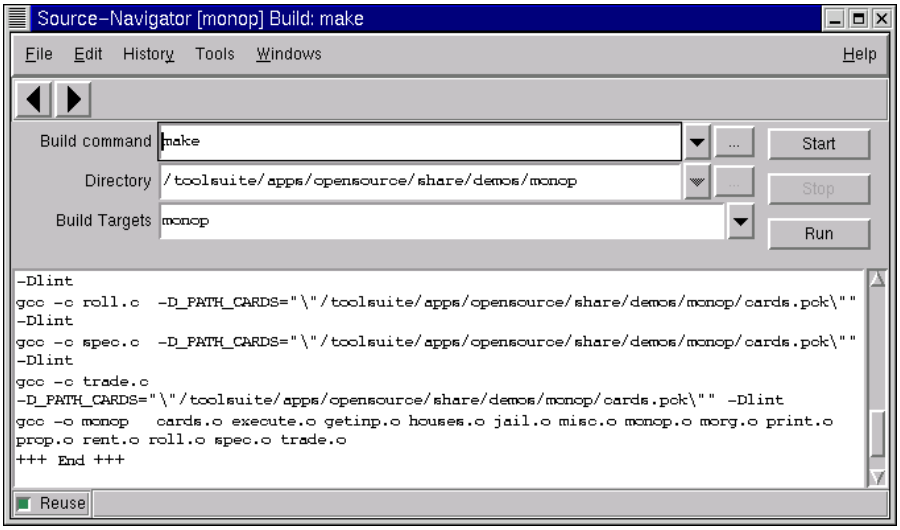
9. Click the Done button to close the Build Settings dialog.

Rebuilding the monop build target

1. From the Tools menu, select Build.

2. From the Build Targets field, select **monop**.

3. Click the Start button.



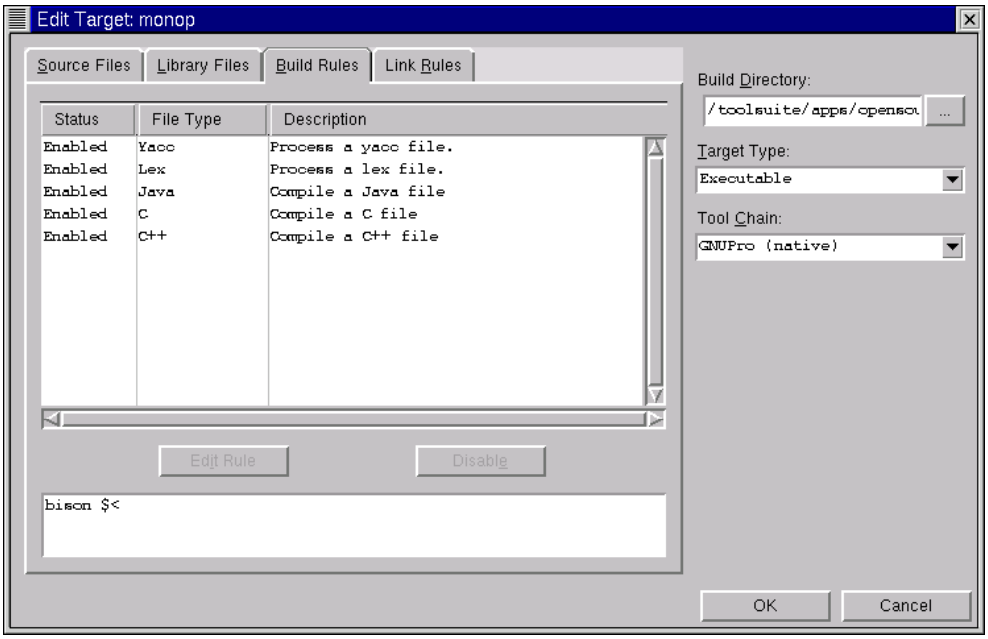
Source−Navigator generates without errors. However, at runtime the program will not execute because the path to the cards must be defined.

Creating the _PATH_CARDS macro

1. From the Tools menu, select Build Settings.

2. Double−click the **monop** build target. The Edit Target dialog opens.

3. Click the Build Rules tab.

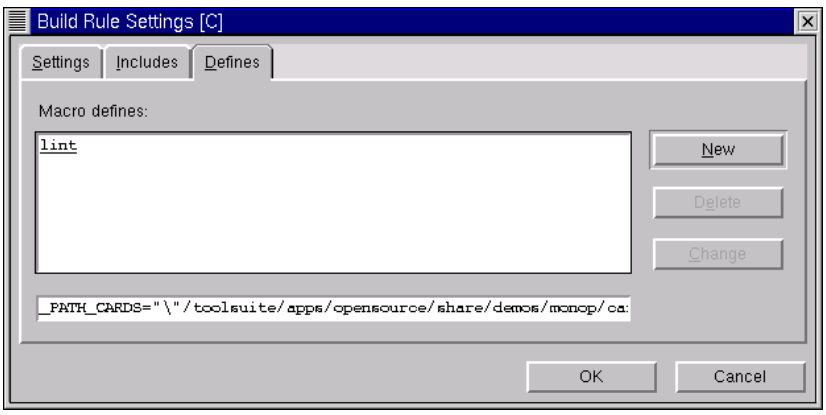


4. Double−click the C rule. The Build Rule Settings dialog opens.

5. Click the Defines tab.

6. Enter the following information in the text entry box, replacing **<project directory>** with the path to the **demos/monop** directory:

```
_PATH_CARDS="\" <project directory> /cards.pck\""
```



