Source-Navigator™ Reference Guide

GNUPro 2001

How to Contact Red Hat

Red Hat Corporate Headquarters
2600 Meridian Parkway
Durham, NC 27713 USA
Telephone (toll free): +1 888 REDHAT 1 (+1 888 733 4281)
Telephone (main line): +1 919 547 0012
Telephone (FAX line): +1 919 547 0024
Website: `http://www.redhat.com`

Contents

Introduction

As was described in the ***Getting Started Guide***, the GNUPro® Toolkit is made up of several components, including Source-Navigator™.

If you are familiar with the basics of Source-Navigator and are looking for additional details on using it to develop and manage your source code, see Part I: “User’s Guide” on page 5. If you want to do your development work from the command line, see “Command Line Options” on page 131.

In addition to the languages supported in this distribution, you can use the Source-Navigator Software Development Kit (SDK) to add new parsers for new programming language or customize the graphical user interface (GUI). For more information, see Part II: “Programmer’s Guide” on page 133.

About this Guide

This document serves as a reference to Source-Navigator GUI elements, tools, and functionality. Part I provides more detailed information than was included in the ***Getting Started Guide***, and discusses why you might choose one option over another when a choice is available to you. Part II provides information on customizing Source-Navigator, using the SDK.

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 when they appear in body text.

`Italic Typewriter Font`
: Represents a variable for which an actual value should be substituted.

`Bold Typewriter Font`
: Represents program output.

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. The term “click” alone always means left-click.
Double-click	Click the left mouse button twice in rapid succession without moving the mouse.

Right-click	Place the cursor on a specific object and click the right mouse button.
Select text	Click and drag the cursor through text (or code) to be selected. Selected text is highlighted.

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 underlined letter of the menu.

Part I:
User’s Guide

1

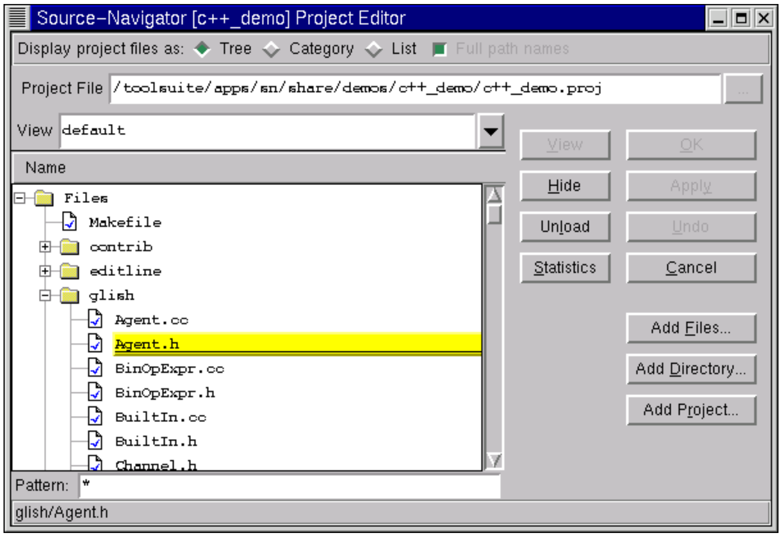
Using the Project Editor

You learned how to create basic Source-Navigator projects in the “Starting a New Project” section in the ***Getting Started Guide***. In this documentation, you will learn to use the **Project Editor** to fine-tune your projects.

Project Editor

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**.

Figure 1: Project Editor Window



Select an option from **Display project files as** to change the layout of the files and directories in the **Project Editor** window; **Tree** is the default layout.

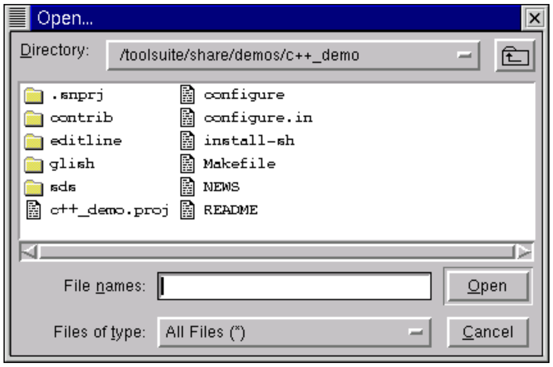
The **Project File** text box contains a file path; the directory that your project file is located in is your *project directory*. From now on, every non-absolute path will be relative to this directory. If you would like to choose a different path, click the “**...**” button. You can change the path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

Figure 2: Dialog for Adding Files to a Project

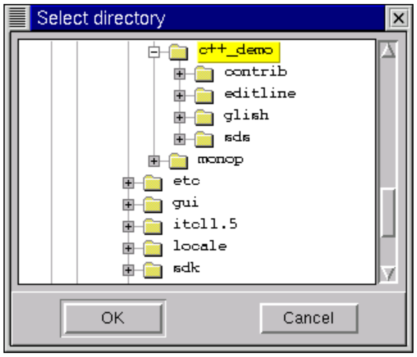


You can filter files based on the **Files of type** menu. **All files (*)** is chosen by default. Select the source files to be added to the project and click the **Open** button.

Adding Directories to a Project

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

Figure 3: Dialog for Adding Directories to a Project

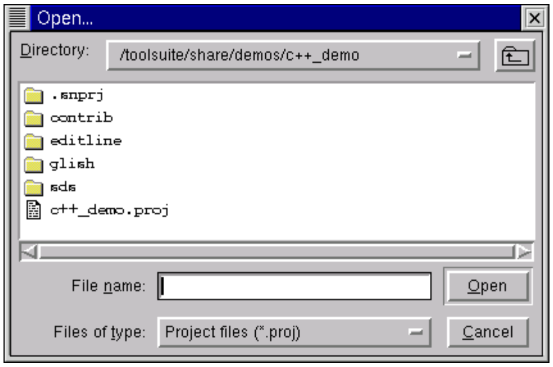


Select the directory you want to add and click **OK**. You can add as many directories as you wish, but you must add them one at a time.

Adding Another Project to a Project

The **Add Project** button allows you to import another Source-Navigator project and all of its files into your current project.

Figure 4: Dialog for Adding Another Project to a Project



This is the same **Open** dialog as when you clicked on the **Add Files** button, except that **Project files (*.proj)** is the default in the **Files of type** field. 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. Hiding files is convenient when there are many files in a view, but you are only interested in seeing a subset.

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 Files and Directories into a Project

As an alternative to using the **Project Editor** to select source code directories for the new project, you can collect file and 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 files or 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 *`filename`* must be a file or directory, 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" on page 131.

2

General Source-Navigator Features

This chapter builds on the basic information in the ***Getting Started Guide*** and provides details about the most commonly used features of Source-Navigator.

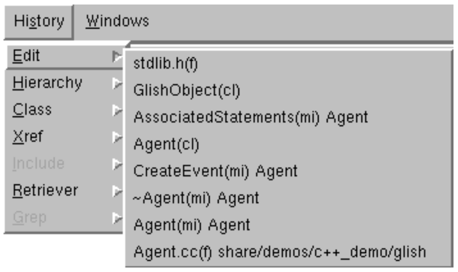
Menus

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**, **Windows**, and **Help** 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’s view of your project.

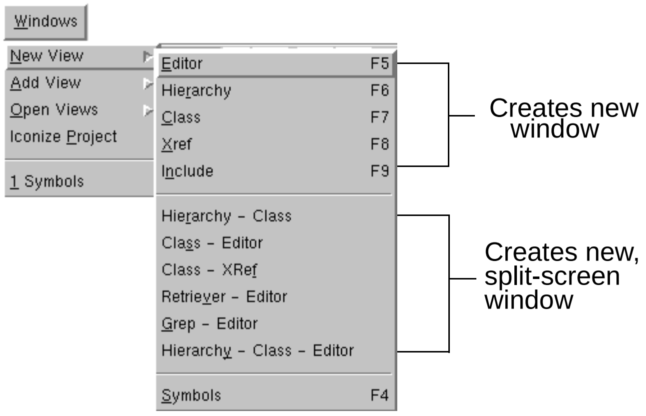
Figure 5: History Menu



Windows Menu

The **Windows** menu contains the tools available in Source-Navigator.

Figure 6: Windows Menu

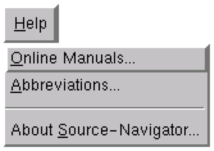


To add a new tool in a view, from the **Windows** menu, select **New View**. To add a tool to your current window, from the **Window** menu, select **Add View**. To change to a currently running instance of a tool, from the **Windows** menu, select **Open Views**.

Help Menu

The **Help** menu provides help for Source-Navigator.

Figure 7: Help Menu

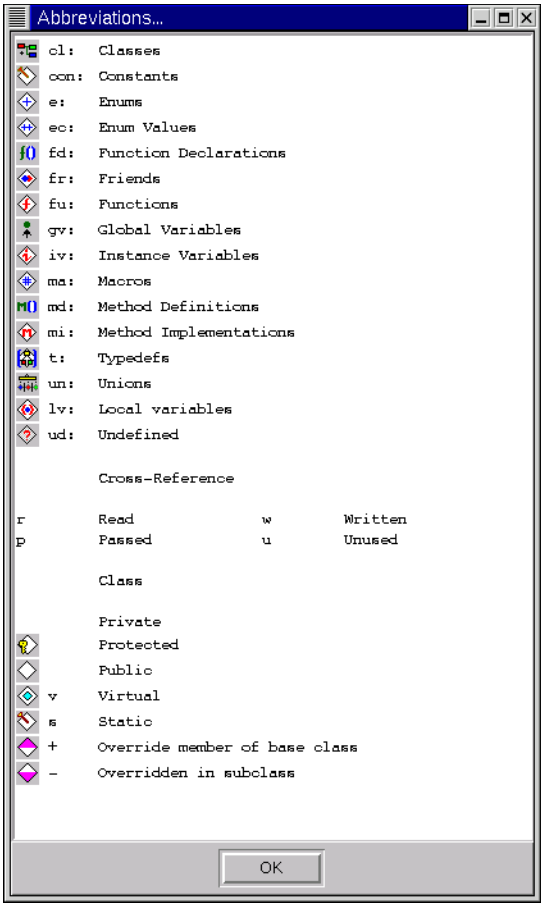


Online Manuals contains an HTML version of the current documentation. **Abbreviations** opens the **Abbreviations** dialog, which displays a list of abbreviations Source-Navigator uses in its browsers. For more information see “Symbol and Type Abbreviations” below. **About Source-Navigator** displays copyright and version information about Source-Navigator.

Symbol and Type Abbreviations

Source-Navigator uses the following abbreviations in its browsers. This dialog is accessible from the **Help** menu by selecting **Abbreviations**.

Figure 8: Abbreviations Dialog



Abbreviations...

	cl:	Classes
	con:	Constants
	e:	Enums
	ec:	Enum Values
	fd:	Function Declarations
	fr:	Friends
	fu:	Functions
	gv:	Global Variables
	iv:	Instance Variables
	ma:	Macros
	md:	Method Definitions
	mi:	Method Implementations
	t:	Typedefs
	un:	Unions
	lv:	Local variables
	ud:	Undefined

Cross-Reference

r	Read	w	Written
p	Passed	u	Unused

Class

	Private
	Protected
	Public
v	Virtual
s	Static
+	Override member of base class
-	Overridden in subclass

OK

General Window Features

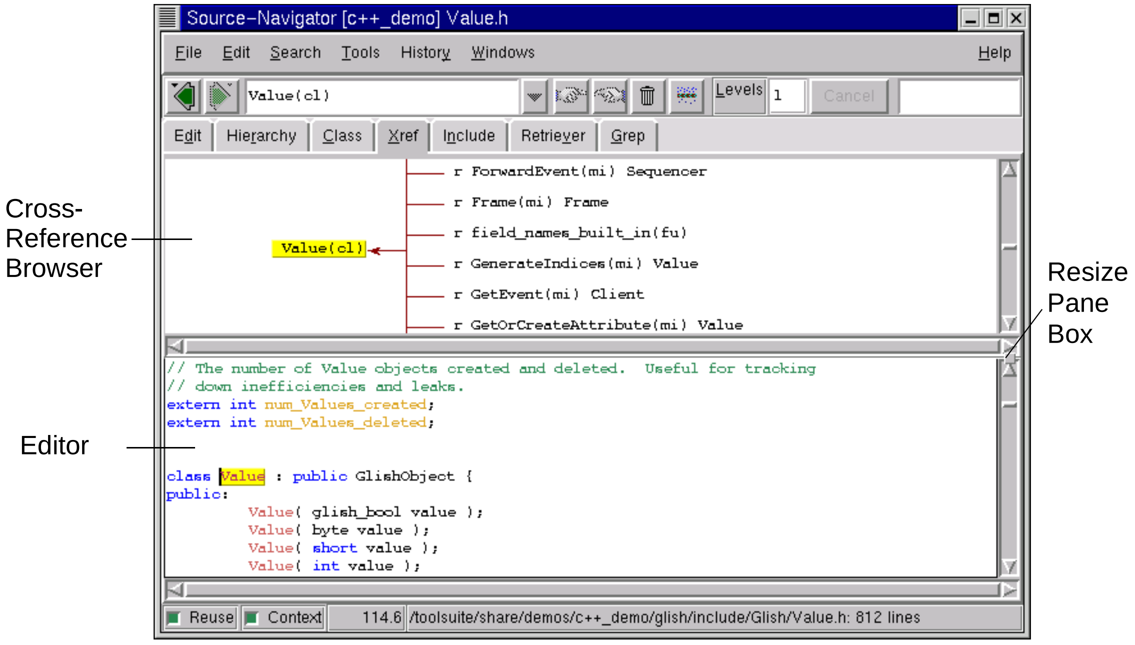
You may reuse Source-Navigator windows and adjust the window column size. The status line, located at the bottom of a window, shows information such as the number of lines of source code and the current directory.

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** appears in the same window. Depending upon your settings in the **Project Preferences** dialog (see page 28), the new **Editor** appears either at the bottom of the window (Vertical) or on the right side of the window (Horizontal).

Figure 9: Vertical Windows

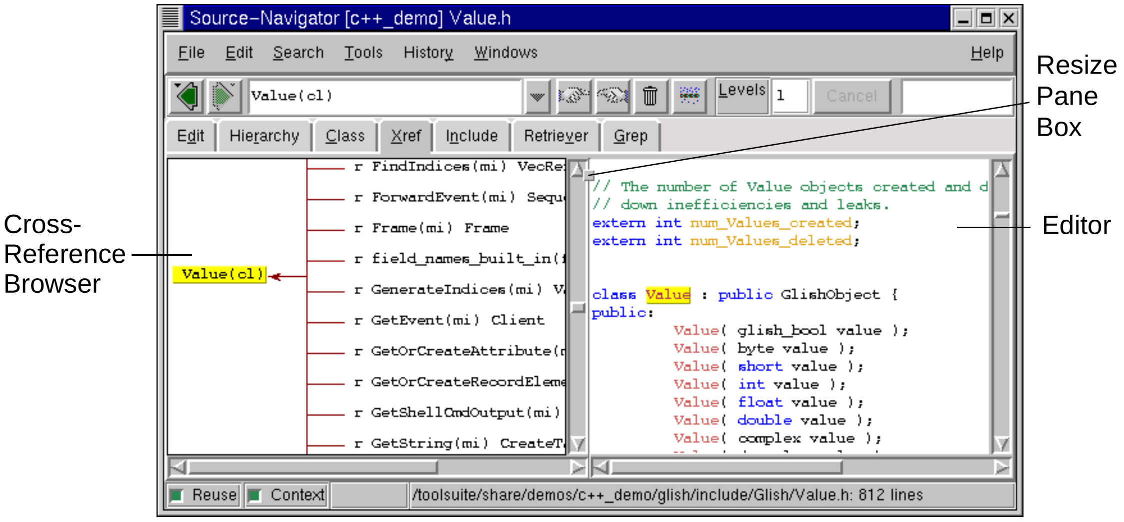


```
r ForwardEvent(mi) Sequencer
r Frame(mi) Frame
r field_names_built_in(fu)
r GenerateIndices(mi) Value
r GetEvent(mi) Client
r GetOrCreateAttribute(mi) Value
```

```
// The number of Value objects created and deleted.  Useful for tracking
// down inefficiencies and leaks.
extern int num_Values_created;
extern int num_Values_deleted;


class Value : public GlishObject {
public:
        Value( glish_bool value );
        Value( byte value );
        Value( short value );
        Value( int value );
```

114.6 /toolsuite/share/demos/c++_demo/glish/include/Glish/Value.h: 812 lines

Figure 10: Horizontal Windows


Source–Navigator [c++_demo] Value.h
File Edit Search Tools History Windows Help
Value(cl)
Levels 1
Cancel
Edit Hierarchy Class Xref Include Retriever Grep
Value(cl)
Reuse Context
Resize Pane Box
Cross-Reference Browser
Editor

```
r FindIndices(mi) VecRe
r ForwardEvent(mi) Sequ
r Frame(mi) Frame
r field_names_built_in(f
r GenerateIndices(mi) V
r GetEvent(mi) Client
r GetOrCreateAttribute(r
r GetOrCreateRecordEleme
r GetShellCmdOutput(mi)
r GetString(mi) CreateT
```

```
// The number of Value objects created and d
// down inefficiencies and leaks.
extern int num_Values_created;
extern int num_Values_deleted;


class Value : public GlishObject {
public:
        Value( glish_bool value );
        Value( byte value );
        Value( short value );
        Value( int value );
        Value( float value );
        Value( double value );
        Value( complex value );
```

/toolsuite/share/demos/c++_demo/glish/include/Glish/Value.h: 812 lines

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

Reusing Windows

When the **Reuse** button in the window status line is selected, a new view is displayed in the current window when you change to another tool. When the **Reuse** button is deselected, when you change to another tool, a new window opens.

Figure 11: Reuse Button Selected

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.

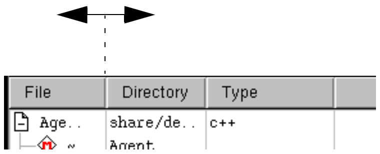
Figure 12: Context Button Selected

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 Column Width 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.

Figure 13: Adjusting Column Width



Using Filters

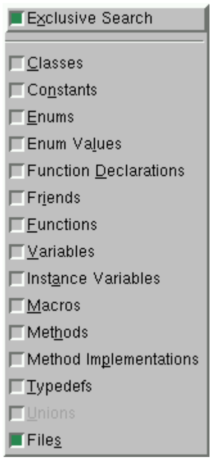
Source-Navigator provides several ways to filter the symbols displayed in the **Symbol Browser**. You can:

- use the **List Filter** buttons (see page 39).
- use the **Symbol Selectors** from the **View** menu.
- use **Column Filters** (see page 41).

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**.

Figure 14: 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 field. 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" on page 15 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.

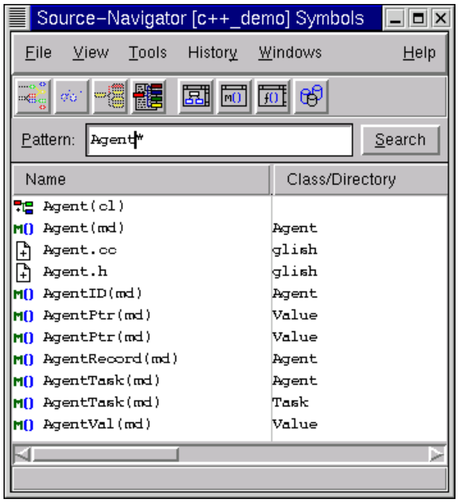
Table 1: Pattern Interpretation of Special Characters

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, the results would be as shown in Figure 15.

Figure 15: Symbol Browser Showing Filter Results



Pattern: Agent*

```
Name                Class/Directory
Agent(cl)
Agent(md)           Agent
Agent.cc            glish
Agent.h             glish
AgentID(md)         Agent
AgentPtr(md)        Value
AgentPtr(md)        Value
AgentRecord(md)     Agent
AgentTask(md)       Agent
AgentTask(md)       Task
AgentVal(md)        Value
```

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

Printing from Source-Navigator

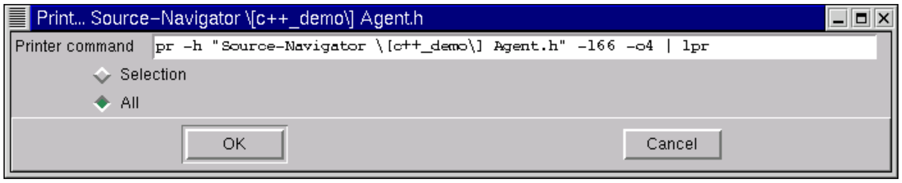
To print the contents of Source-Navigator browsers, such as the **Editor** or **Cross-Reference Browser**, from the **File** menu, select **Print**.

Print Dialog

The print dialog box varies with different printers, printer drivers, and platforms.

UNIX

Figure 16: UNIX Print Dialog



Printer command: `pr -h "Source-Navigator \[c++_demo\] Agent.h" -l66 -o4 | lpr`

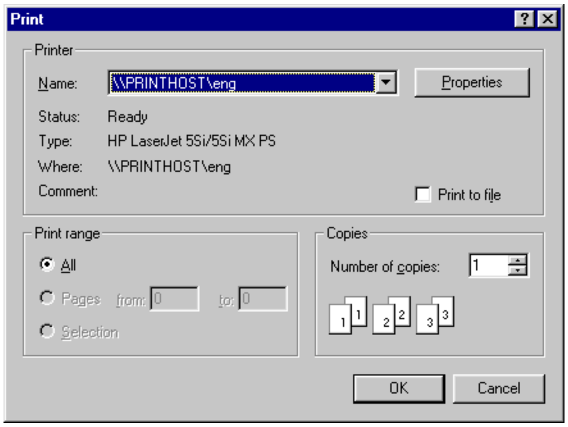
The default command line shown in the **Printer command** text box is set from the **Others** tab of the **Project Preferences** dialog (see “Others tab” on page 31).

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.

Windows

Figure 17: Windows Print Dialog



Name: \\PRINTHOST\eng
Status: Ready
Type: HP LaserJet 5Si/5Si MX PS
Where: \\PRINTHOST\eng
Comment:
Number of copies: 1

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.

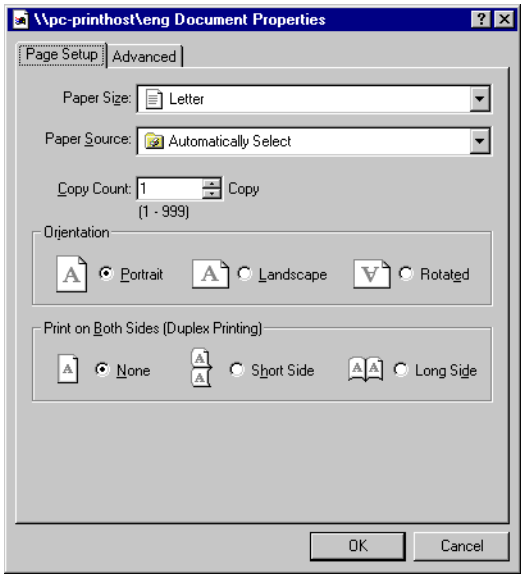
Copies

Number of copies
: Choose the number of copies to print.

Properties

If you click the **Properties** button, the **Document Properties** dialog appears.

Figure 18: Document Properties Dialog



\\pc-printhost\eng Document Properties

Page Setup | Advanced

Paper Size: Letter

Paper Source: Automatically Select

Copy Count: 1 Copy (1 - 999)

Orientation: [✓ Portrait] Landscape Rotated

Print on Both Sides (Duplex Printing): [✓ None] Short Side Long Side

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.

3

Customizing Source-Navigator

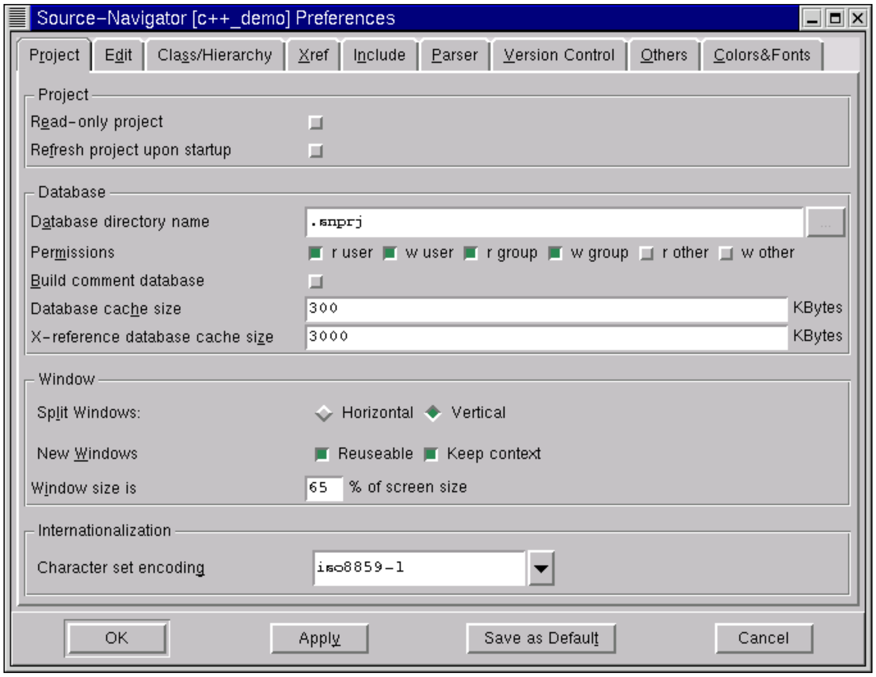
This chapter describes how to customize Source-Navigator to reflect your preferences. For additional information on changing the start-up and runtime behaviors, see “Customization” on page 137.

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.

Figure 19: Project Tab of the Preferences Dialog



Project
- Read-only project ☐
- Refresh project upon startup ☐

Database
- Database directory name: .snprj
- Permissions: ☑ r user ☑ w user ☑ r group ☑ w group ☐ r other ☐ w other
- Build comment database ☐
- Database cache size: 300 KBytes
- X-reference database cache size: 3000 KBytes

Window
- Split Windows: ◇ Horizontal ◆ Vertical
- New Windows: ☑ Reuseable ☑ Keep context
- Window size is 65 % of screen size

Internationalization
- Character set encoding: iso8859-1

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” on page 51.

Class/Hierarchy see “Class/Hierarchy Preferences” on page 59.

Xref see “Cross-Reference Preferences” on page 69.

Include see “Include Preferences” on page 75.

Version Control see “Version Control Preferences” on page 94.

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 filesystem), 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 four 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` for Tcl" on page 168.

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 eight 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` for Tcl” on page 168.

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

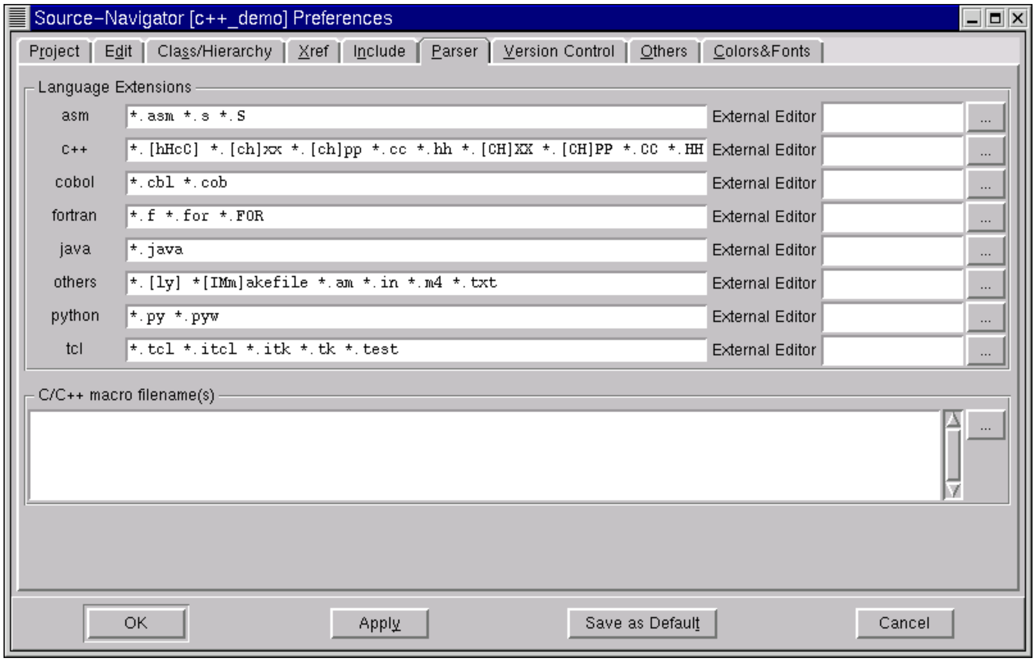
This combo-box allows you to choose the character set encoding for your project to match the character set of your source files.

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.

Figure 20: Parser Tab of the Preferences Dialog



Language Extensions

asm	`*.asm *.s *.S`	External Editor	...
c++	`*.[hHcC] *.[ch]xx *.[ch]pp *.cc *.hh *.[CH]XX *.[CH]PP *.CC *.HH`	External Editor	...
cobol	`*.cbl *.cob`	External Editor	...
fortran	`*.f *.for *.FOR`	External Editor	...
java	`*.java`	External Editor	...
others	`*.[ly] *[IMm]akefile *.am *.in *.m4 *.txt`	External Editor	...
python	`*.py *.pyw`	External Editor	...
tcl	`*.tcl *.itcl *.itk *.tk *.test`	External Editor	...

C/C++ macro filename(s)

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 “Adding Parsers” on page 153.

Table 2: 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 *.am *.in *.m4 *.txt`
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 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

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.

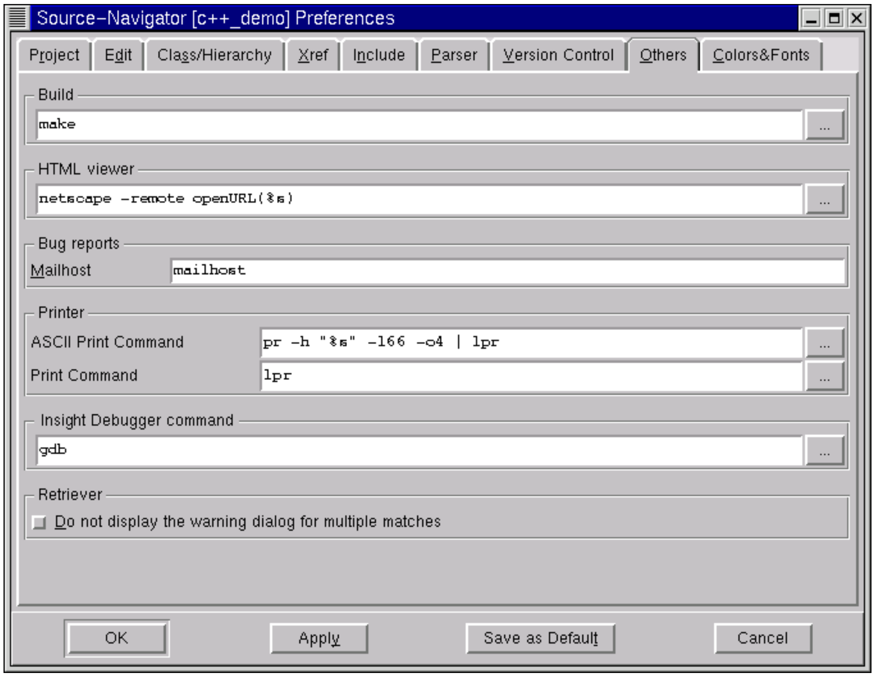
Macro files are ASCII files and every non-blank line is a macro directive. Leading blanks and # characters are stripped, and if the first character of a line is an apostrophe (`'`), the line is treated as a comment. The macro file can contain continuation lines, for example:

```
define ABC\
    \
    5
```

Others tab

The **Others** tab allows you to configure external interfaces.

Figure 21: Others Tab of the Preferences Dialog



Build
make

HTML viewer
netscape -remote openURL(%s)

Bug reports
Mailhost mailhost

Printer
ASCII Print Command pr -h "%s" -l66 -o4 | lpr
Print Command lpr

Insight Debugger command
gdb

Retriever
Do not display the warning dialog for multiple matches

Build

Type the executable (shell) command used to start your `make` system.

HTML viewer

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

This preference can only be set in UNIX.

Bug reports

Mailhost

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

Printer

Printer preferences can only be set in UNIX.

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 Windows **Registry Editor** to decide how to print.

Insight Debugger command

`gdb` is the executable (shell) command used to start **Insight**.

Retriever

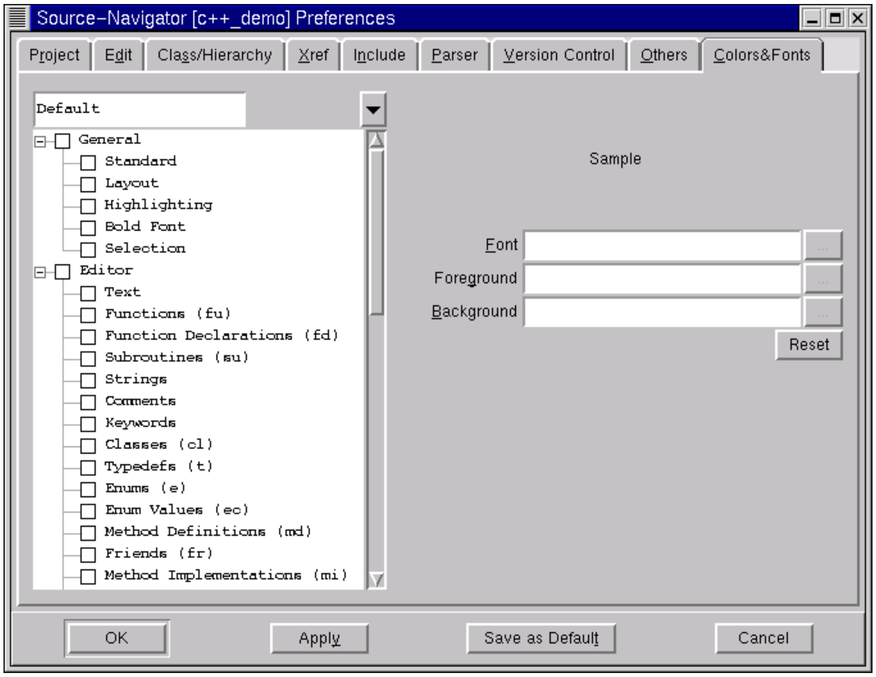
Do not display the warning dialog for multiple matches

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 page 79 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.

Figure 22: Colors & Fonts Tab of the Preferences Dialog



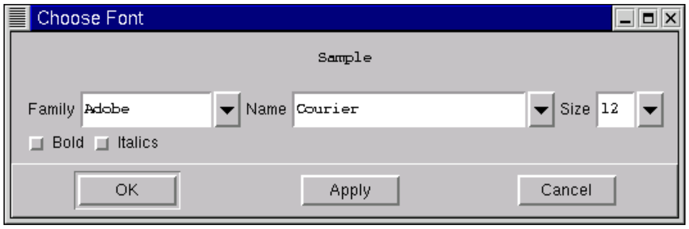
From the list of components, choose the item you want to change. The current settings appear in the **Font**, **Foreground**, and **Background** text boxes, and “Sample” displays the text with these settings.

Changing Fonts

Changing the font assigned to a component varies, depending upon your platform.

On UNIX, click the “**...**” button at the end of the **Font** text entry box. The **Choose Font** dialog appears:

Figure 23: UNIX Font Dialog

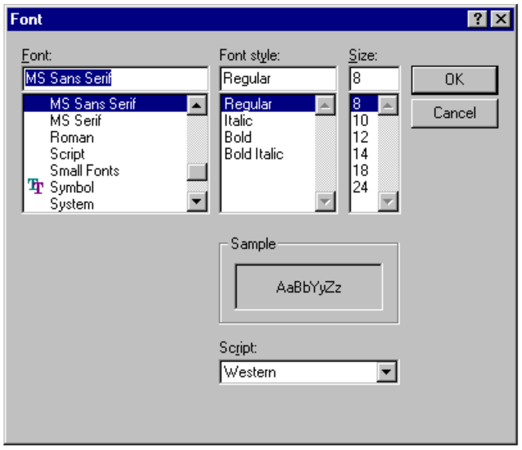


Family: Adobe
Name: Courier
Size: 12

Choose the **Family**, **Name**, and **Size** of the 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.

On Windows, click the “**...**” button at the end of the **Font** text entry box. The **Font** window appears.

Figure 24: Windows Font Dialog



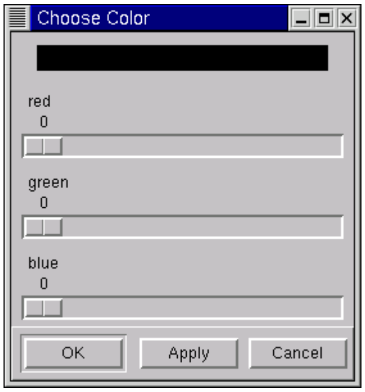
Choose the **Font**, **Font style**, and **Size** of the font to use and your changes appear in the **Sample** box. When you are satisfied, click **OK**. Your changes take effect the next time you access the item you changed.

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.

Figure 25: Choose Color Dialog



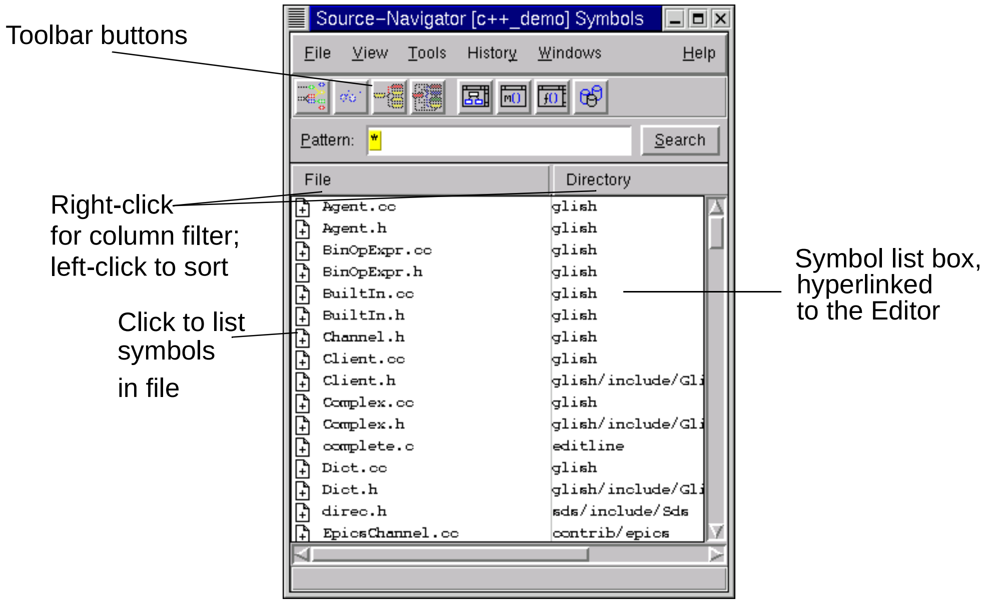
Move the red, green, and blue sliders until the color you want appears in the box at the top of the dialog. Click the **Apply** button to see how your text or background looks with the new color. If you are satisfied, click **OK**. Your changes take effect the next time you start Source-Navigator.

4

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.

Figure 26: Symbol Browser Window


Toolbar buttons
Right-click for column filter; left-click to sort
Click to list symbols in file
Symbol list box, hyperlinked to the Editor
Source-Navigator [c++_demo] Symbols
File View Tools History Windows Help
Pattern: *
Search

File	Directory
Agent.cc	glish
Agent.h	glish
BinOpExpr.cc	glish
BinOpExpr.h	glish
BuiltIn.cc	glish
BuiltIn.h	glish
Channel.h	glish
Client.cc	glish
Client.h	glish/include/Gli
Complex.cc	glish
Complex.h	glish/include/Gli
complete.c	editline
Dict.cc	glish
Dict.h	glish/include/Gli
direc.h	sds/include/Sds
EpicsChannel.cc	contrib/epics

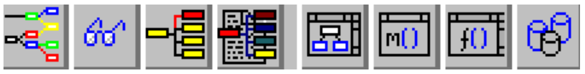
Using the Symbol Browser

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 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 page 45.

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 Figure 27.

Figure 27: 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 "Customization" on page 137.

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 page 57.

Class Browser button

This button starts the **Class Browser**. For more information about this tool, see page 61.

Cross-Reference Browser button

Select this button to start the **Cross-Reference Browser**. For more information about this tool, see page 65.

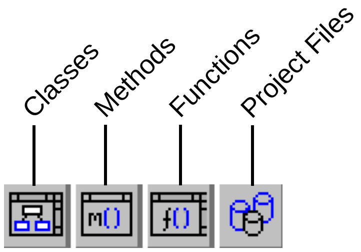
Include Browser button

Select this button to start the **Include Browser**. For more information about this tool, see page 73.

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**.

Figure 28: Filter Toolbar Buttons

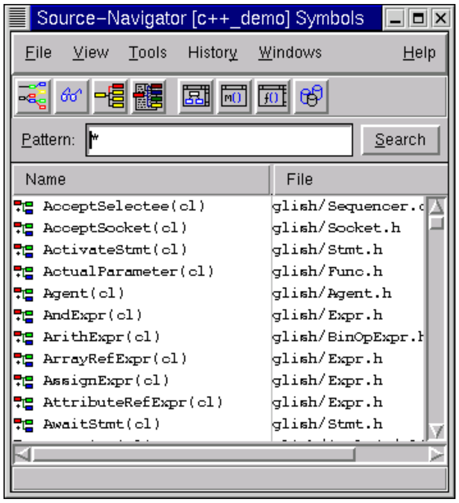


For details on how to filter for other symbols, see “Using Filters” on page 19.

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.

Figure 29: Symbol Browser with Exclusive Search and Classes Selected

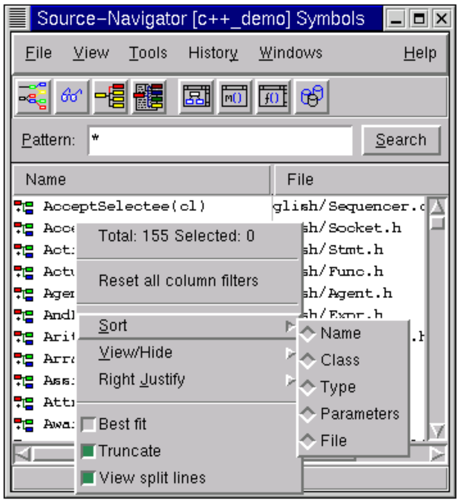


Pattern: *

Name	File
AcceptSelectee(cl)	glish/Sequencer.c
AcceptSocket(cl)	glish/Socket.h
ActivateStmt(cl)	glish/Stmt.h
ActualParameter(cl)	glish/Func.h
Agent(cl)	glish/Agent.h
AndExpr(cl)	glish/Expr.h
ArithExpr(cl)	glish/BinOpExpr.h
ArrayRefExpr(cl)	glish/Expr.h
AssignExpr(cl)	glish/Expr.h
AttributeRefExpr(cl)	glish/Expr.h
AwaitStmt(cl)	glish/Stmt.h

To find the number of symbols matching your selection, right-click in the **Symbol** list box. From the popup menu you may sort the symbol list by column and hide columns to simplify the display.