This tells Source−Navigator which card pack to use when running the **monop** program.

7. Click the New button to create the macro.

8. Click OK to close the Build Rules Settings dialog.

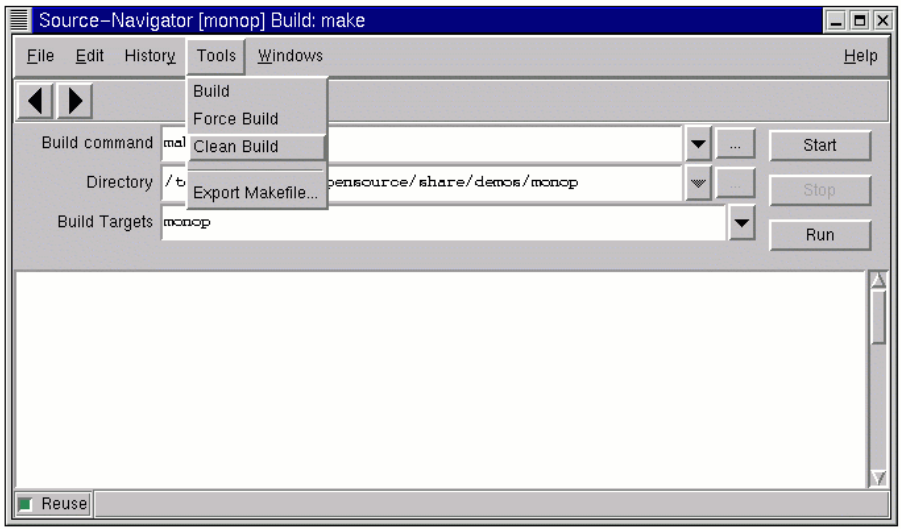
9. Click OK to close the Edit Target dialog.

10. Click the Done button to close the Build Settings dialog.

Performing a clean build

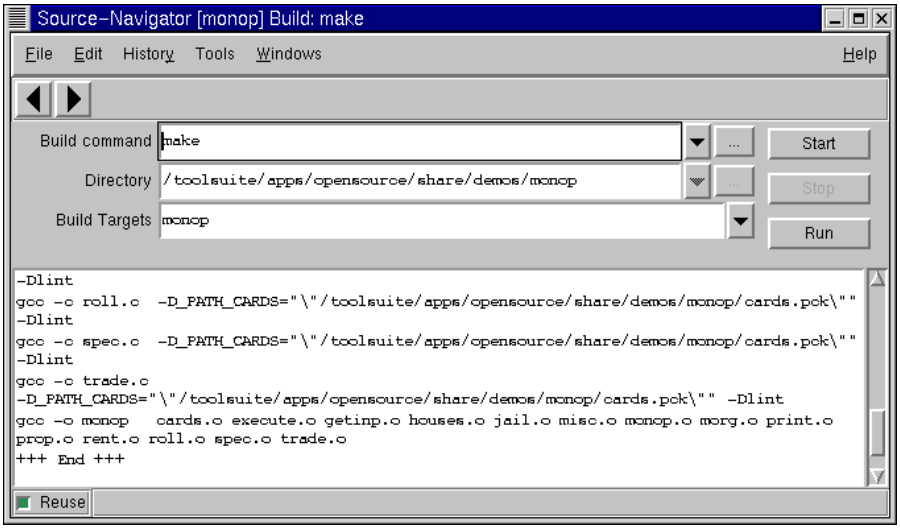
To ensure that the new macro is picked up at compile time, perform a clean build on the **monop** target.

1. In the Build window, from the Tools menu, select Clean Build.



   A clean build is equivalent to the command **make clean**.

2. Click the Start button to perform the build.



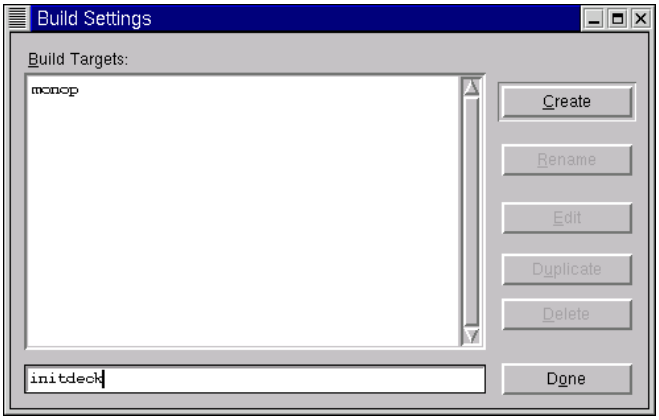
Again **monop** compiles without errors, but now it links to the correct card pack.

Now you need to create another target to initialize the cards used in the **monop** game.

Creating the initdeck Target

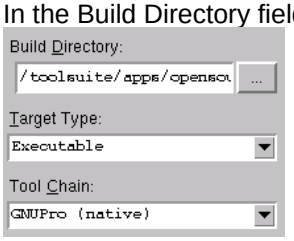
1. From the Tools menu, select Build Settings. The Build Settings dialog opens.

2. Enter **initdeck** as the name of the build target.



3. Click the Create button. The Edit Target dialog opens.

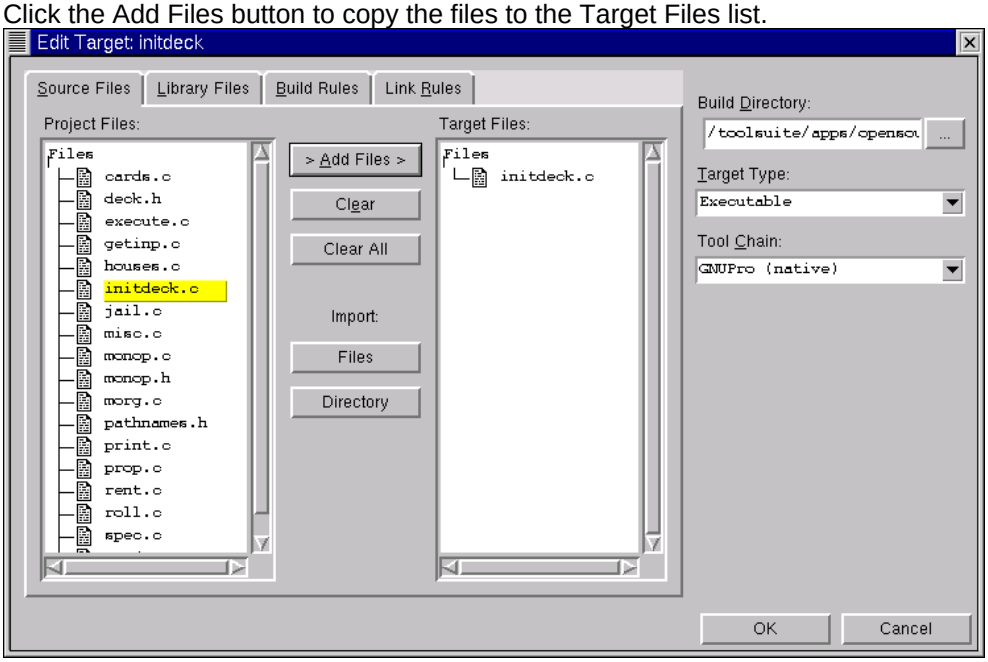
4. In the Build Directory field, click the "..." button and select the build directory for the **monop** project.



5. From Project Files, select the **initdeck.c** file.

6.

Click the Add Files button to copy the files to the Target Files list.



Note

To execute the program correctly in UNIX, click the Link Rules tab. Enter **`xterm -e ./monop`** in the Command to launch Application field. Click OK to close the Link Rules dialog.

7. Click OK to close the Edit Target dialog.

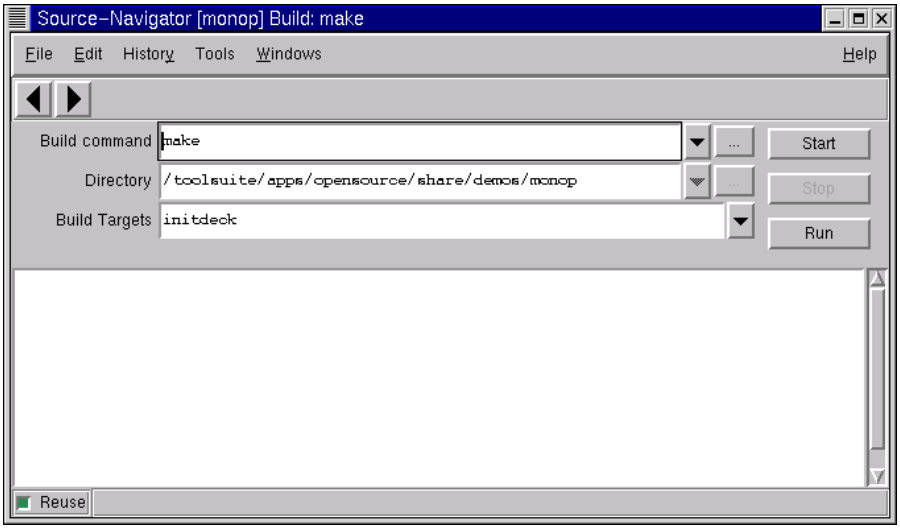
8. Click the Done button to close the Build Settings dialog.