Figure 30: Symbol Browser Right Button Menu



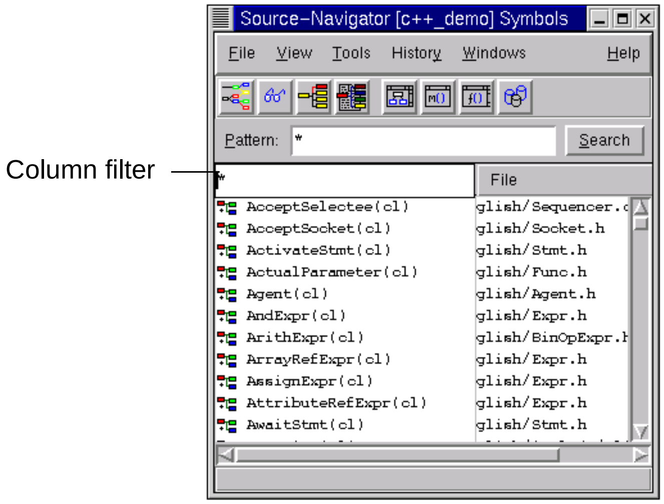
For faster searching you can use the toolbar buttons to browse project files, functions, methods, and classes. For more information on the toolbar see “Toolbar Buttons” on page 38.

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.

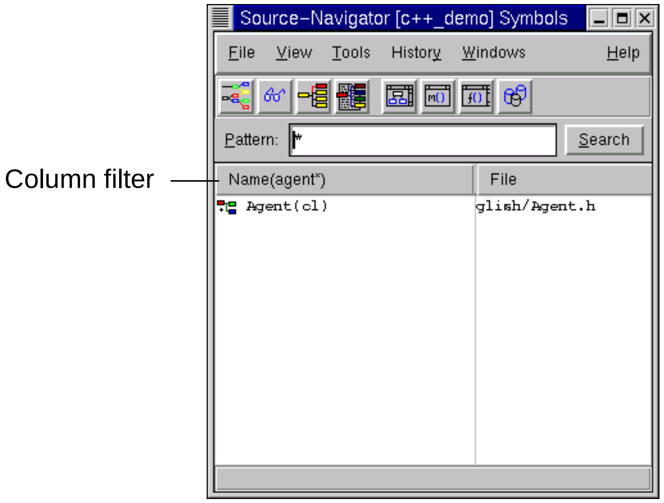
Figure 31: Column Filter Box


Column filter
Source–Navigator [c++_demo] Symbols
File View Tools History Windows Help
Pattern: *
Search
*
File

```
AcceptSelectee(cl)      glish/Sequencer.c
AcceptSocket(cl)        glish/Socket.h
ActivateStmt(cl)        glish/Stmt.h
ActualParameter(cl)     glish/Func.h
Agent(cl)               glish/Agent.h
AndExpr(cl)             glish/Expr.h
ArithExpr(cl)           glish/BinOpExpr.h
ArrayRefExpr(cl)        glish/Expr.h
AssignExpr(cl)          glish/Expr.h
AttributeRefExpr(cl)    glish/Expr.h
AwaitStmt(cl)           glish/Stmt.h
```

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

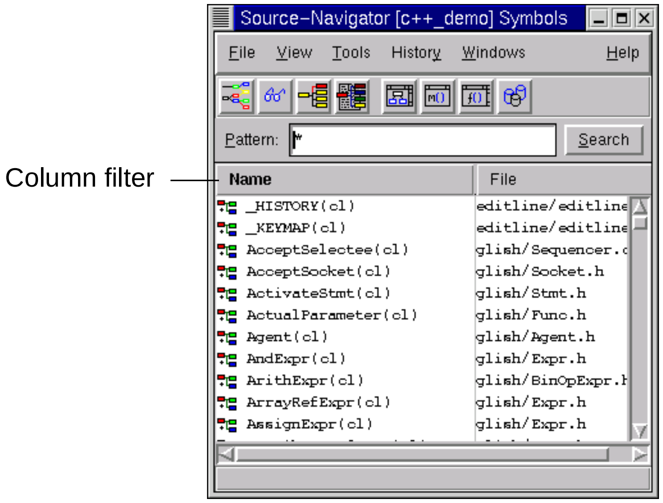
Figure 32: Symbol Browser Showing Pattern Matches


Column filter
Source–Navigator [c++_demo] Symbols
File View Tools History Windows Help
Pattern: *
Search
Name(agent*)
File

```
Agent(cl)               glish/Agent.h
```

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

Figure 33: Column Data Sorted


Column filter
Source–Navigator [c++_demo] Symbols
File View Tools History Windows Help
Pattern:
Search
Name
File

```
_HISTORY(cl)            editline/editline
_KEYMAP(cl)             editline/editline
AcceptSelectee(cl)      glish/Sequencer.c
AcceptSocket(cl)        glish/Socket.h
ActivateStmt(cl)        glish/Stmt.h
ActualParameter(cl)     glish/Func.h
Agent(cl)               glish/Agent.h
AndExpr(cl)             glish/Expr.h
ArithExpr(cl)           glish/BinOpExpr.h
ArrayRefExpr(cl)        glish/Expr.h
AssignExpr(cl)          glish/Expr.h
```

5

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” on page 51). For instructions on how to specify an external editor, see “Common editor configurations” on page 53.

The Editor Window

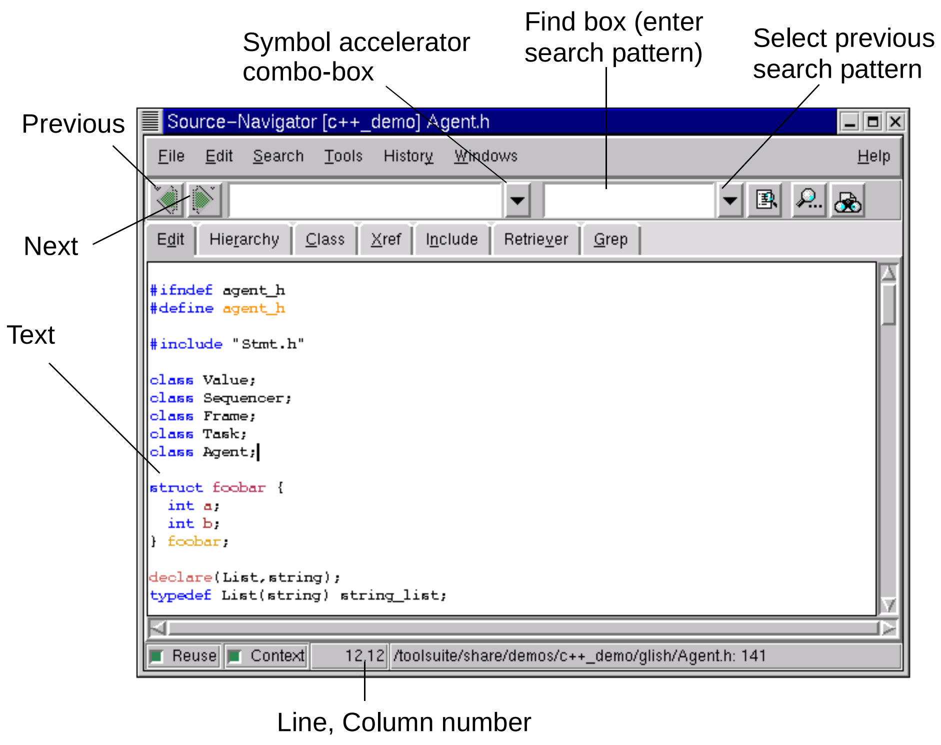
To open the **Editor**, double-click a symbol in the **Symbol Browser** (or other Source-Navigator browser window). Symbols are hyperlinked to the **Editor**, which displays the contents of a project source file and allows you to edit it.

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 a word selects the entire word; triple-clicking selects the entire line.

Figure 34: Editor Window

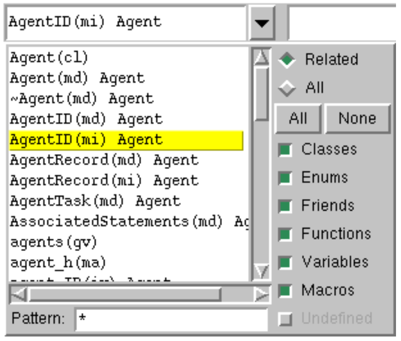

Symbol accelerator combo-box
Find box (enter search pattern)
Select previous search pattern
Previous
Next
Text
Line, Column number
Source–Navigator [c++_demo] Agent.h
File Edit Search Tools History Windows Help
Edit Hierarchy Class Xref Include Retriever Grep
Reuse Context 12,12 /toolsuite/share/demos/c++_demo/glish/Agent.h: 141

```
#ifndef agent_h
#define agent_h

#include "Stmt.h"

class Value;
class Sequencer;
class Frame;
class Task;
class Agent;

struct foobar {
  int a;
  int b;
} foobar;

declare(List,string);
typedef List(string) string_list;
```

Symbol Accelerator Combo-box

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

Figure 35: 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.

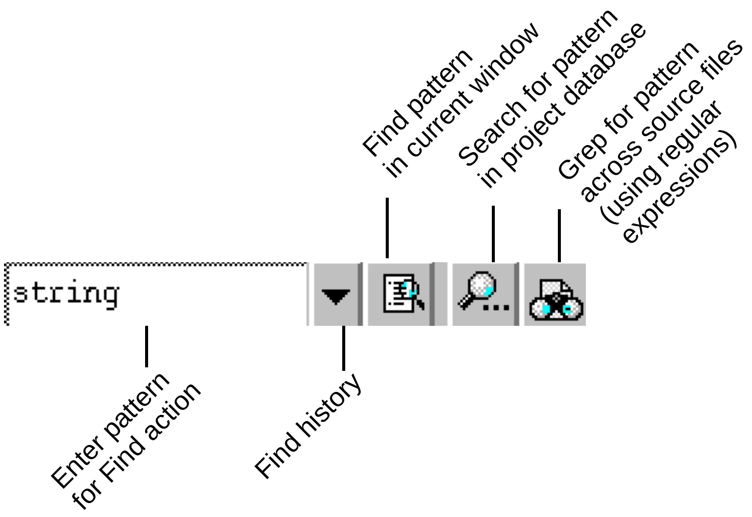
Find Box

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.

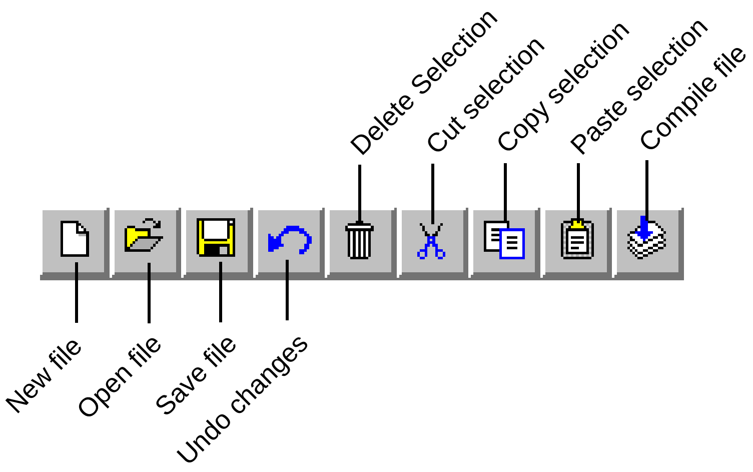
Figure 36: Pattern Searching Toolbar Buttons


Find pattern in current window
Search for pattern in project database
Grep for pattern across source files (using regular expressions)
string
Enter pattern for Find action
Find history

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.

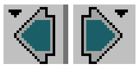
Figure 37: File and Text Management Toolbar Buttons



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.

Figure 38: View History Buttons



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.

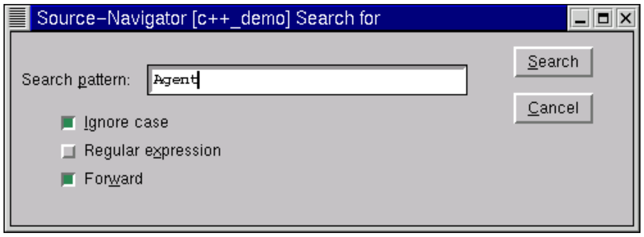
Search Menu

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**.

Figure 39: Find Dialog

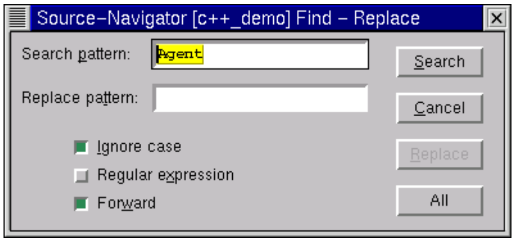


Type your string or pattern into the text box and click the **Search** button; you can click the **Search** button multiple times to find more instances of the string. Deselect **Ignore case** if you want a case-sensitive search. When **Regular expression** is selected, the pattern is treated as a regular 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**.

Figure 40: Replace Dialog



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 page 83.

Go To menu

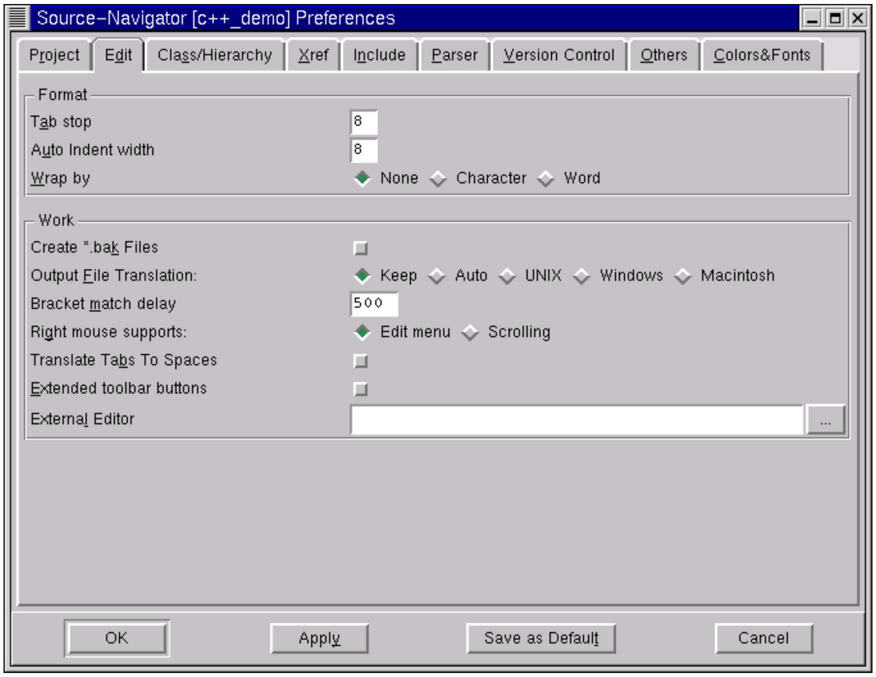
When you want to go to a specific line in the file, from the **Search** menu, select **Go To** → **Go To Line**. This allows you to type in the line number you wish to go to. **Set Mark** allows you to set a place in the file you wish to come back to later. **Go To Mark** jumps to the last mark you set. **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.

Figure 41: Edit Tab of the Preferences Dialog



Format
Tab stop: 8
Auto Indent width: 8
Wrap by: [✓ None] Character Word

Work
Create *.bak Files: ☐
Output File Translation: [✓ Keep] Auto UNIX Windows Macintosh
Bracket match delay: 500
Right mouse supports: [✓ Edit menu] Scrolling
Translate Tabs To Spaces: ☐
Extended toolbar buttons: ☐
External Editor: ...

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" on page 48.

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

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".

Notepad

On Windows, you can invoke Notepad by typing:

```
notepad %f
```

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.

Windows NT can execute DOS executables without launching a shell. The command line for a DOS version of vi is:

```
vi +%l %f
```

For information on customizing your key bindings or using another external editor, see “Customization” on page 137.

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 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.

6

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**.

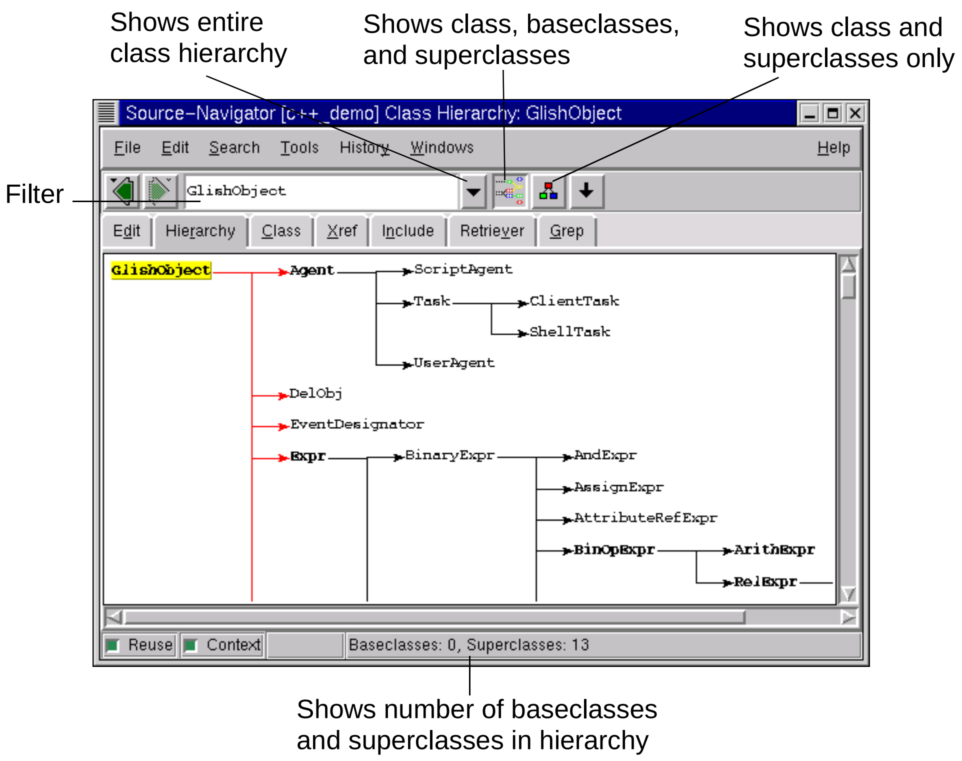
If class **a** is a superclass of class **b**, then class **b** is a subclass of class **a**.

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.

Figure 42: 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

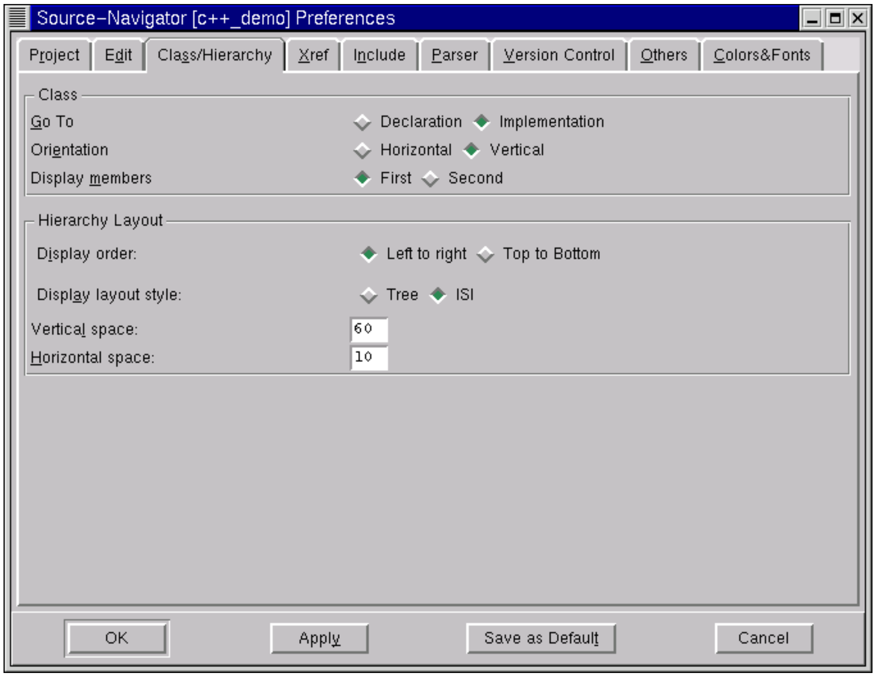
Displays the file names of the superclasses and subclasses.

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.

Figure 43: Class/Hierarchy Tab of the Preferences Dialog



Class
Go To: Declaration [✓ Implementation]
Orientation: Horizontal [✓ Vertical]
Display members: [✓ First] Second

Hierarchy Layout
Display order: [✓ Left to right] Top to Bottom
Display layout style: Tree [✓ ISI]
Vertical space: 60
Horizontal space: 10

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.

7

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 page 59).

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

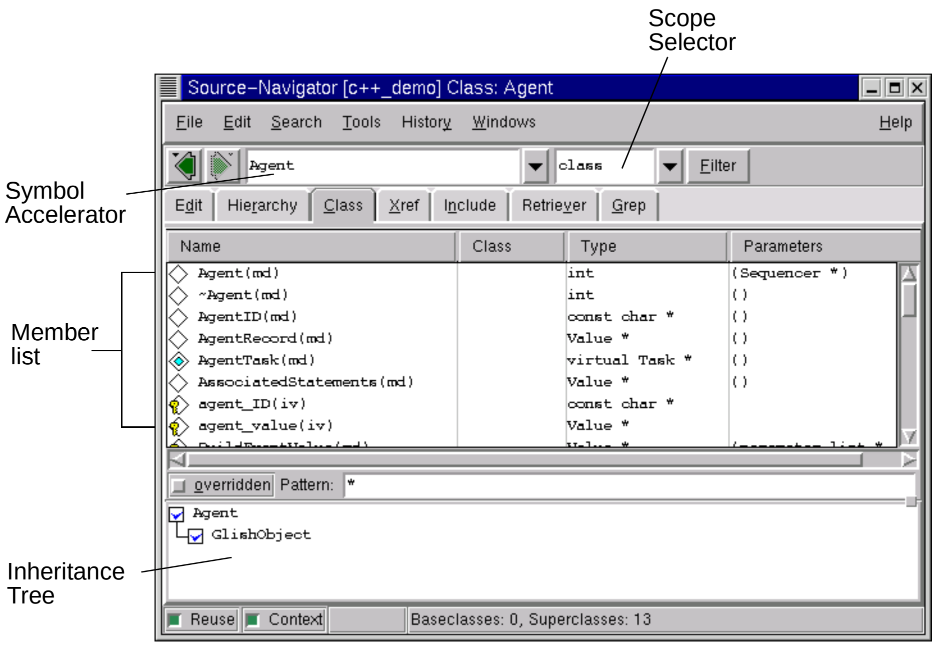
NOTE Source-Navigator treats structures, classes, and common blocks in the same way. The only difference is that classes have inheritance and the others do not.

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” on page 39).
- from the **Windows** menu, select **New View** → **Class**.
- choose the **Class** tab.

Figure 44: Class Browser Window



Name	Class	Type	Parameters
Agent(md)		int	(Sequencer *)
~Agent(md)		int	()
AgentID(md)		const char *	()
AgentRecord(md)		Value *	()
AgentTask(md)		virtual Task *	()
AssociatedStatements(md)		Value *	()
agent_ID(iv)		const char *	
agent_value(iv)		Value *	

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.

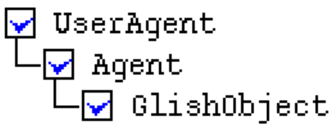
Member List

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 Figure 8 on page 16.

Inheritance Tree

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

Figure 45: Inheritance Tree



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.

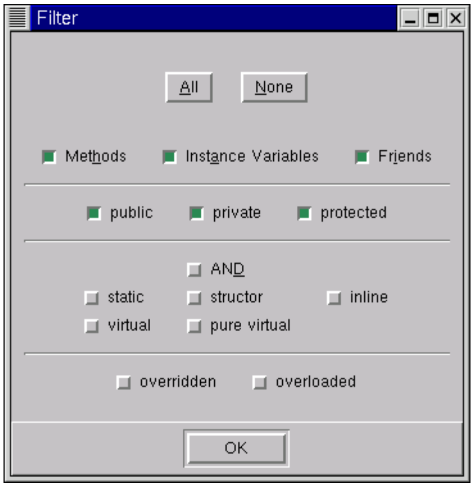
Double-click
: Starts the **Editor**, which displays the source file.

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.

Figure 46: Member List Filter Dialog



All
: Sets all items except AND, overridden, and overloaded.

None
: Clears all items except AND, overridden, and overloaded.

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.

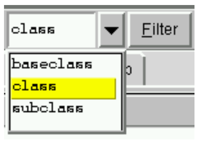
overloaded

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.

Figure 47: Scope Selector Menu



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.

8

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.

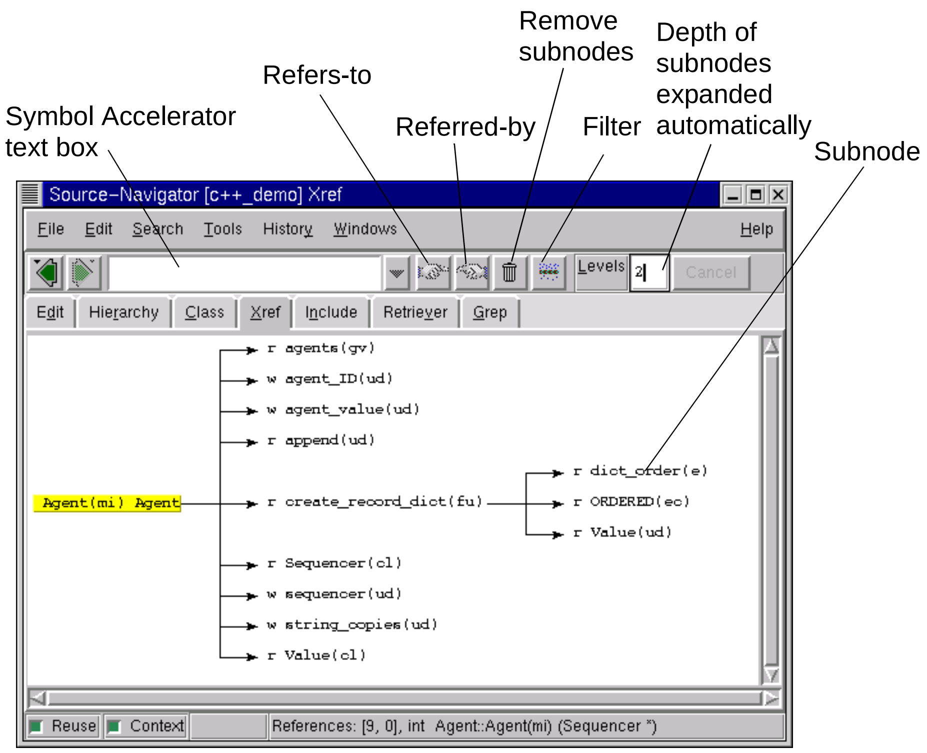
Source-Navigator creates the cross-reference database in the background, which enables you to work in other views. During this process, the **Cross-Reference** tool button is disabled (grayed-out). After the database is built, the **Cross-Reference** tool can be opened.

Using the Cross-Reference Browser

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 Figure 48.

Figure 48: Cross-Reference Browser Window



```
Agent(mi) Agent
r agents(gv)
w agent_ID(ud)
w agent_value(ud)
r append(ud)
r create_record_dict(fu)
    r dict_order(e)
    r ORDERED(ec)
    r Value(ud)
r Sequencer(cl)
w sequencer(ud)
w string_copies(ud)
r Value(cl)
```

References: [9, 0], int Agent::Agent(mi) (Sequencer *)

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

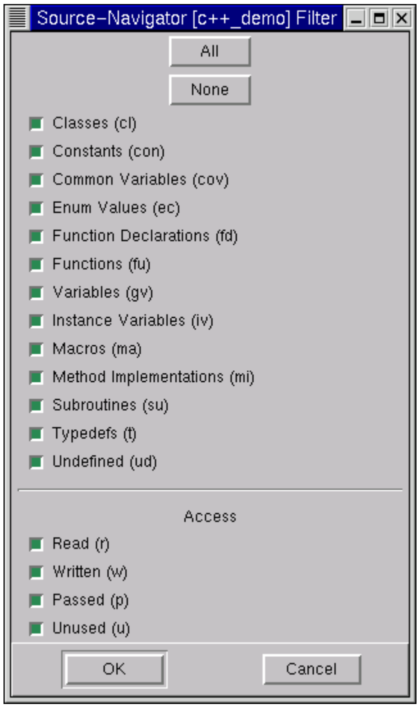
To set the number of subnode levels to display, enter a positive integer in the **Levels** text entry box.

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** button to bring up the **Filter** dialog. The symbols displayed by the **Cross-Reference Browser** are included based on these settings.

Figure 49: Cross-Reference Filter



- ✓ Classes (cl)
- ✓ Constants (con)
- ✓ Common Variables (cov)
- ✓ Enum Values (ec)
- ✓ Function Declarations (fd)
- ✓ Functions (fu)
- ✓ Variables (gv)
- ✓ Instance Variables (iv)
- ✓ Macros (ma)
- ✓ Method Implementations (mi)
- ✓ Subroutines (su)
- ✓ Typedefs (t)
- ✓ Undefined (ud)

Access

- ✓ Read (r)
- ✓ Written (w)
- ✓ Passed (p)
- ✓ Unused (u)

All

Sets all non-Access selections.

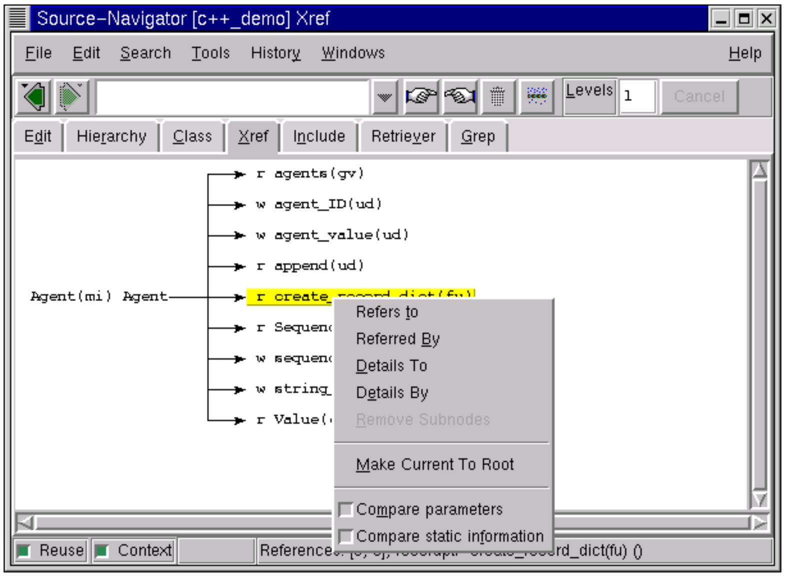
None

Clears all non-Access selections.

Cross-Reference Browser Details

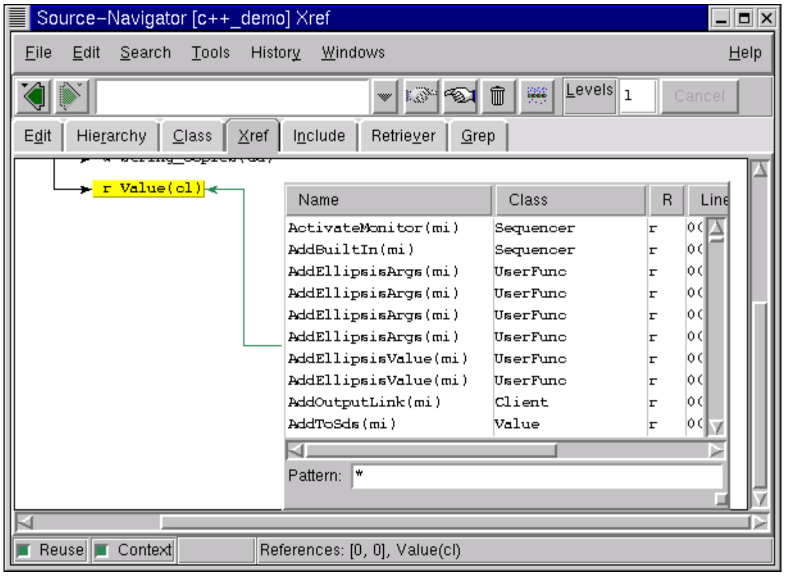
Right-clicking on a symbol in the **Cross-Reference Browser** brings up a popup menu that allows you to filter the list of symbols you’re working with, as well as to gather new information about the symbols you’re interested in.

Figure 50: Cross-Reference Browser Right Button Menu



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

Figure 51: Cross-Reference Browser Showing Details By Window

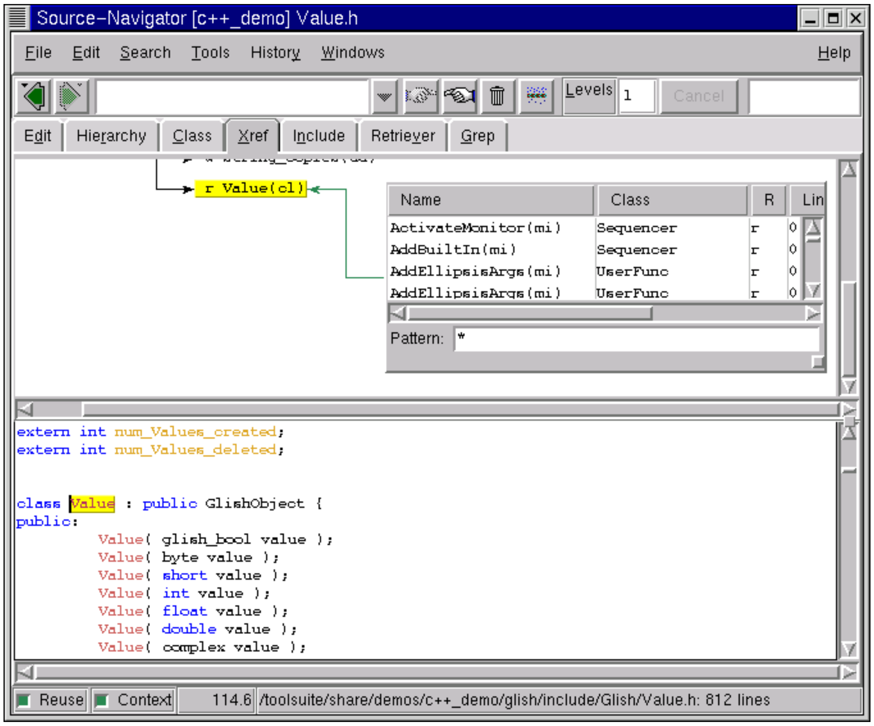

Source–Navigator [c++_demo] Xref
File Edit Search Tools History Windows Help
Levels 1 Cancel
Edit Hierarchy Class Xref Include Retriever Grep
r Value(cl)
Pattern: *
Reuse Context
References: [0, 0], Value(cl)

Name	Class	R	Line
ActivateMonitor(mi)	Sequencer	r	0(
AddBuiltIn(mi)	Sequencer	r	0(
AddEllipsisArgs(mi)	UserFunc	r	0(
AddEllipsisArgs(mi)	UserFunc	r	0(
AddEllipsisArgs(mi)	UserFunc	r	0(
AddEllipsisArgs(mi)	UserFunc	r	0(
AddEllipsisValue(mi)	UserFunc	r	0(
AddEllipsisValue(mi)	UserFunc	r	0(
AddOutputLink(mi)	Client	r	0(
AddToSds(mi)	Value	r	0(

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.

Figure 52: Editor Showing Referencing of Symbol



Name	Class	R	Lin
ActivateMonitor(mi)	Sequencer	r	0
AddBuiltIn(mi)	Sequencer	r	0
AddEllipsisArgs(mi)	UserFunc	r	0
AddEllipsisArgs(mi)	UserFunc	r	0

```
extern int num_Values_created;
extern int num_Values_deleted;


class Value : public GlishObject {
public:
        Value( glish_bool value );
        Value( byte value );
        Value( short value );
        Value( int value );
        Value( float value );
        Value( double value );
        Value( complex value );
```

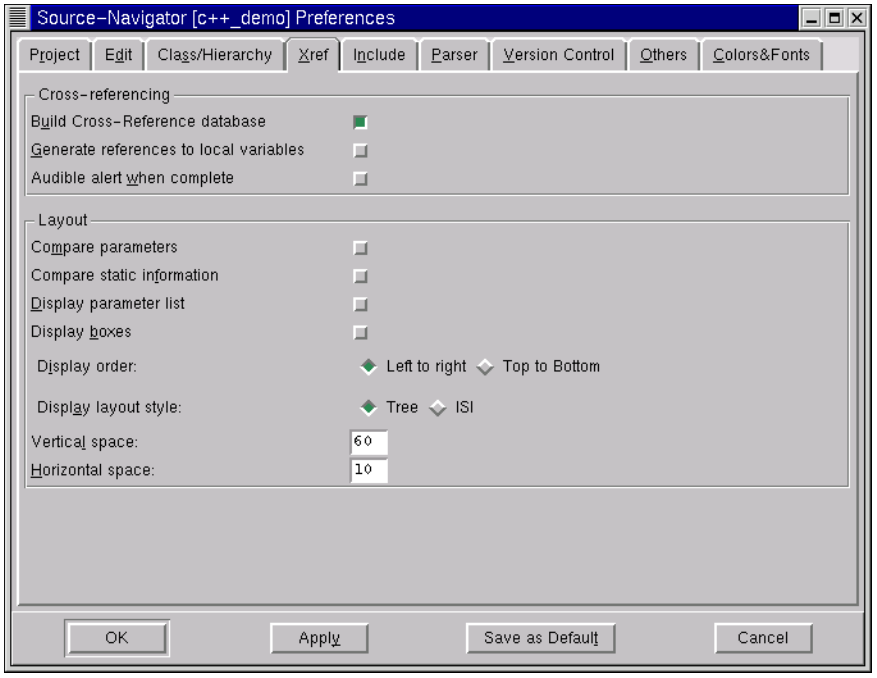
114.6 /toolsuite/share/demos/c++_demo/glish/include/Glish/Value.h: 812 lines

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.

Figure 53: Cross-Reference Tab of the Preferences Dialog


Source–Navigator [c++_demo] Preferences

Cross–referencing
Build Cross–Reference database [✓]
Generate references to local variables [ ]
Audible alert when complete [ ]

Layout
Compare parameters [ ]
Compare static information [ ]
Display parameter list [ ]
Display boxes [ ]
Display order: [✓ Left to right] Top to Bottom
Display layout style: [✓ Tree] ISI
Vertical space: 60
Horizontal space: 10

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 ten 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.

9

Include Browser

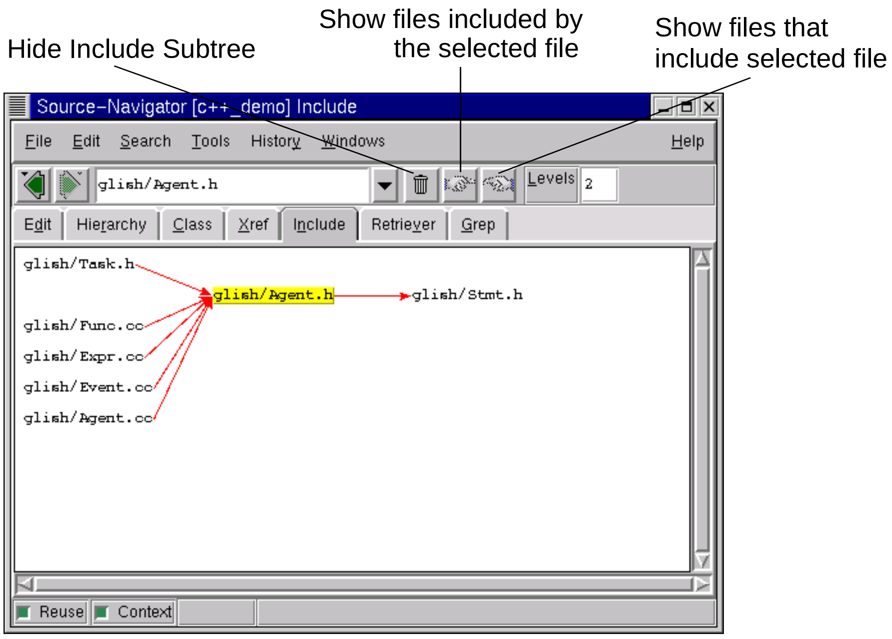
Some programming languages provide a facility for including other source files. In C/C++ this is achieved using the `#include` preprocessor directive. The **Include Browser** lets you display *Includes* and *Included by* relationships simultaneously.

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” on page 39).
- choose the **Include** tab.

Figure 54: Include Browser Window


Hide Include Subtree
Show files included by the selected file
Show files that include selected file
Source–Navigator [c++_demo] Include
File Edit Search Tools History Windows Help
glish/Agent.h
Levels 2
Edit Hierarchy Class Xref Include Retriever Grep
glish/Task.h
glish/Agent.h
glish/Stmt.h
glish/Func.cc
glish/Expr.cc
glish/Event.cc
glish/Agent.cc
Reuse Context

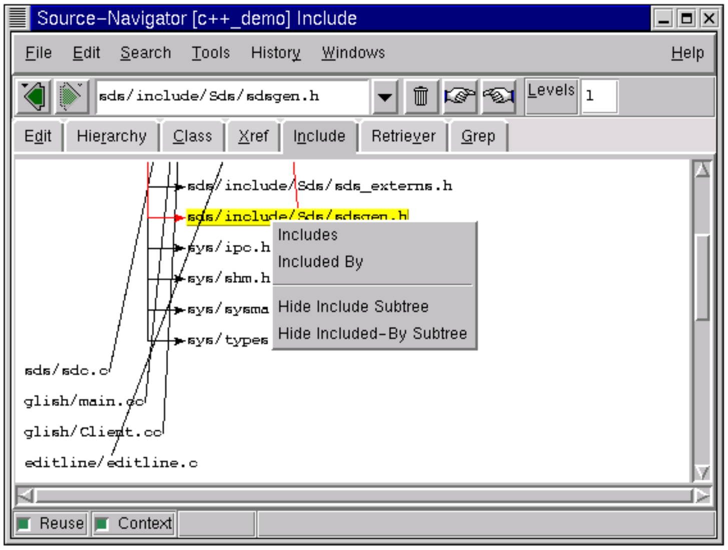
To see further relationships, after selecting a file in the **Include Browser** window, use the right pointing-hand tool icon to show files included by the selected file. Use the left pointing-hand tool icon to show files that include the selected file.

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

Reducing Displayed Information

Right-clicking on a symbol in the **Include Browser** window brings up a popup menu that allows you to show or hide include information.