The build target is created. Now you need to compile the program.

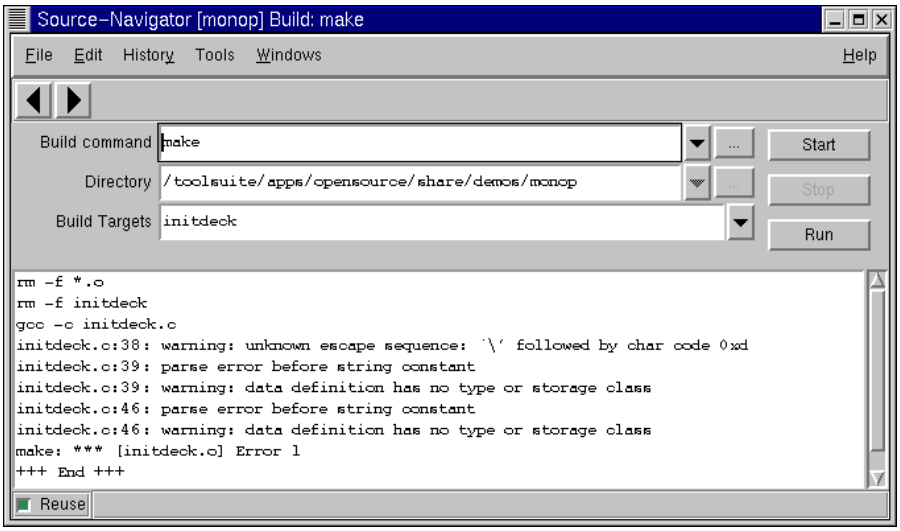
Debugging the initdeck build target

1. From the Tools menu, select Build. The Build dialog opens.

2. From the Build Targets field, select **`initdeck`**.



3. Click the Start button.



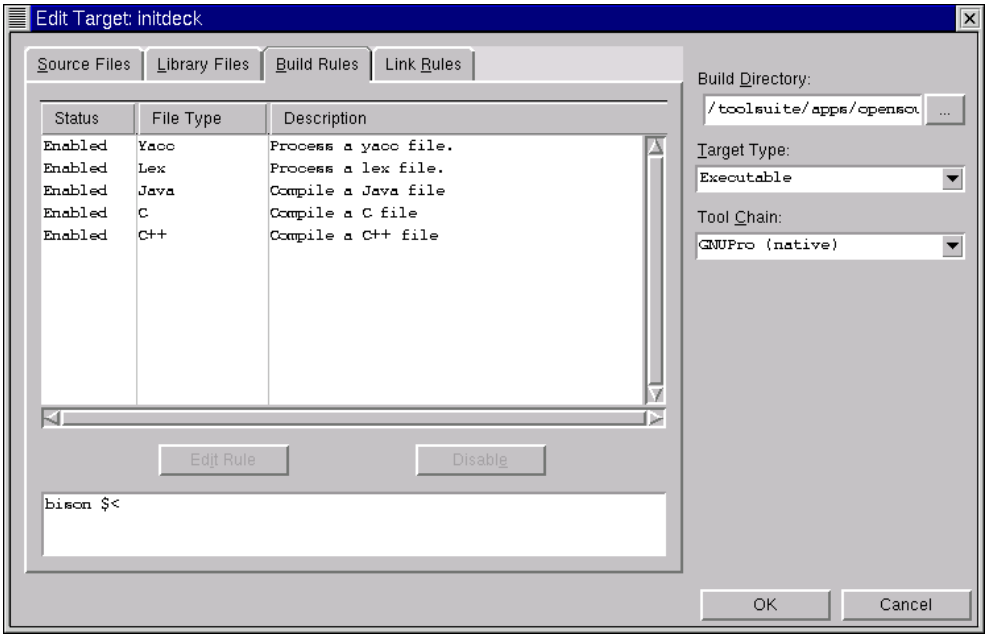
Source−Navigator generates some errors from the build. The **`lint`** and **`_PATH_CARDS`** macros must be defined.