Figure 55: Include Browser Right Button Menu



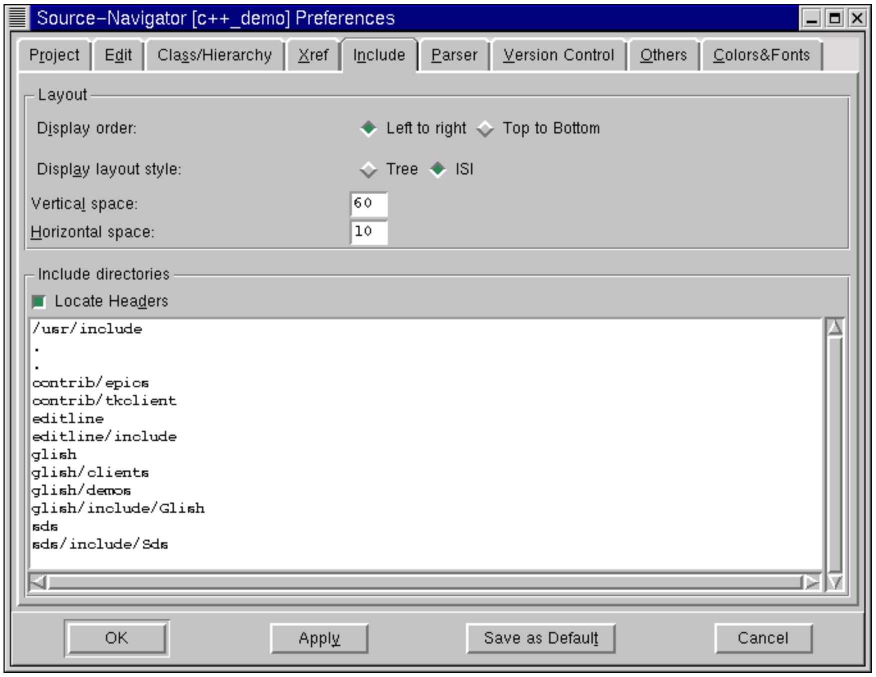
This is useful when the **Include Browser** displays more information than you need, and you'd like to hide all relationships but the particular one you're interested in.

Include Preferences

Preference settings for the **Include Browser** are located in the **Include** tab of the **Preferences** dialog.

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.

Figure 56: Include Tab of the Preferences Dialog



Layout

Display order: [✓ Left to right] Top to Bottom

Display layout style: Tree [✓ ISI]

Vertical space: 60

Horizontal space: 10

Include directories

[✓] Locate Headers

```
/usr/include
.
.
contrib/epics
contrib/tkclient
editline
editline/include
glish
glish/clients
glish/demos
glish/include/Glish
sds
sds/include/Sds
```

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 cross-referenced.

In the **Include Directories** box you can choose which directories should be searched for include files. Using the set of directories shown in Figure 56, 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**.

10

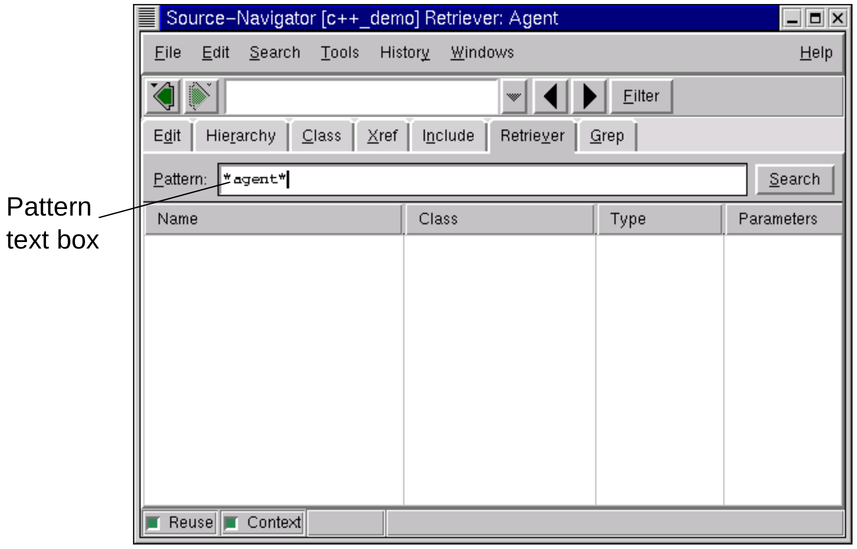
Retriever

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” on page 83.

Using the Retriever

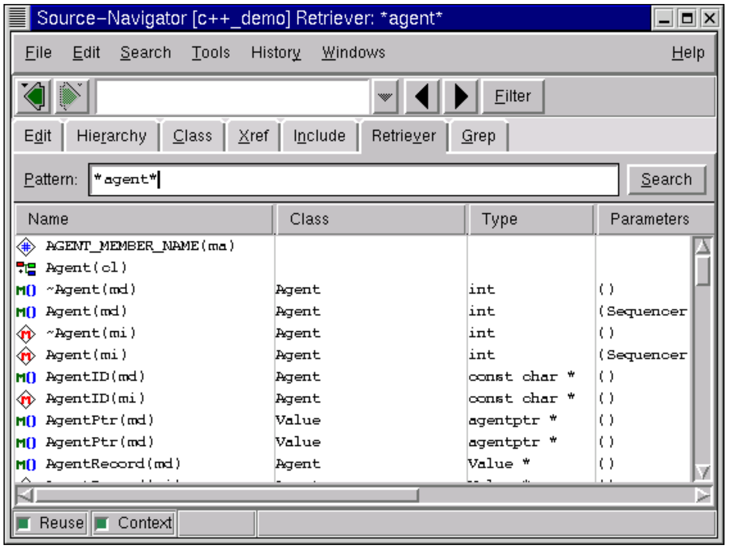
To find a symbol, enter its search pattern surrounded by asterisks (for example `*agent*`) in the **Pattern** text box and click **Search**. The *****, **?**, **[**, and **]** wildcard characters are supported.

Figure 57: Retriever Window


Source–Navigator [c++_demo] Retriever: Agent
File Edit Search Tools History Windows Help
Filter
Edit Hierarchy Class Xref Include Retriever Grep
Pattern: *agent*
Search
Name Class Type Parameters
Reuse Context
Pattern text box

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

Figure 58: Retriever Window Showing Search Results


Source–Navigator [c++_demo] Retriever: *agent*
File Edit Search Tools History Windows Help
Filter
Edit Hierarchy Class Xref Include Retriever Grep
Pattern: *agent*
Search
Reuse Context

Name	Class	Type	Parameters
AGENT_MEMBER_NAME(ma)			
Agent(cl)			
~Agent(md)	Agent	int	()
Agent(md)	Agent	int	(Sequencer
~Agent(mi)	Agent	int	()
Agent(mi)	Agent	int	(Sequencer
AgentID(md)	Agent	const char *	()
AgentID(mi)	Agent	const char *	()
AgentPtr(md)	Value	agentptr *	()
AgentPtr(md)	Value	agentptr *	()
AgentRecord(md)	Agent	Value *	()

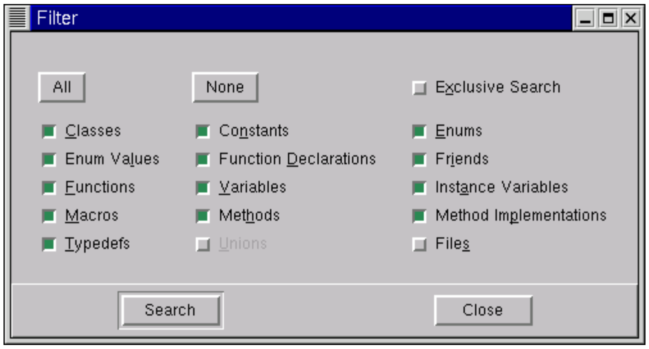
Double-clicking an item in the list opens the **Editor** showing the symbol in context in the source code.

For more information on reusing windows for multiple searches, see “Reusing Windows” on page 18.

Retriever Filter

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

Figure 59: Retriever Filter Dialog



- [✓] Classes
- [✓] Constants
- [✓] Enums
- [✓] Enum Values
- [✓] Function Declarations
- [✓] Friends
- [✓] Functions
- [✓] Variables
- [✓] Instance Variables
- [✓] Macros
- [✓] Methods
- [✓] Method Implementations
- [✓] Typedefs
- [ ] Unions
- [ ] Files
- [ ] Exclusive Search

All
: Selects all items except **Unions** and **Files**.

None
: Clears all selections.

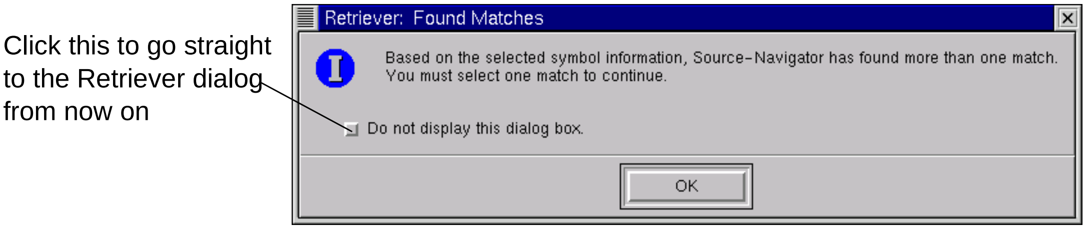
Executive Search
: If the **Exclusive Search** box is selected, you may choose one symbol type to search for. If it's deselected, you may combine symbol types for more complex searches.

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.

Figure 60: Multiple Symbols Notification Dialog

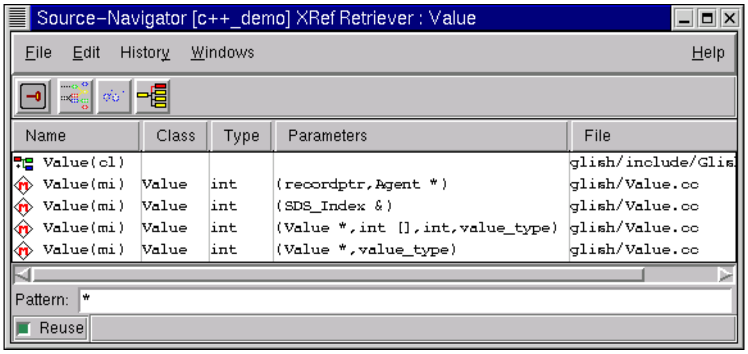

Click this to go straight to the Retriever dialog from now on
Retriever: Found Matches
OK

Based on the selected symbol information, Source-Navigator has found more than one match. You must select one match to continue.

Do not display this dialog box.

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 page 31).

Figure 61: Xref Retriever Window



Name	Class	Type	Parameters	File
Value(cl)				glish/include/Glis
Value(mi)	Value	int	(recordptr,Agent *)	glish/Value.cc
Value(mi)	Value	int	(SDS_Index &)	glish/Value.cc
Value(mi)	Value	int	(Value *,int [],int,value_type)	glish/Value.cc
Value(mi)	Value	int	(Value *,value_type)	glish/Value.cc

Pattern: *

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" on page 65 for more information.

11

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:

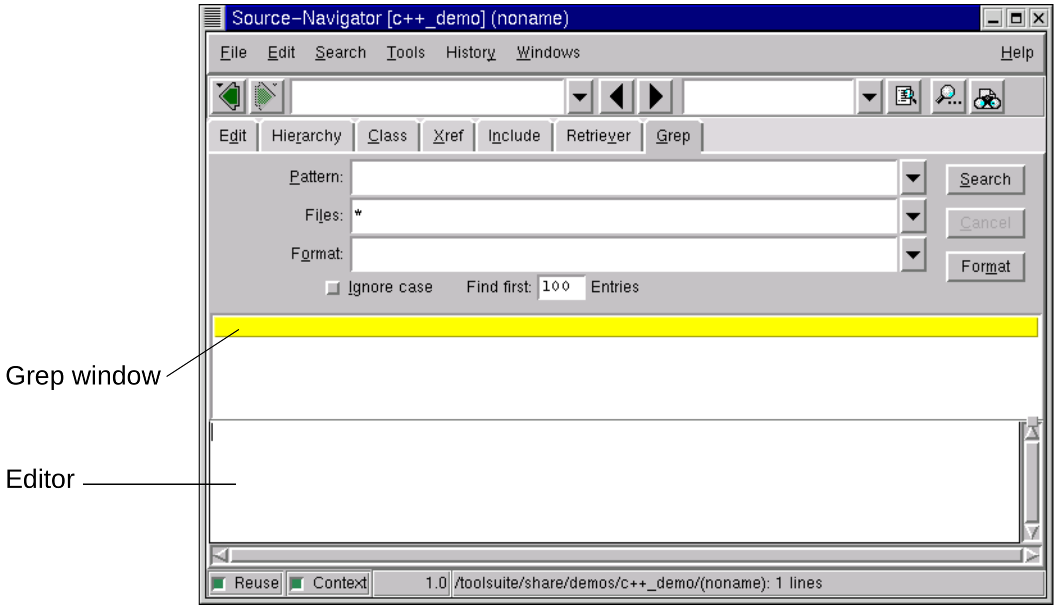
- allows you to search multiple directories.
- searches only the files in your project.
- can be restricted to a subset of the files in your project.
- saves a list of previously performed searches so you can repeat a search later.
- provides “one-click” **Editor** access to the lines of code matching your search.

To search for text patterns in the symbols database, use the **Retriever** tool. For more information, see “Retriever” on page 79.

Using Grep

A convenient way to use the **Grep** tool is by using a **Grep-Editor** window. From the **Windows** menu, select **New View** → **Grep-Editor** (for more information on split windows, see page 16).

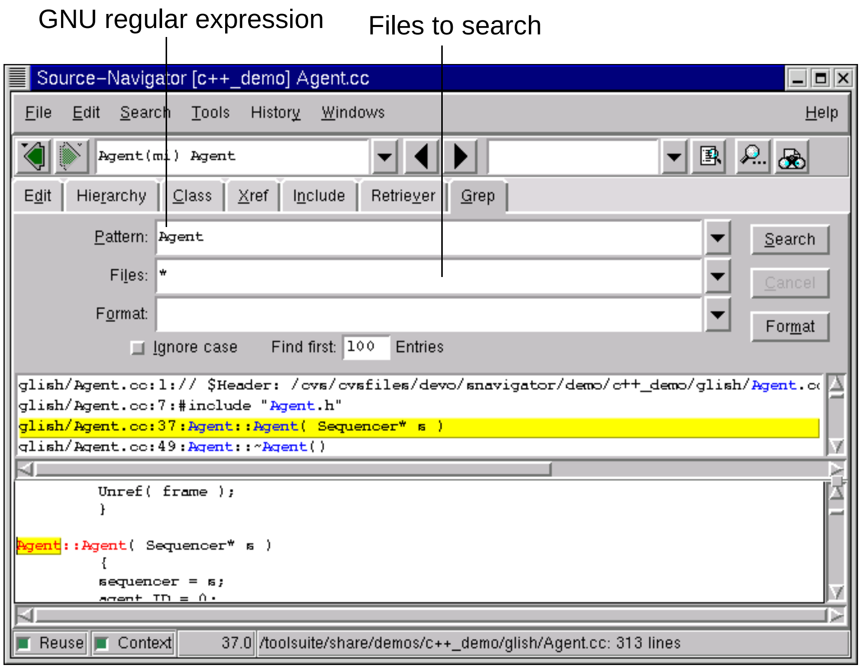
Figure 62: The Grep/Editor Window



In the **Pattern** text box, enter a regular expression then click the **Search** icon. For more information on regular expressions, see “GNU Regular Expressions” on page 85.

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 Table 1 on page 20. Figure 63 shows the results of a sample **Grep** search.

Figure 63: Sample Grep Search Results


GNU regular expression
Files to search
Source–Navigator [c++_demo] Agent.cc
File Edit Search Tools History Windows Help
Agent(mi) Agent
Edit Hierarchy Class Xref Include Retriever Grep
Search
Cancel
Format
Reuse
Context

Pattern: Agent
Files: *
Format:
Ignore case   Find first: 100 Entries

```
glish/Agent.cc:1:// $Header: /cvs/cvsfiles/devo/snavigator/demo/c++_demo/glish/Agent.cc
glish/Agent.cc:7:#include "Agent.h"
glish/Agent.cc:37:Agent::Agent( Sequencer* s )
glish/Agent.cc:49:Agent::~Agent()
```

```
        Unref( frame );
        }

Agent::Agent( Sequencer* s )
        {
        sequencer = s;
        agent_ID = 0;
```

37.0 /toolsuite/share/demos/c++_demo/glish/Agent.cc: 313 lines

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.

[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.

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 zero (0) and nine (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 **[^**.

Table 3 lists special characters and examples of how to use them.

Table 3: 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**, and **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.

Table 4: 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 operators:

Table 5: 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**, and **foo**)

Table 5: Interval Expressions (Continued)

+ (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,}	preceding 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.

Table 6: Escape Sequences

Symbol	*Description*
\\	a literal backslash
\a	alert
\b	backspace
\e	escape character
\f	form feed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\?	question mark
\(	left parenthesis

Table 6: Escape Sequences (Continued)

\)	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-b][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].

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

12

Version Control Systems

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.

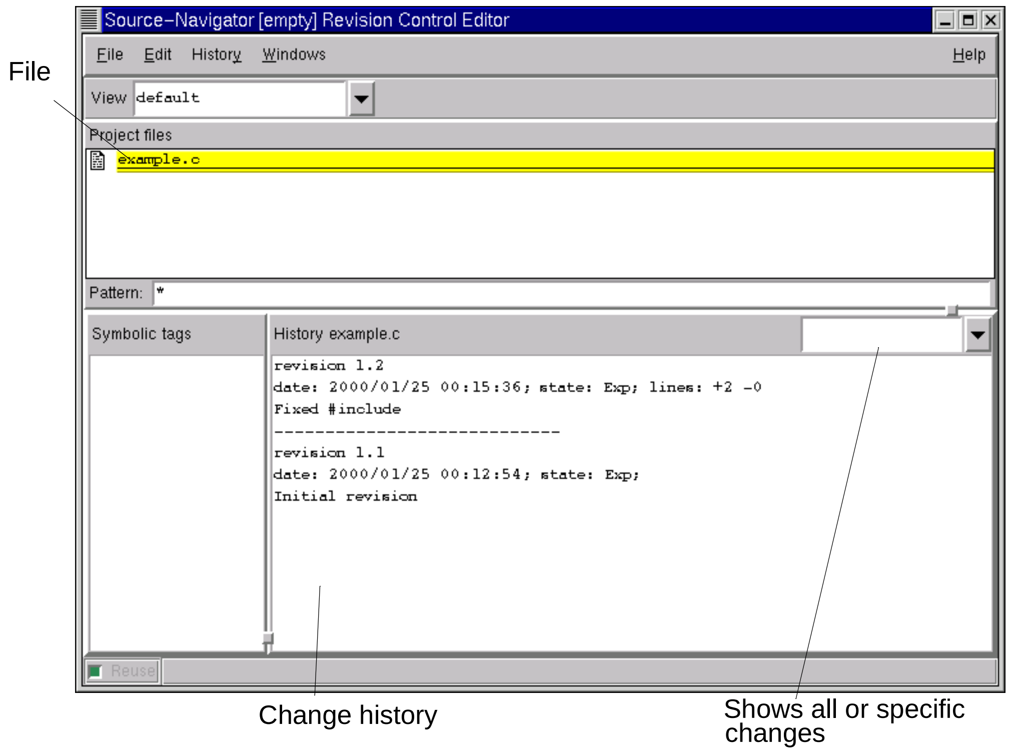
Source-Navigator provides a GUI to several external version control systems. With version control you can organize your development to manage versions, version history, labels, and related documents.

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 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 page 94).

To open the **Revision Control Editor**, from the **Tools** menu, select **Revision Control** → **Revision Control Editor** (see Figure 64).

Figure 64: Revision Version Control Window



View: default

example.c

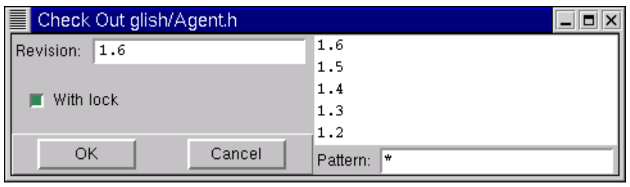
Pattern: *

```
revision 1.2
date: 2000/01/25 00:15:36; state: Exp; lines: +2 -0
Fixed #include
----------------------------
revision 1.1
date: 2000/01/25 00:12:54; state: Exp;
Initial revision
```

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.

Figure 65: Check Out Dialog Box



Revision: 1.6

With lock

```
1.6
1.5
1.4
1.3
1.2
```

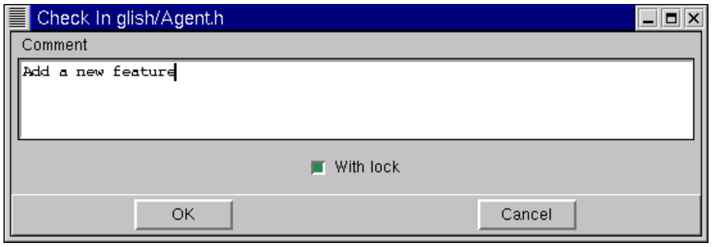
Pattern: *

Checking out with **With lock** selected in the **Check Out** dialog box prevents other users from checking out the same version in write mode.

Checking In a File

You can check in all or selected project files into your version control system. When you check in, you may 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.

Figure 66: Check In Dialog Box



Add a new feature

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 others the opportunity to make modifications to that file.

Discarding Changes to a File

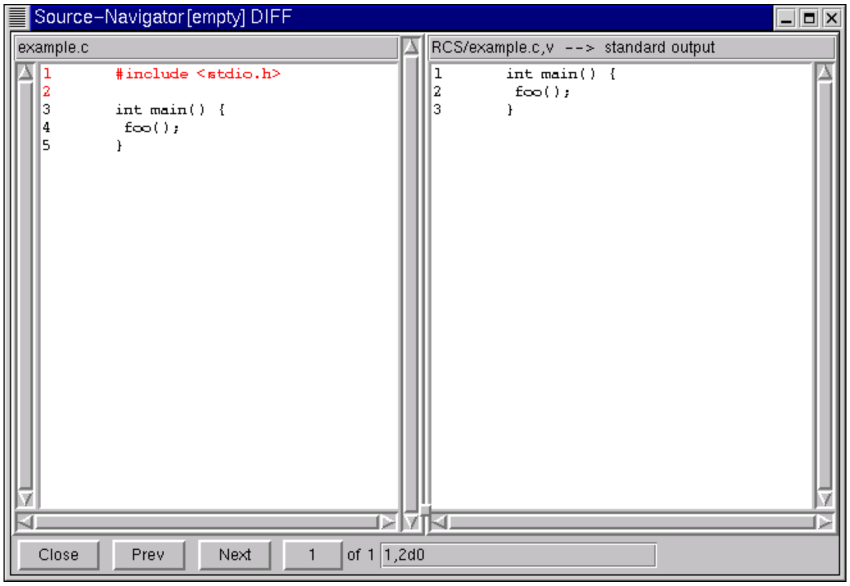
To revert working files to the repository version, in the **Revision Control Editor** window, from the **Edit** menu, select **Discard Changes**.

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 **Diff**, from the **Edit** menu, select **Compare Revisions**.

Figure 67: Showing Differences


Source–Navigator [empty] DIFF
example.c
RCS/example.c,v --> standard output
Close
Prev
Next
1
of 1
1,2d0

```
#include <stdio.h>

int main() {
 foo();
}
```

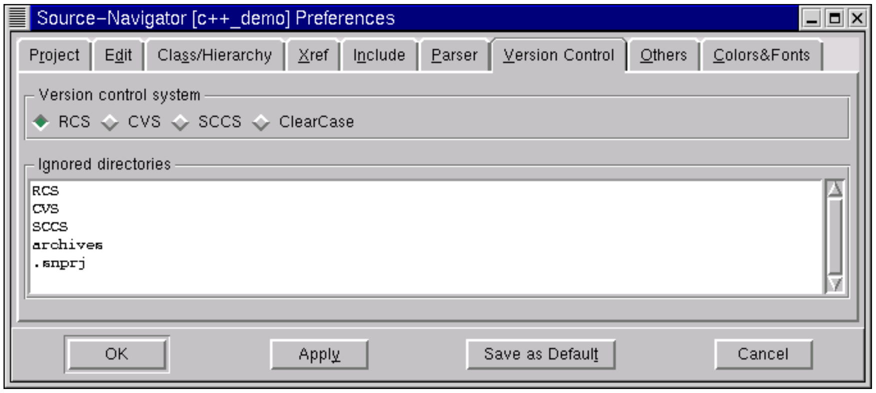
```
int main() {
 foo();
}
```

Version Control Preferences

Preference settings for the **Version Control** window are located in the **Version Control** tab of the **Preferences** dialog.

1. In the **Symbol Browser**, from the **File** menu select **Project Preferences**. In the **Version Control** window, from the **Edit** menu, select **View Preferences**.
2. Choose the **Version Control** tab.

Figure 68: Version Control Tab of the Preferences Dialog



Version control system

[✓ RCS] CVS SCCS ClearCase

Ignored directories

```
RCS
CVS
SCCS
archives
.snprj
```

Version control system

Select your external version control system for the project.

Ignored directories

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

For more information on integrating version control packages, see the "Integrating with Version Control Systems" on page 195.

13

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.

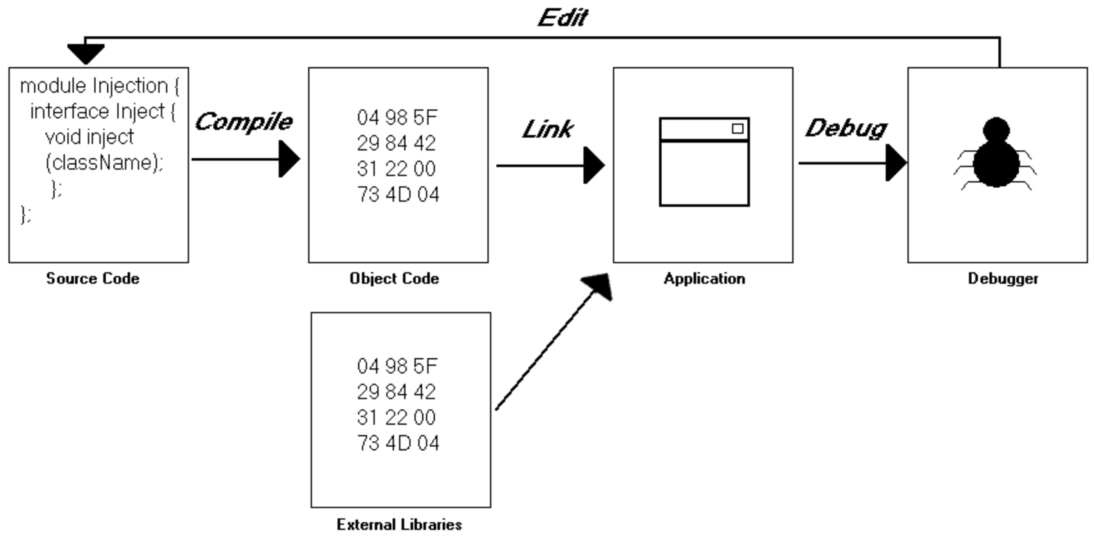
NOTE The compiler, `make`, and the Insight debugger must be installed on your machine before using these features.

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.

Figure 69: 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, optimization flags, and included paths.

You can edit your code using Source-Navigator. This chapter explains compiling and linking. For information about the Insight debugger, see “Working with Insight, the Debugger Interface” in the ***Getting Started Guide***.

`make`

Source-Navigator uses a utility called GNU `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 to be execute in order to build your program.

Build Targets

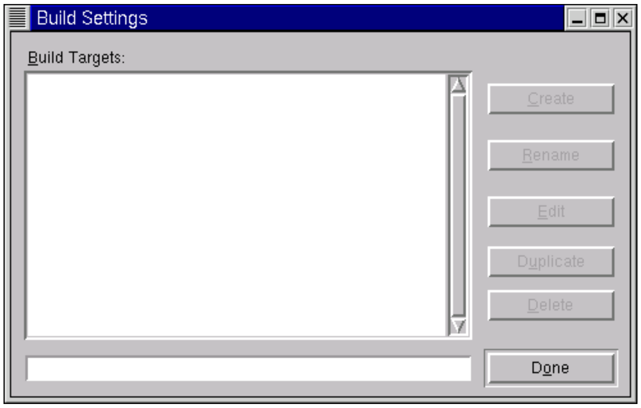
A build target is a conceptual object that contains information needed to compile and link a project. For example, `hello.c` converts into `hello.o` before producing the `hello` executable. The `hello.o` object file is linked with required libraries.

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.

Figure 70: Build Settings Dialog



Type 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” on page 100.

Modifying Build Targets

Rename a build target by selecting the target, typing 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.

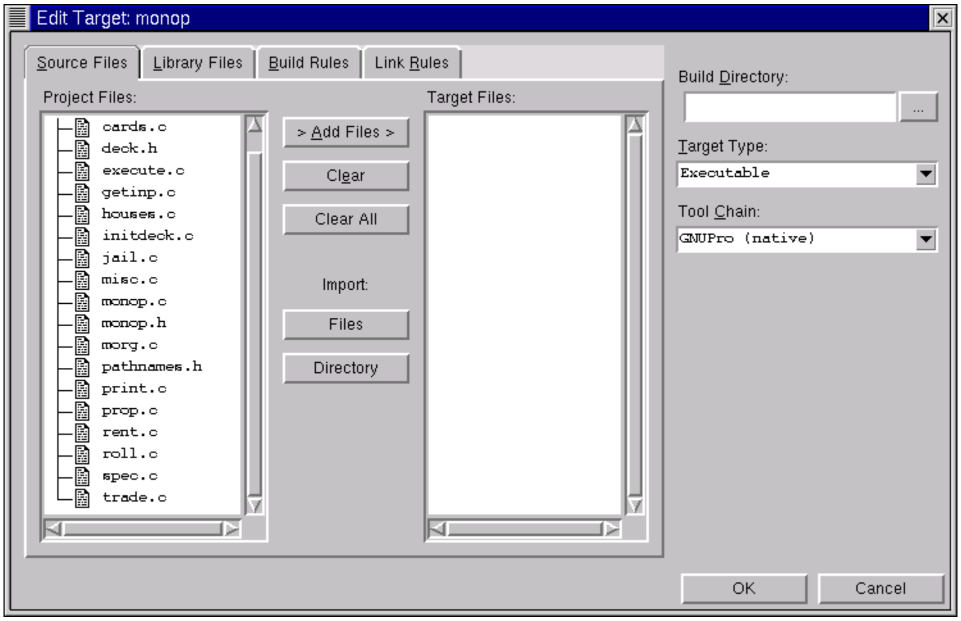
Editing a Target

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.

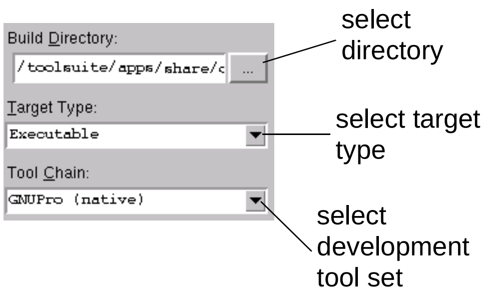
Figure 71: Edit Target Dialog



Edit Target tabs

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

Figure 72: Build Directory, Target Type, and Tool Chain Combo-boxes

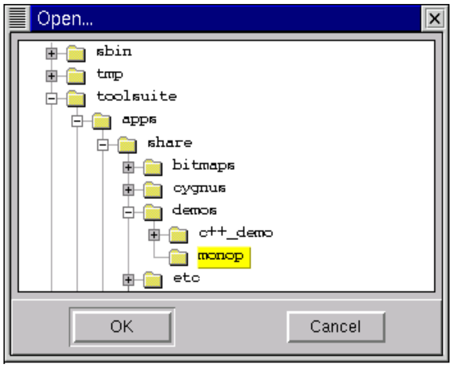


These store information about the build target.

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.

Figure 73: Build Directory Open Dialog



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*.

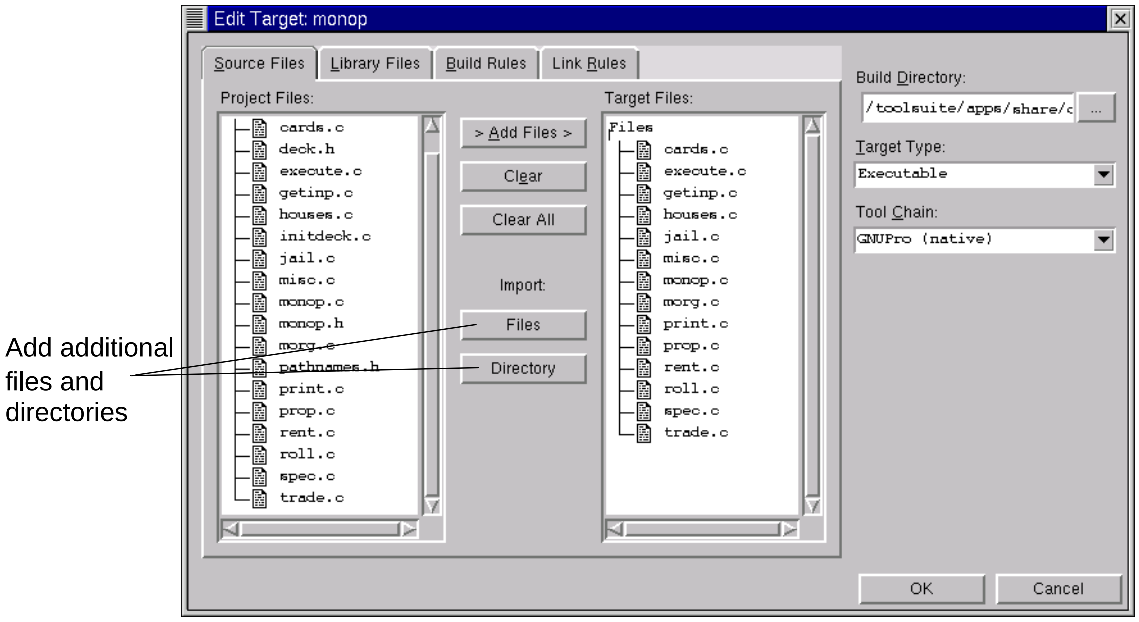
Tool Chain

A *tool chain* is a set of compilers, debuggers, and linkers. The *GNUPro (native)* option is always available. Other toolchains may be available, depending upon the Source-Navigator package purchased.

Source Files tab

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

Figure 74: Source Files Tab



Project Files:
cards.c
deck.h
execute.c
getinp.c
houses.c
initdeck.c
jail.c
misc.c
monop.c
monop.h
morg.c
pathnames.h
print.c
prop.c
rent.c
roll.c
spec.c
trade.c

Target Files:
Files
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
trade.c

Build Directory: /toolsuite/apps/share/c

Target Type: Executable

Tool Chain: GNUPro (native)

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

Importing files and directories

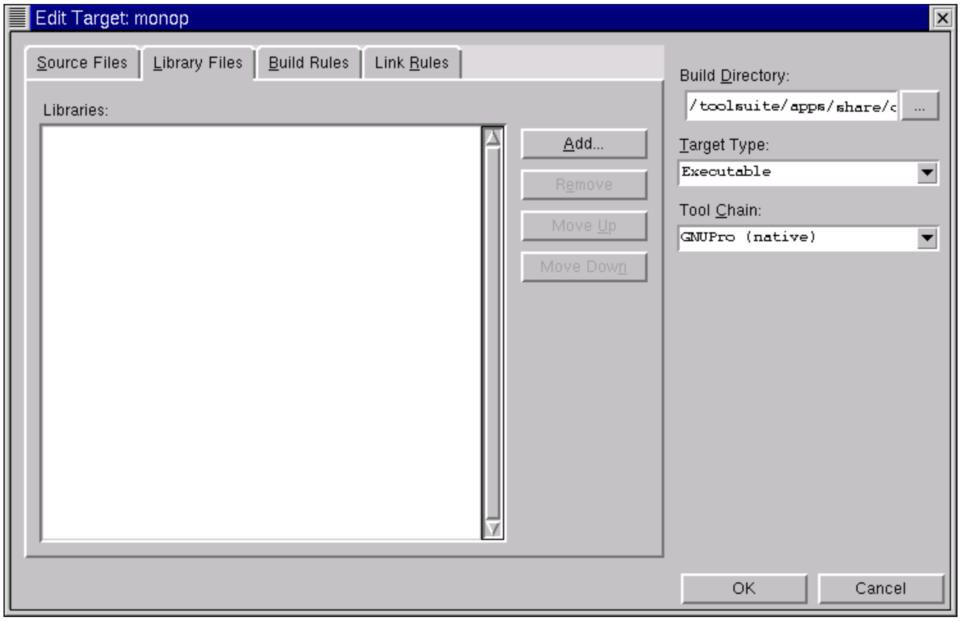
To import files or directories into your currently active project:

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

Figure 75: 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 **Libraries** list, select the library and click the **Remove** button. The library is removed from the build.

Libraries are linked in the order listed in the **Libraries** list. 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

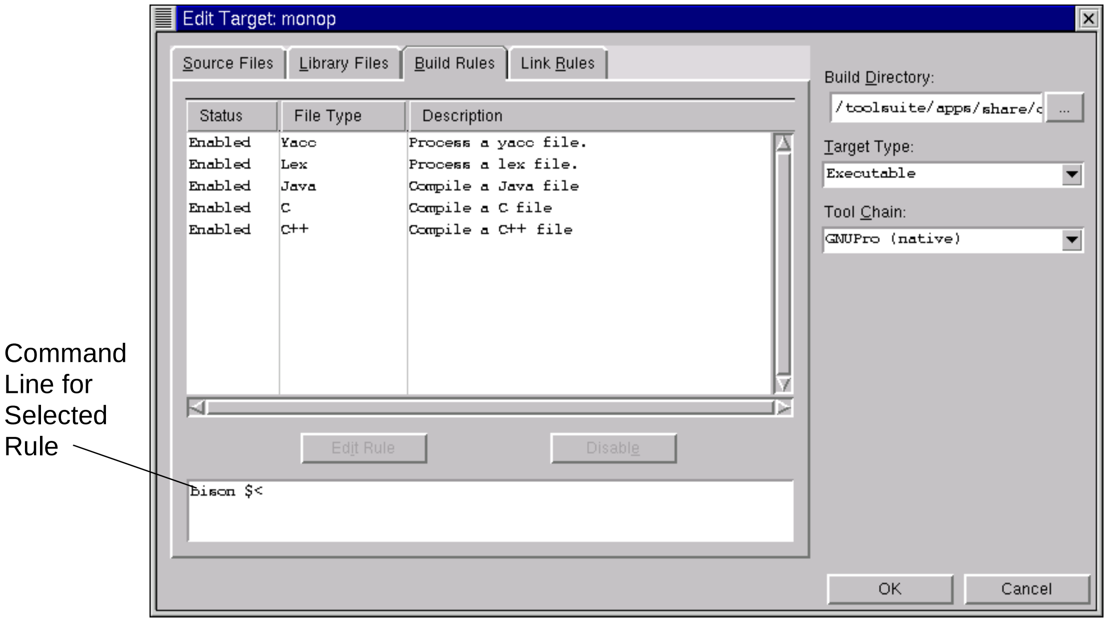
Click the **Build Rules** tab to configure each rule in the build target.

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.

Figure 76: Build Rules Tab



Status	File Type	Description
Enabled	Yacc	Process a yacc file.
Enabled	Lex	Process a lex file.
Enabled	Java	Compile a Java file
Enabled	C	Compile a C file
Enabled	C++	Compile a C++ file

```
Bison $<
```

Build Directory: /toolsuite/apps/share/c

Target Type: Executable

Tool Chain: GNUPro (native)

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, either

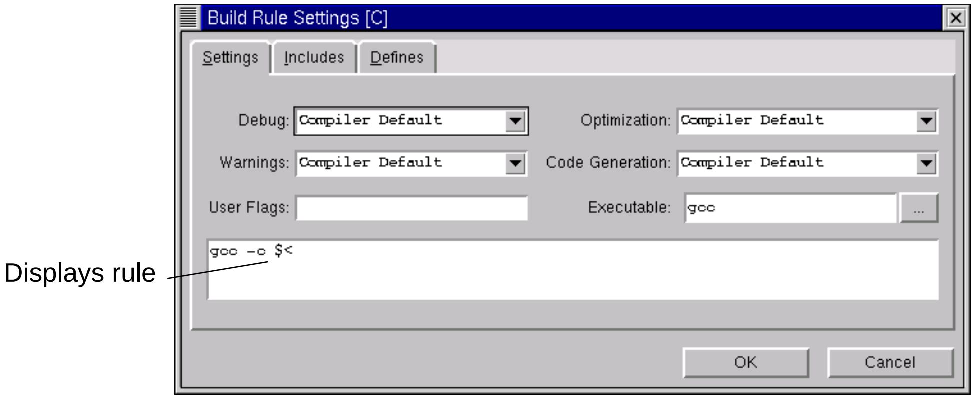
- select the rule from the rule listing and click the **Edit Rule** button, or
- double-click the rule.

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.

Figure 77: Build Rule Settings with Options Selected



Debug: Compiler Default
Optimization: Compiler Default
Warnings: Compiler Default
Code Generation: Compiler Default
User Flags:
Executable: gcc

```
gcc -c $<
```

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” on page 106.

Optimization
: Compiler optimization for the code.

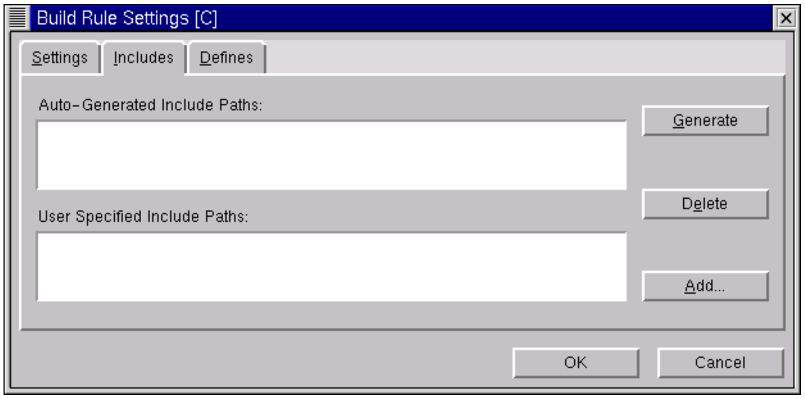
Code Generation
: Processor- or code control-specific optimizations of settings.

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.