Creating the lint and _PATH_CARDS macros

1. From the Tools menu, select Build Settings. The Build Settings dialog appears.

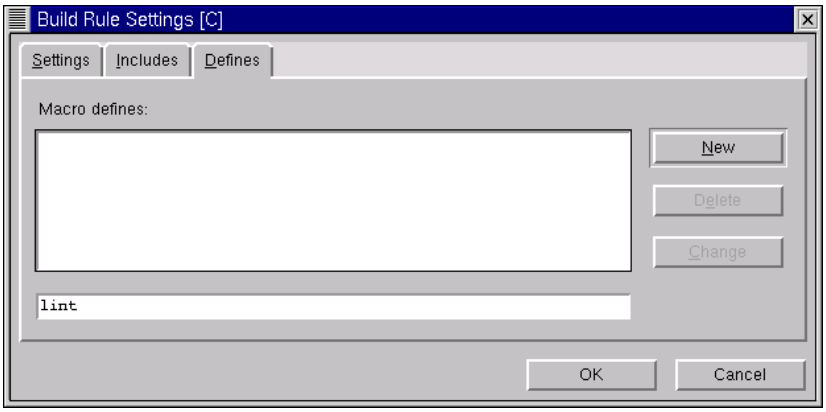
2. Double−click the **`initdeck`** build target. The Edit Target dialog opens.

3. Click the Build Rules tab.



4. Because **`initdeck`** is written in C, double−click the C rule. The Build Rule Settings dialog opens.

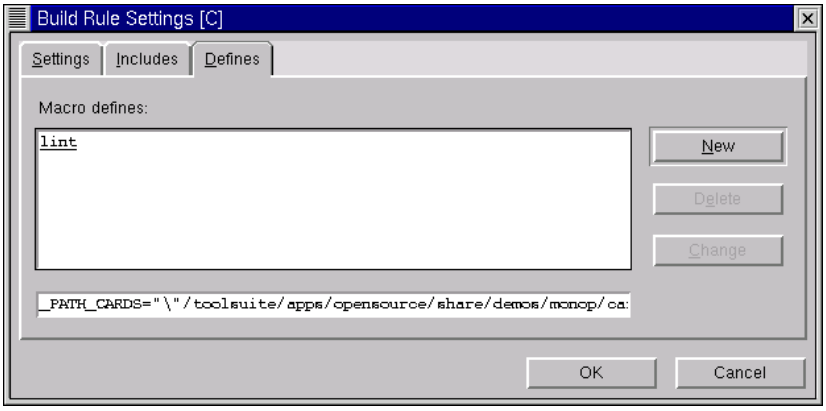
5. Click the Defines tab. Enter **`lint`** in the text entry box.



6. Click the **New** button to create the macro.

7. Enter the following information in the text entry box, replacing **`<project directory>`** with the path to the **`demos/monop`** directory:

```
_PATH_CARDS="\" <project directory> /cards.pck\""
```

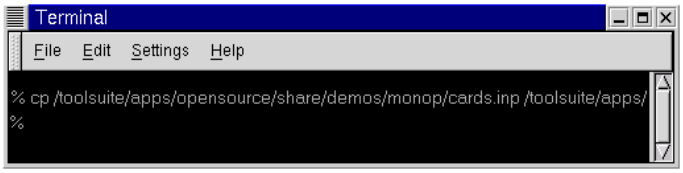


This tells Source−Navigator which card pack to use when running the **monop** program.

8. Click the New button to create the macro.

9. Click OK to close the Build Rules Settings dialog. Click OK to close the Edit Target dialog. Click the Done button to close the Build Settings dialog.

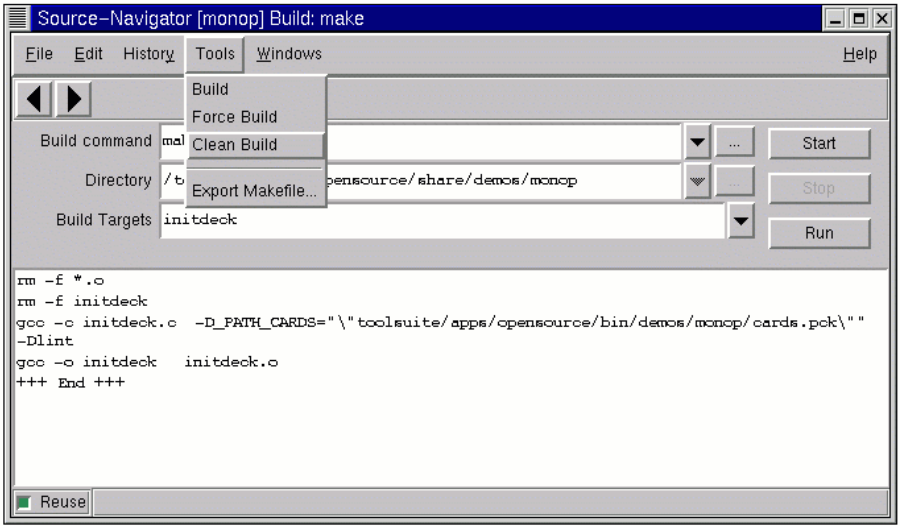
10. Open a console window and copy the **card.inp** files, located in the **demos/monop** directory, into the build directory.