Figure 78: Includes Tab



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

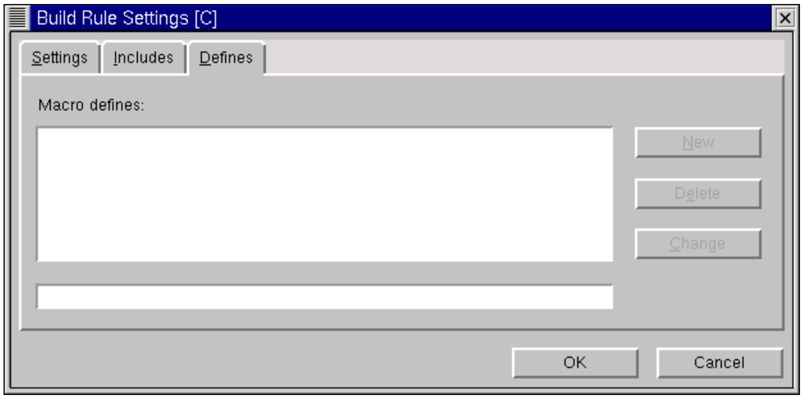
Click the **Generate** button to generate a list of Source-Navigator included paths. These appear in the **Auto-Generated Include Paths** list.

To add additional paths, click the **Add** button. The selected paths appear in the **User Specified Include Paths** list. 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.

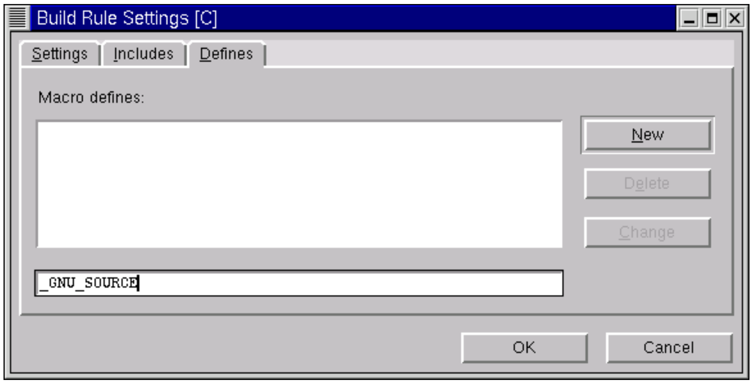
Figure 79: Defines Tab



Modifying macro definitions

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

Figure 80: New Macro Created

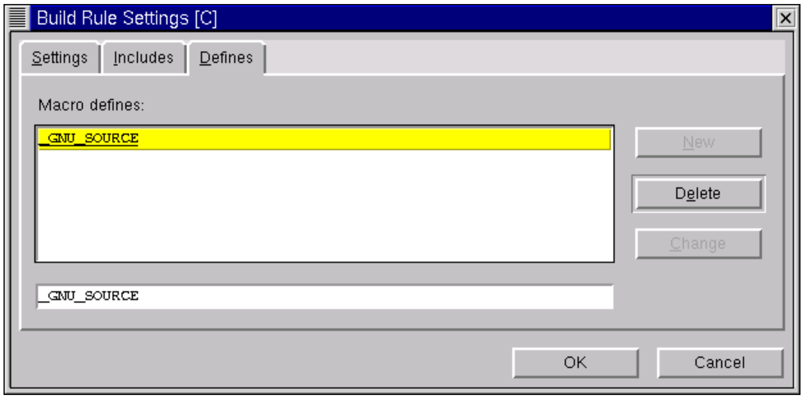


Macro defines:

_GNU_SOURCE

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

Figure 81: Macro Created and Selected



Macro defines:

_GNU_SOURCE

_GNU_SOURCE

Make the necessary modifications to the macro.

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

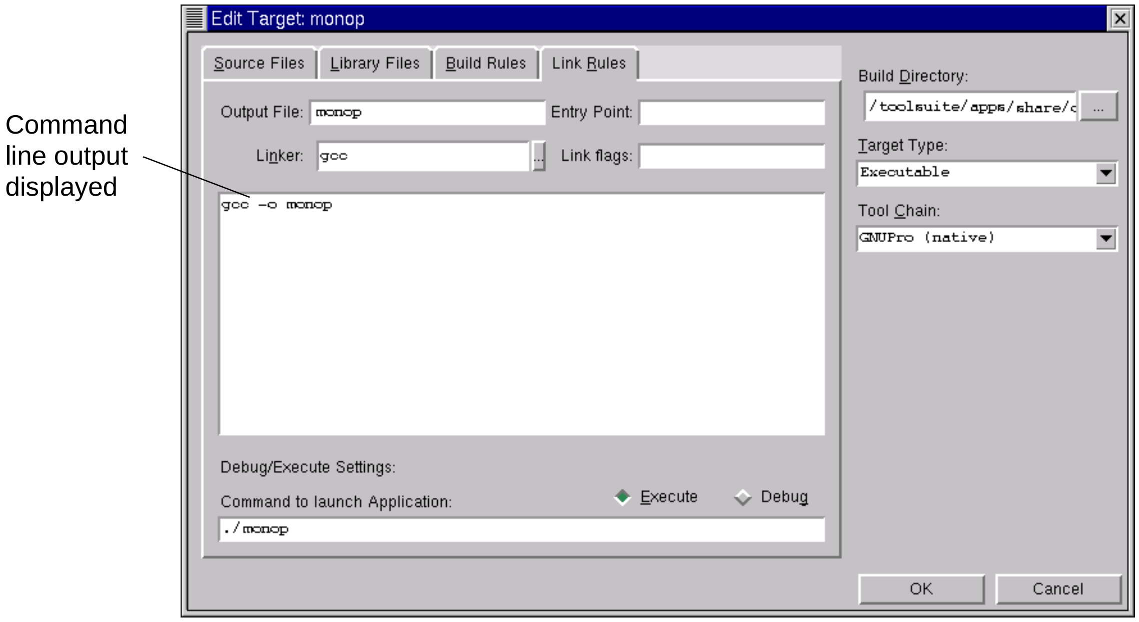
To delete a macro, select it from the **Macro defines** list and click the **Delete** button. The macro is removed from the listing.

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

Figure 82: Link Rules Tab of the Build Targets Menu



Edit Target: monop

Output File: monop

Entry Point:

Linker: gcc

Link flags:

```
gcc -o monop
```

Command to launch Application: ./monop

[✓ Execute] Debug

Build Directory: /toolsuite/apps/share/c

Target Type: Executable

Tool Chain: GNUPro (native)

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” on page 113 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 must also change the name here.

On UNIX, type `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.

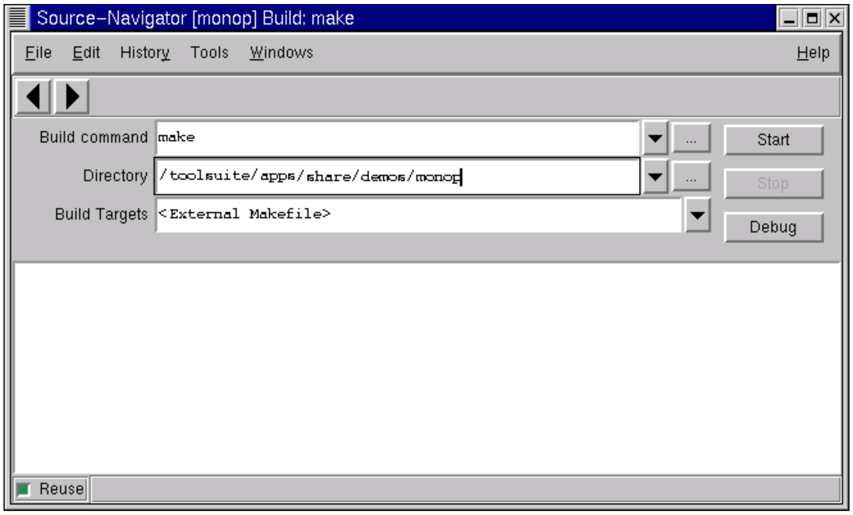
Source-Navigator can generate its own `makefiles` or work with one that you supply.

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.

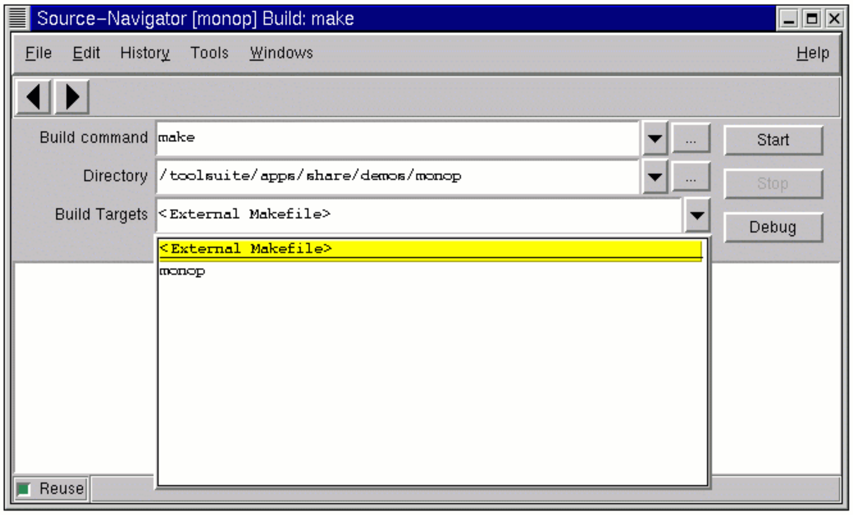
Figure 83: Build Window



Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: <External Makefile>

3. Select the build target from the **Build Target** list.

Figure 84: Selecting a Build Target



```
make
/toolsuite/apps/share/demos/monop
<External Makefile>
<External Makefile>
monop
```

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” on page 111.

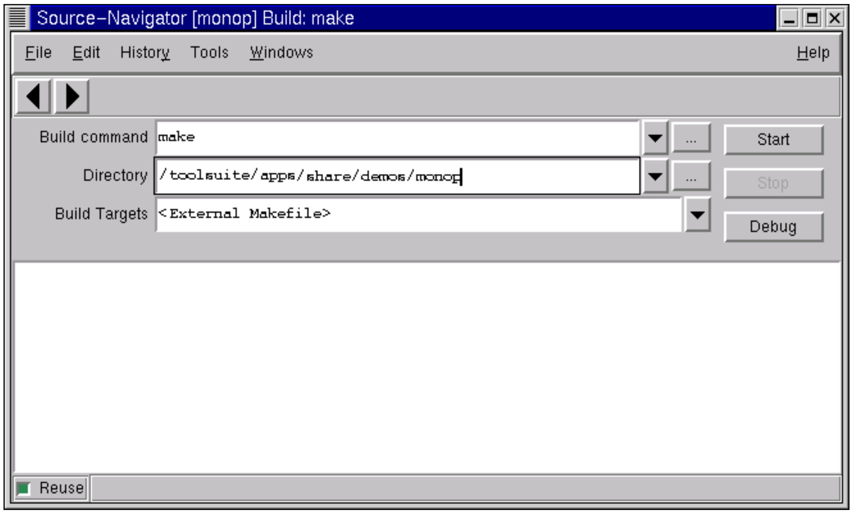
   If no errors appear, the build target is compiled and ready to be executed. Click the **Run** button to execute the application.

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>`.

Figure 85: Selecting the <External Makefile> Build Target



Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: <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 Figure 21 on page 31).
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.

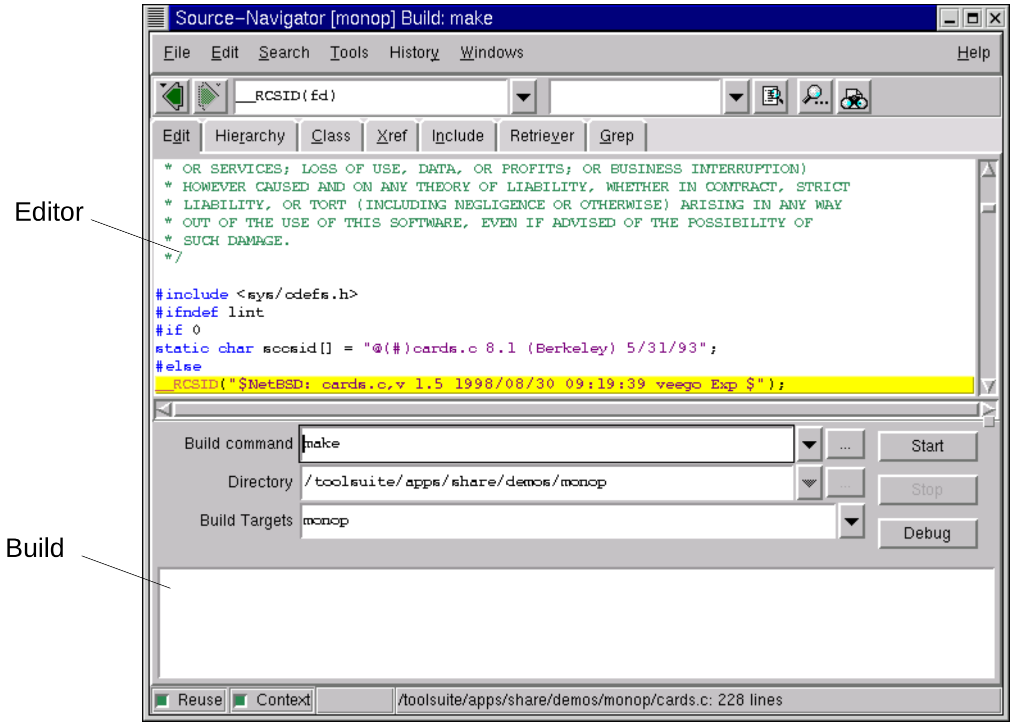
Modifying the Build

If errors appear in the **Build** window, the source code must be modified before the program will run.

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.

Figure 86: Editor-Build Window



```
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <sys/cdefs.h>
#ifndef lint
#if 0
static char sccsid[] = "@(#)cards.c 8.1 (Berkeley) 5/31/93";
#else
__RCSID("$NetBSD: cards.c,v 1.5 1998/08/30 09:19:39 veego Exp $");
```

__RCSID(fd)

Build command: make

Directory: /toolsuite/apps/share/demos/monop

Build Targets: monop

/toolsuite/apps/share/demos/monop/cards.c: 228 lines

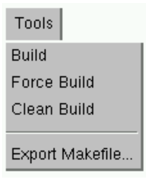
To save and recompile the modified files, press the **Start** button. The **Fast Save** dialog appears asking if you want to perform a fast save. Click the **Yes** button to save the changes and rebuild. Click the **No** button to discard changes and perform a rebuild. Clicking the **Cancel** button 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.

Figure 87: 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. This is equivalent to the command `make clean`.

Export Makefile
: Saves your build target as a `makefile` for external use.

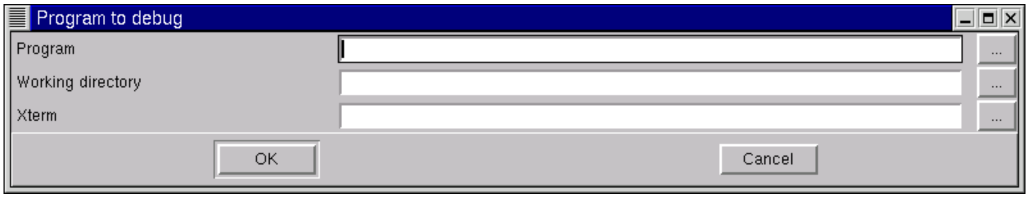
Debugging the build target

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.

Figure 88: Program to Debug Dialog



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

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

   On UNIX, in the **Xterm** field, type `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” on page 31), is used.
3. Click **OK** to debug the build target.

   The debugger launches.

   For more information on the debugger, see “Working with Insight, the Debugger

Interface” in the ***Getting Started Guide***.

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. Under **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” on page 109).

Executing the application

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

Build Tutorial

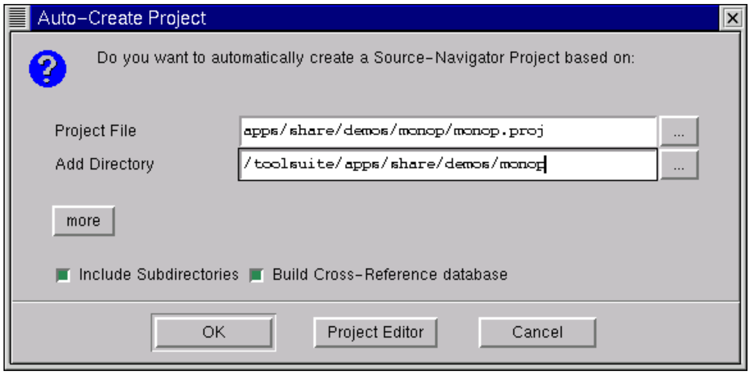
This build tutorial is based on the `monop` demo, located in the `demos` directory. `monop` is a command line based game that you can build, edit, and play. In this tutorial, you will create two build targets and use many build and debugging features of Source-Navigator.

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.

Figure 89: Creating the monop Project

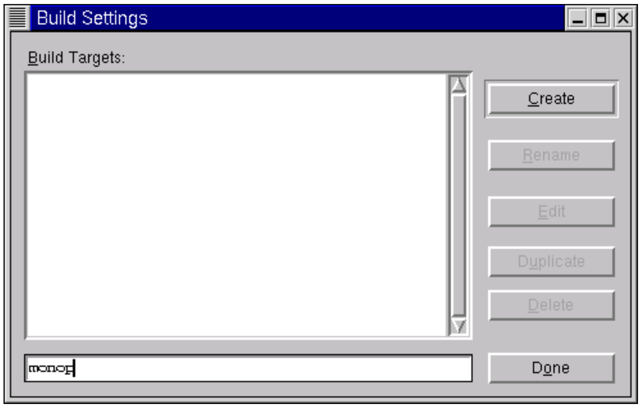

Auto–Create Project
Do you want to automatically create a Source–Navigator Project based on:
Project File
apps/share/demos/monop/monop.proj
Add Directory
/toolsuite/apps/share/demos/monop
...
more
Include Subdirectories
Build Cross–Reference database
OK
Project Editor
Cancel

4. Click **OK** to create the project.

Creating the monop Target

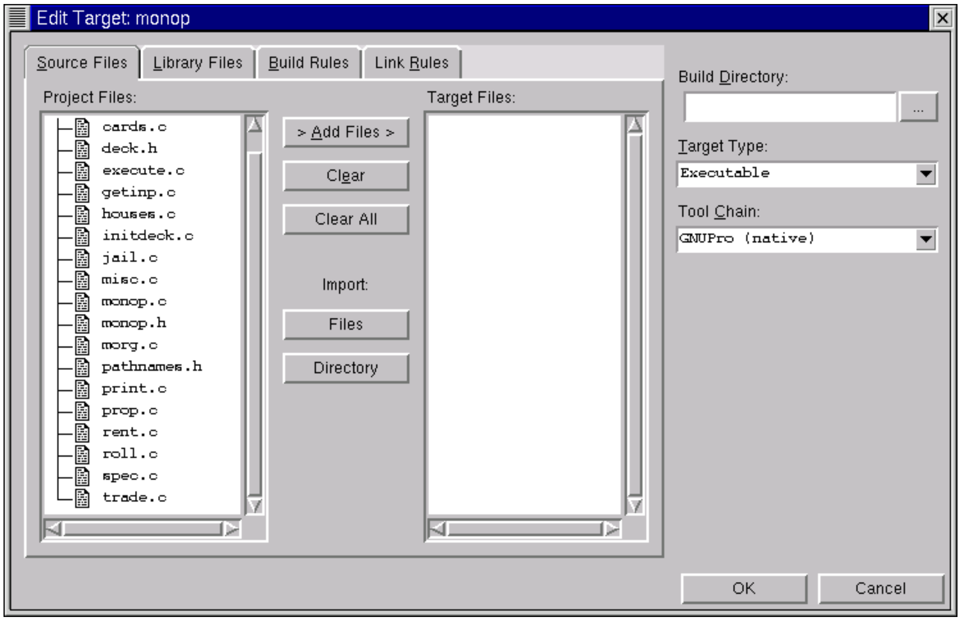
1. From the **Tools** menu, select **Build Settings**. The **Build Settings** dialog opens.
2. Type `monop` as the name of the build target.

Figure 90: Creating the monop Build Target



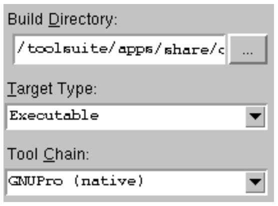
3. Click the **Create** button. The **Edit Target** dialog opens.

Figure 91: Edit Target Dialog



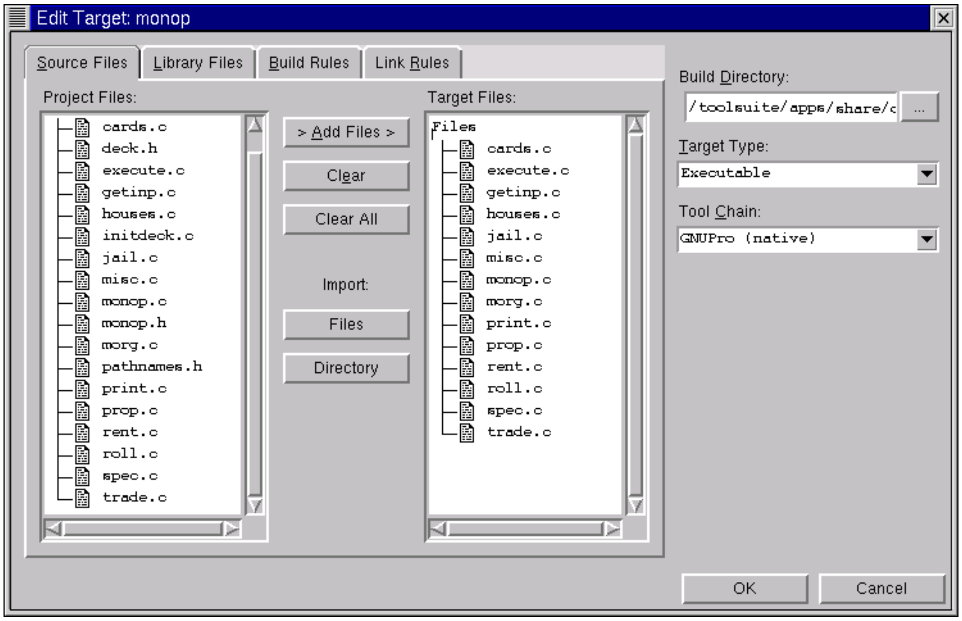
4. In the **Build Directory** field, click the “**...**” button and select the build directory for the `monop` project.