Performing a clean build

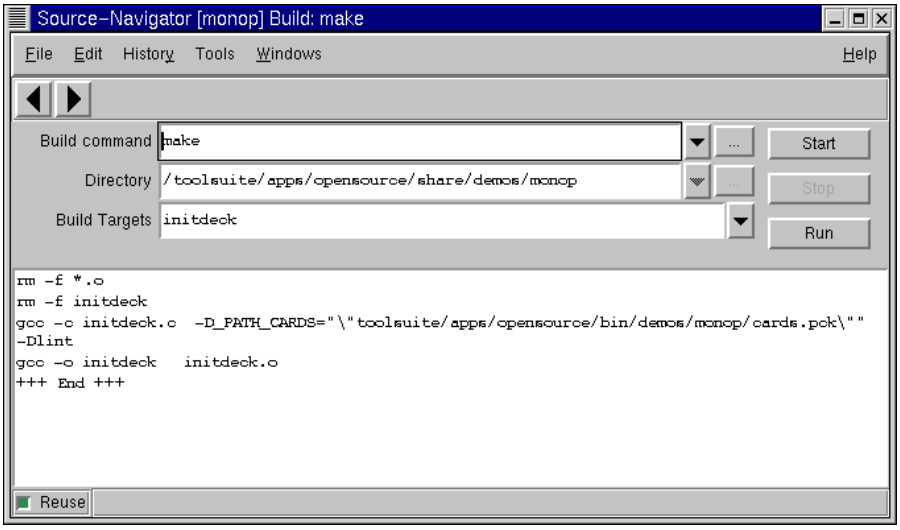
To ensure that the macros are picked up at compile time, perform a clean build on the **initdeck** target.

1. In the Build window, from the Tools menu, select Clean Build.



2.

Click the Start button to perform the build.



This time **initdeck** compiles and links without errors. **initdeck** is the name of the working executable.

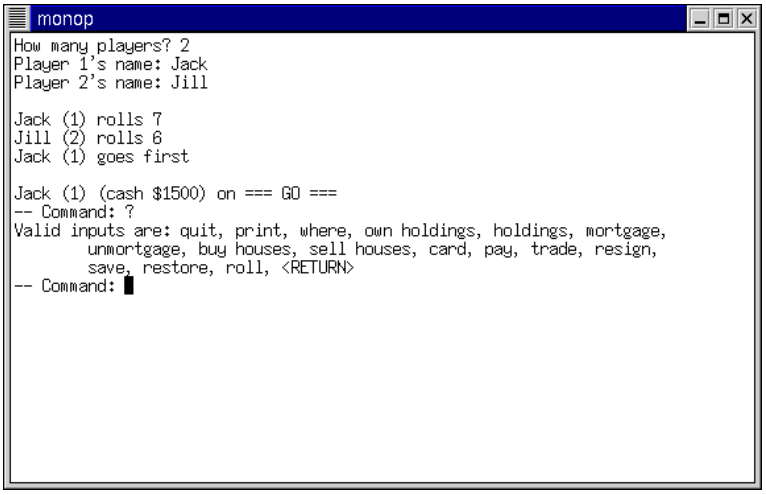
3. Click the Run button to run **initdeck**, which creates the cards used in the game.

   A console window opens to build the cards and closes after the build is complete.

4. From the Build Targets field, select **monop**.

5. Click the Run button.

   A console window opens and **monop** runs automatically. Enter the number of players and their names. Press the **?** key to list the playing options.



Command Line Options

Source−Navigator supports the following command line options:

−−batchmode

This forces batch mode to create a new project. If this is set, Source−Navigator will not launch. Instead the following command line options will be used to create a new project.

−−projectname

This command creates a project in the current directory using the directory path. For example, if the current directory is **/home/foo** and **−−projectname** is run, the project name becomes **/home/foo/foo.proj**.

−−avail−options

Sets preferences for an option used in project creation.

−−define <option>=<value>

Lists options that can be set using **−−define**.

−−databasedir <directory>

(synonyms: **−dbdir**, **−database**, **−db**)

Defines the directory for the symbol databases. Without this, the symbol databases are put in a directory called **.snprj** at the same level as the project file.

−−import <file>

Specifies a text file with a list of all files or directories to add to the project.

−−noxref

This option prevents the creation of cross−references. By default, Source−Navigator generates cross−reference information for the project.

−−create

This option is used to start the Auto−Create dialog. It is not normally used with **−−batchmode**. Source−Navigator prompts the user for information used to create a project.

Example: Creating a new project in devo−files

The following command creates a project named **/home/smith/devo.proj** using the files listed in devo−files in batch mode. The database files are stored in **~/db_files**. Source−Navigator returns when the project has been created.

```
~bin/snavigator −−batchmode \
  −−import devo−files \
  −−databasedir ~/db_files \
  −−project /home/smith/devo
```

Example: Creating a new project in the current directory

The following command creates a project in batch mode using the current directory. It adds all of the files in the current directory and in all of the subdirectories. The current working directory is **`/home/smith/devo/snavigator`**. The generated project name is **`/home/smith/devo/snavigator/snavigator.proj`**.

```
~bin/snavigator −−batchmode
```

Example: Auto−Create dialog

The following command displays the Auto−Create dialog, initialized with the current directory, with **`foo.proj`** as the project name. To view the project, you must open Source−Navigator.

```
~bin/snavigator −−projectname foo
```

Glossary

For more information about terms used in this document, see ***The C Programming Language*** [1] or ***The C++ Annotated Reference Manual*** [2] .

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

B

baseclass

A class from which other classes derive by inheritance. Compare *subclass*, *superclass*.

browser

A tool which presents information about source code in a specific way.

build

The process of configuring, compiling, and linking a set of tools. Also used as a noun, to denote the results of the process.

build process

The steps of Compiling and linking source files, such as libraries and executable files, to produce an output binary file. There are four steps to building a program: editing the code, compiling the source into object files, linking the object files together to produce an executable, and debugging the executable. This cycle is repeated as necessary.

build target

A conceptual object containing necessary information to compile and link a project.

Return to top

C

class

In object−oriented programming, a category of objects. The class defines all the common properties of the different objects belonging to it. See *baseclass*, *inheritance*, *subclass*, *superclass*.

class hierarchy

A graph or diagram of the relationship between classes. You can tell which classes derive from each other and what the class types are (baseclass, superclass, subclass).

class variable

In object−oriented programming, a variable used by the class definition. It is defined once and all instances of the class can access it. Contrast with *instance variable*. See *class*.

constants

A named item that retains a consistent value throughout the execution of a program. Compare to *variable*.

context

A point of reference in source code. Different browsers show this differently. Editor shows all code around this point, while the cross−reference browser shows the cross−reference information for the nearest identifier in the source code.

cross−reference

Shows where symbols are used in the source code. Selecting an item displays the refers−to and referred−by relationships; these relationships are based on the point−of−view of the selected symbol. See *refers−to relationship*, *referred−by relationship*.

Return to top

D

debugger

A program used to examine other programs while the other programs are running. A debugger allows a programmer to stop a program at any point and examine and change the values of variables.

declaration

The binding of an identifier to the information it relates to. Declaration usually occurs in a program's source code, while the actual binding occurs at either compile time or runtime.

Return to top

E

enumerated data type (enum)

A data type restricted to a sequence of named values given in a particular order.

enumerated values

The values named in an enumerated data type.

executable

A file that can be run. It consists of libraries and object files bound together by a linker. Compare *library*.

Return to top

F

files

The source code files, object files, and libraries in the project.

friends

In C++, a non−member function or class allowed access to a member function or class, that would otherwise be prohibited.

function

A self−contained unit of code. It has parameters and a return value. Also called a *subroutine* in FORTRAN. Compare *macro*. See *program*.

function declarations

Information given to the compiler to define the function parameters and return value of a particular function.

Return to top

G

global variable

A variable with a single value at a time that is in effect for the whole system. Contrast *local variable*. See *variable*.

Return to top

I

inheritance

In object−oriented programming, the ability of one class of objects to get properties from a higher class. See *class*, *multiple inheritance*.

implementation

The carrying out or physical realization of something. The phrase "there are various implementations of the protocol" means that there are several software products that execute that protocol. An information system implementation would be the installation of new databases and application programs and the adoption of new manual procedures.

included by relationship

In C or C++, a relationship where a selected file is included by another file. For instance, **`hello.h`** may include **`foo.h`** and **`bar.h`**. **`foo.h`** and **`bar.h`** are included by **`hello.h`**. Compare *includes relationship*.

includes relationship

A relationship where a selected file is included in another file. For instance, **`hello.h`** may include **`foo.h`** and **`bar.h`**. **`hello.h`** includes **`foo.h`** and **`bar.h`**. Compare *included by relationship*.

instance variable

In object−oriented programming, an instance variable is defined on a per object basis. Each instance of an object has its own copy of an instance variable. Contrast *class variable*. See *class*.

Return to top

L

library

A package of function calls, classes, and other code and data. A library cannot execute on its own. It can be included in multiple executables and can also be included in other libraries. It is composed of other libraries and object files. It is produced by linkers or special library tools, depending upon the operating system. See *shared library*, *static library*.

linker

A tool that merges object files to create libraries or a tool that merges object files and libraries to create programs.

local variable

A variable that is specific to a function or procedure. The value of a local variable disappears when the function returns. Contrast *global variable*. See *variable*.

Return to top

M

macro

A name that defines a set of instructions. These instructions are substituted at compile time for the macro name whenever the macro name appears in the source code. Compare *function*.

make

The utility for automating recompilation, linking, etc., of programs, taking account of the interdependencies of modules and their modification times. **`make`** reads instructions from a makefile, **`Makefile`**, which specifies a set of targets to build, a set of files on which the targets depend and the commands to execute in order to produce them. If **`make`** is run with no arguments, it looks for a makefile, **`Makefile`**.

makefile

A specific script that tells a UNIX program, **`make`**, how to build a particular computer program or set of programs. A **`makefile`** contains variable assignments and rules that which, with input, say if any files have been modified more recently than a target file (or if the target does not exist), then execution of specific commands will normally build the target from the inputs.

member function

Another term for method. See *method*.

metacharacter

A character embedded in a program that conveys information about other characters, rather then representing the character itself. An example in C programming is the backslash character, which, indicates that the letter following the backslash is part of an escape sequence that enables C to display an uprintable character, such as a tab or return.

method

A function attached to a class, also known as a member function. See *function*.

multiple inheritance

A relationship between objects where an object gets behavior with more than one other object. See *inheritance*.

Return to top

O

object

The principal building blocks of object−oriented programs. Each object is a programming unit consisting of data variables and functionality. See *class*.

object file

A binary−format file containing machine instructions and possibly symbolic relocation information.

Return to top

P

program

A piece of software that can be run. For instance, you create, compile and then link a software program file, **`foo.exe`**. Type **`foo`** at the shell window's prompt. Your operating system calls the defined entry point of the program, in C this is typically called **`main()`**; the program runs and, when it finishes, control goes back to the system.

project

An entity containing references to source code files. A project describes where files are located and how to operate on them.

project database

A representation of program structures, locations of function declarations, components of class declarations, and relationships between components. Also the file that contains this representation.

project directory

The directory where the project file is stored.

project file

A file that contains the database table describing all project−specific files, symbols, and references to other database files.

Return to top

R

refers−to relationship

A relationship where a selected symbol is used in the context of another symbol. For instance, **`hello`** may refer to the symbols **`foo`** and **`bar`**. **`hello`** refers to **`foo`** and **`bar`**. Compare *referred−by relationship*.

referred−by relationship

A relationship where a selected symbol is used by another symbol. For instance, **`hello`** may refer to the symbols **`foo`** and **`bar`**. **`foo`** and **`bar`** are referred by **`hello`**. Compare *refers−to relationship*.

Return to top

S

shared library

A library that exists once on a system that multiple executables can link to at runtime. Compare *static library*.

static library

A library that is linked into a program at link time. There is one copy of the library for each executable that links to it. All data is executable specific. Compare *shared library*.

subclass

A class that is derived from another class, known as a superclass. Compare *baseclass*, *superclass*.

superclass

A class from which other class, a subclass, is derived. Compare *baseclass*, *subclass*.

symbols

Used to refer to variables, labels, functions, methods, macros, procedures, or other programatic constructs in a program.

Return to top

T

target type

Target types include executables and libraries. Executables may include libraries and object files. Libraries may include other libraries and object files.

toolchain

The set of compilers, debuggers, and linkers specific to the type of machine compiling on and executing on.

type

The nature of a variable, such as an integer, real number, text character, or floating−point number. The type of a variable defines and restricts the values it can hold.

typedef

In C, a keyword used to name new data types.

Return to top

U

union

A structure that can be used to store different types of variables, such as integers, characters, or Boolean, in the same location, but only one at a time.

Return to top

V

variable

A variable is a name used in a program to stand for a value. See *class variable*, *global variable*, *instance variable*, *local variable*.

view

1) A window that contains a context. Selecting Add View creates a new window with a new context.

2) A set of files for a project. For more information, see Using the Project Editor.

Return to top

X

xref

The name of the cross−reference browser tab. See *cross−reference*.

Return to top

---

1. Kernighan, Brian, and Ritchie, Dennis. 1988. ***The C Programming Language***. ISBN: 0131103628 Return to top

2. Margaret A. Ellis, Bjarne Stroustrup. 1990. ***The C++ Annotated Reference Manual***. ISBN−0201514591. Return to top

---

Contents Previous Next
Contents Previous Next

GNU General Public License

This license contains the legal terms and conditions applying to Source−Navigator. See the **`Copying`** file in the source distribution for the complete copyright status and terms. For the Tcl/Tk tools, see the **`license.terms`** file in the **`src/tcl8.1`**, **`src/itcl`**, **`src/tix`**, and **`src/tk8.1`** directories of the source distributions.

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software−−to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions for Copying, Distribution, and Modification

O.

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. "Program" refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1.

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2.

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

`.

You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

a.

You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

b.

If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

- You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

c. Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

d. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

e. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

f. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

g. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

h. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties to this License.

i.

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

j.
If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

k.
The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

l.
If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11.
BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE

PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line for the program's name and a brief idea of what it does.

Copyright (C) 19**yy name of author**

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like the following example when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands **show w** and **show c** should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than **show w** and **show c**; they can be mouse clicks or menu items−−whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. The following is a sample (when copying, alter the names).

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

```
signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Contents Previous Next

Contents Previous

Index

Return to top

S

Return to top

T