Figure 92: Adding the Build Directory



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.

Figure 93: Files Added to the Target



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

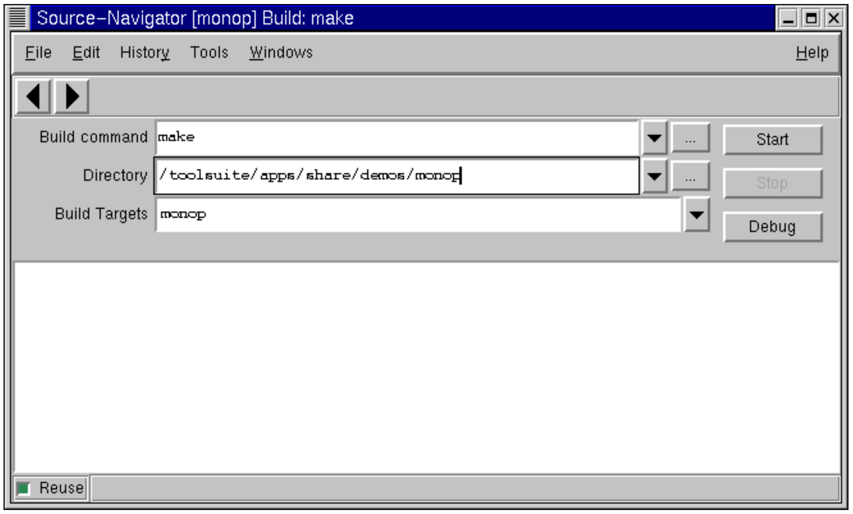
7. Click **OK** to close the **Edit Target** dialog.
8. Click the **Done** button to close the **Build Settings** dialog.

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`.

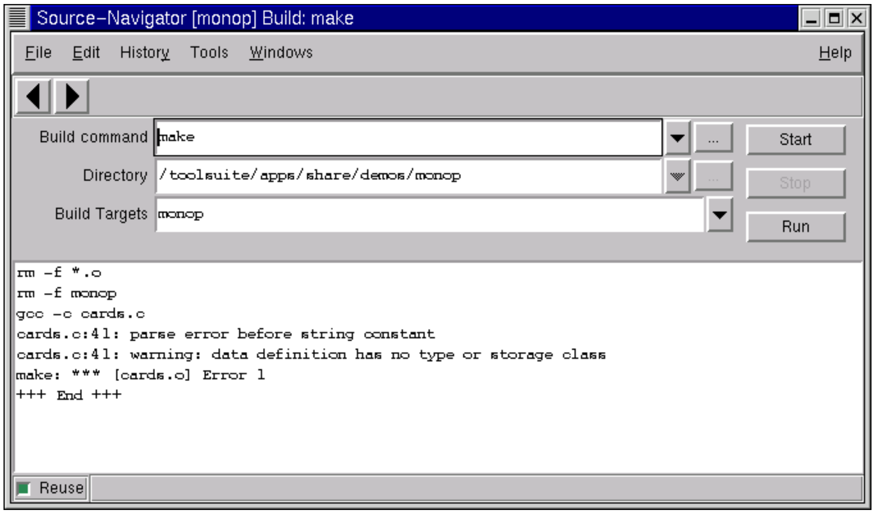
Figure 94: Selecting the monop Build Target


Source–Navigator [monop] Build: make
File Edit History Tools Windows Help
Start
Stop
Debug
Reuse

Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: monop

3. Click the **Start** button.

Figure 95: Building monop


Source–Navigator [monop] Build: make
File Edit History Tools Windows Help
Start
Stop
Run
Reuse

Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: monop

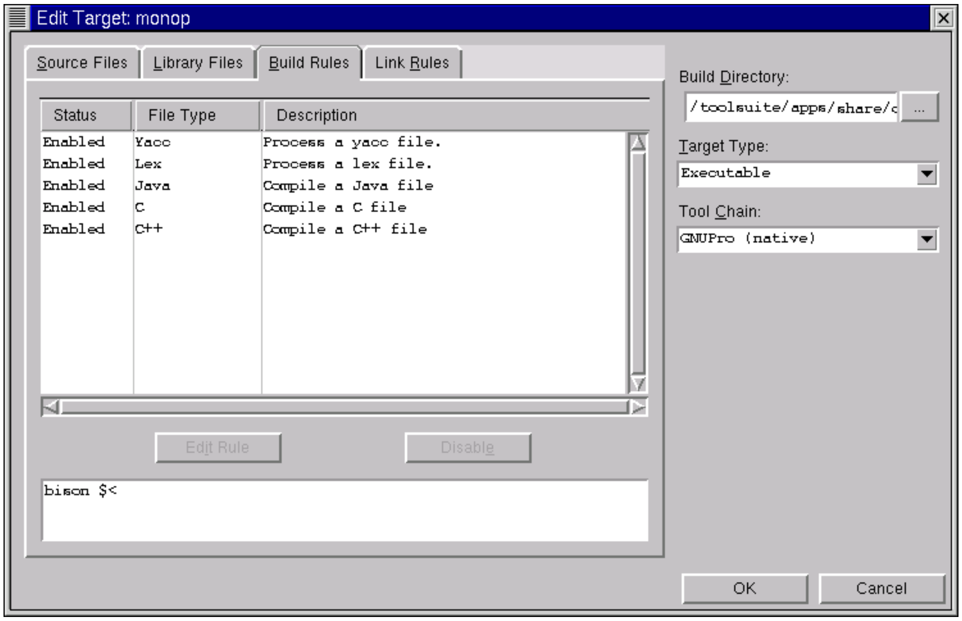
```
rm -f *.o
rm -f monop
gcc -c cards.c
cards.c:41: parse error before string constant
cards.c:41: warning: data definition has no type or storage class
make: *** [cards.o] Error 1
+++ End +++
```

Errors are generated 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.

Figure 96: Build Rules Tab



Status	File Type	Description
Enabled	Yacc	Process a yacc file.
Enabled	Lex	Process a lex file.
Enabled	Java	Compile a Java file
Enabled	C	Compile a C file
Enabled	C++	Compile a C++ file

```
bison $<
```

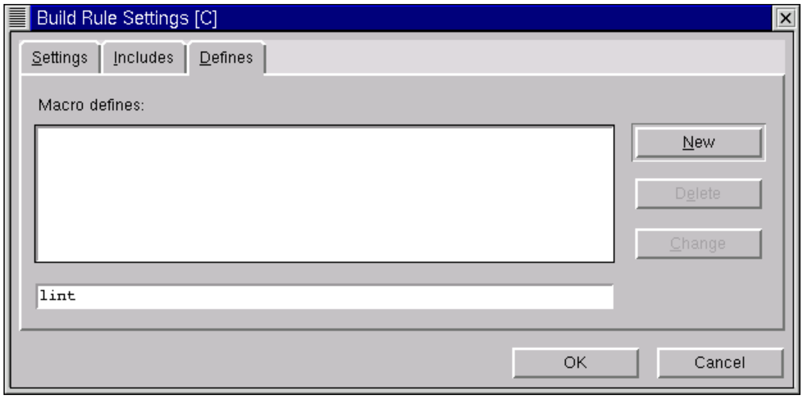
Build Directory: /toolsuite/apps/share/c

Target Type: Executable

Tool Chain: GNUPro (native)

4. Because `monop` is written in C, double-click the **C** rule. The **Build Rule Settings** dialog opens.
5. Click the **Defines** tab. Type `lint` in the text entry box.

Figure 97: Creating the lint Macro



Macro defines:

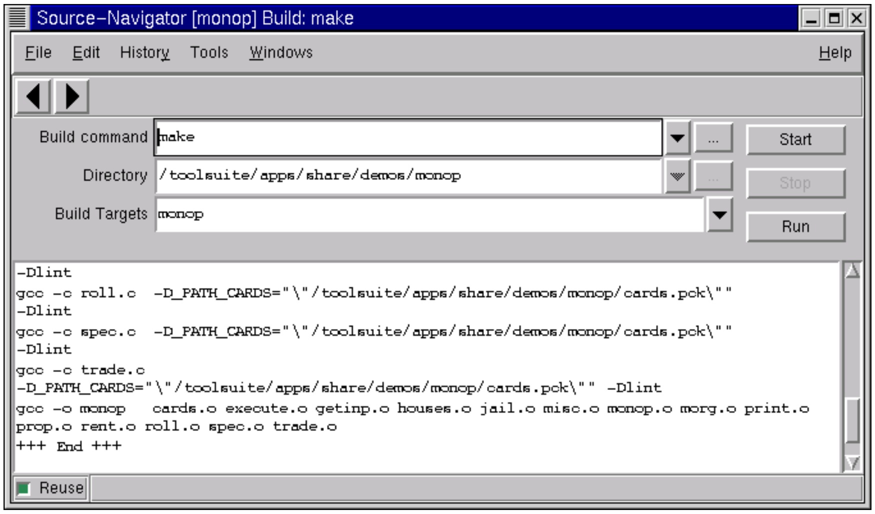
lint

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** combo-box, select `monop`.
3. Click the **Start** button.

Figure 98: Rebuilding the monop Build Target



Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: monop

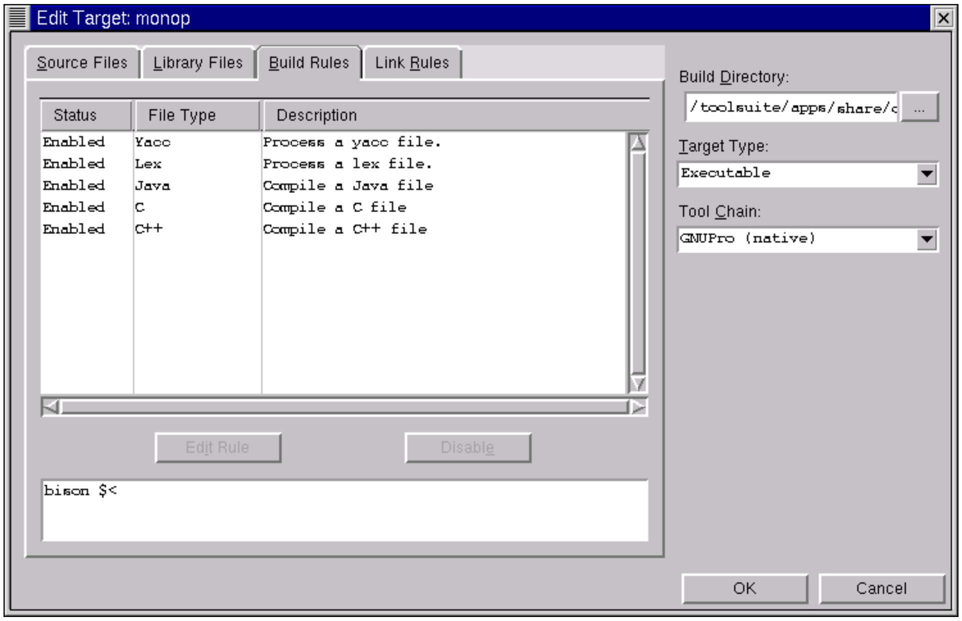
```
-Dlint
gcc -c roll.c  -D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -c spec.c  -D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -c trade.c
-D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\"" -Dlint
gcc -o monop   cards.o execute.o getinp.o houses.o jail.o misc.o monop.o morg.o print.o
prop.o rent.o roll.o spec.o trade.o
+++ End +++
```

The build 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.

Figure 99: Build Rules Tab


Edit Target: monop

Source Files | Library Files | Build Rules | Link Rules

Status	File Type	Description
Enabled	Yacc	Process a yacc file.
Enabled	Lex	Process a lex file.
Enabled	Java	Compile a Java file
Enabled	C	Compile a C file
Enabled	C++	Compile a C++ file

Edit Rule   Disable

```
bison $<
```

Build Directory: /toolsuite/apps/share/c

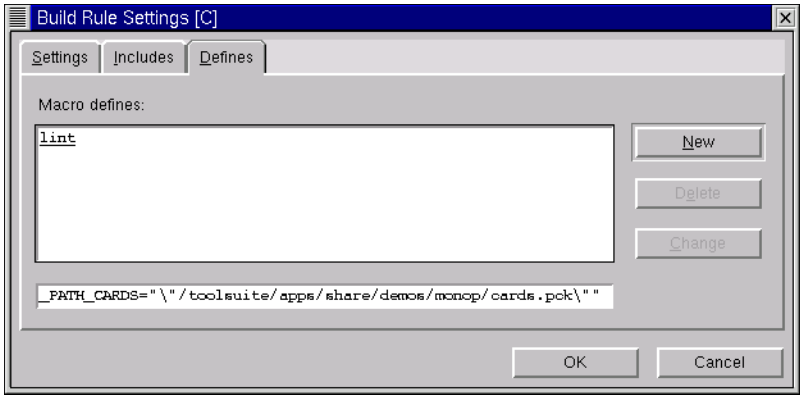
Target Type: Executable

Tool Chain: GNUPro (native)

OK   Cancel

4. Double-click the **C** rule. The **Build Rule Settings** dialog opens.
5. Click the **Defines** tab.
6. Type 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\""`

Figure 100: Creating the _PATH_CARDS Macro


Build Rule Settings [C]

Settings | Includes | Defines

Macro defines:

```
lint
```

New   Delete   Change

```
_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
```

OK   Cancel

This tells Source-Navigator which card pack to use when running the `monop` program.

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

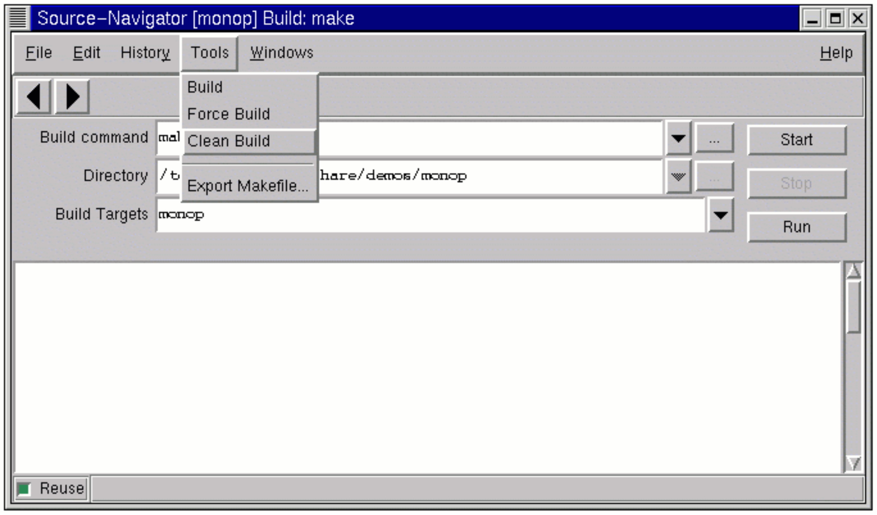
8. Click **OK** to close the **Build Rules Settings** dialog.
9. Click **OK** to close the **Edit Target** dialog.
10. Click the **Done** button to close the **Build Settings** dialog.

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**.

Figure 101: Selecting Clean Build



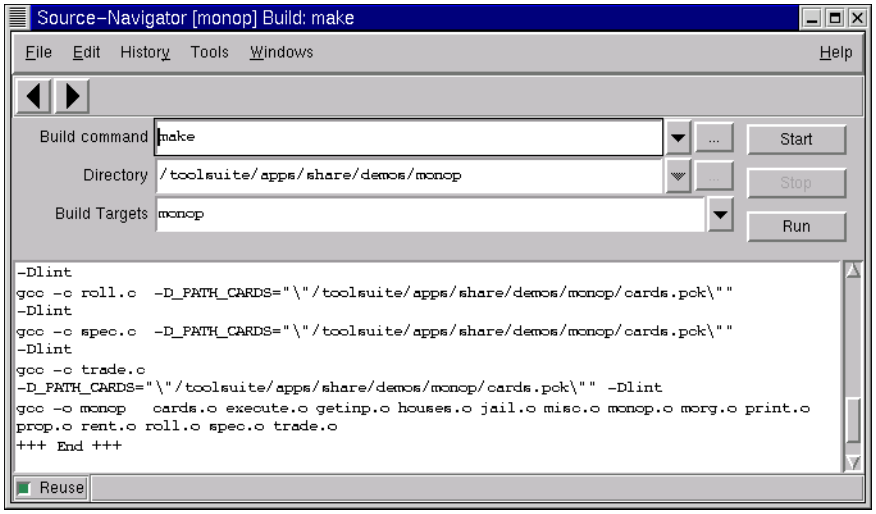
Build command: mal

Directory: /t … hare/demos/monop

Build Targets: monop

2. Click the **Start** button to perform the build.

Figure 102: Performing a Clean Build


Source–Navigator [monop] Build: make
File Edit History Tools Windows Help
Start
Stop
Run
Reuse

Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: monop

```
-Dlint
gcc -c roll.c  -D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -c spec.c  -D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -c trade.c
-D_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\"" -Dlint
gcc -o monop   cards.o execute.o getinp.o houses.o jail.o misc.o monop.o morg.o print.o
prop.o rent.o roll.o spec.o trade.o
+++ End +++
```

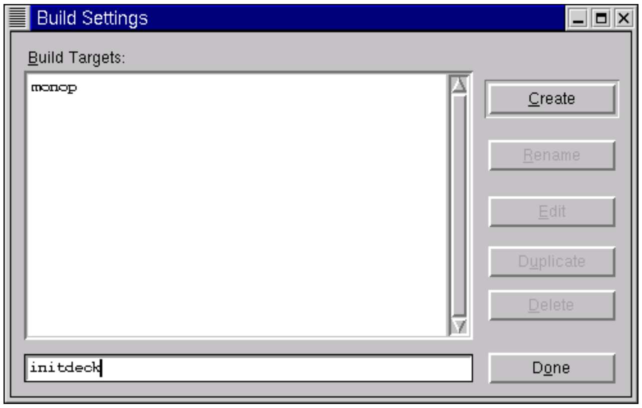
Again `monop` compiles without errors, but now uses the correct card pack.

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. Type `initdeck` as the name of the build target.

Figure 103: Creating the initdeck Target

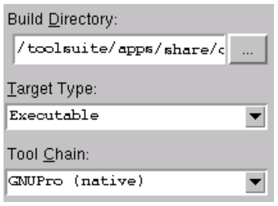


Build Targets: monop

initdeck

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. This is the same build directory used for the `monop` target.

Figure 104: Adding the Build Directory



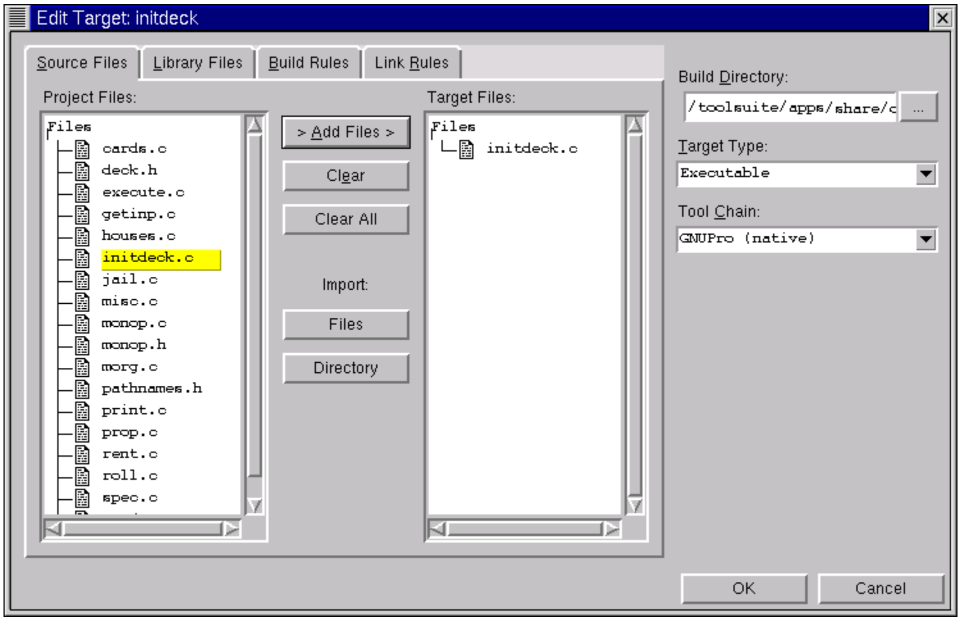
Build Directory:
/toolsuite/apps/share/c

Target Type:
Executable

Tool Chain:
GNUPro (native)

5. From **Project Files**, select the `initdeck.c` file.
6. Click the **Add Files** button to copy the file to the **Target Files** list.

Figure 105: Files Added to the Target



Edit Target: initdeck

Project Files:

```
Files
├─ cards.c
├─ deck.h
├─ execute.c
├─ getinp.c
├─ houses.c
├─ initdeck.c
├─ jail.c
├─ misc.c
├─ monop.c
├─ monop.h
├─ morg.c
├─ pathnames.h
├─ print.c
├─ prop.c
├─ rent.c
├─ roll.c
├─ spec.c
```

Target Files:

```
Files
└─ initdeck.c
```

Build Directory:
/toolsuite/apps/share/c

Target Type:
Executable

Tool Chain:
GNUPro (native)

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

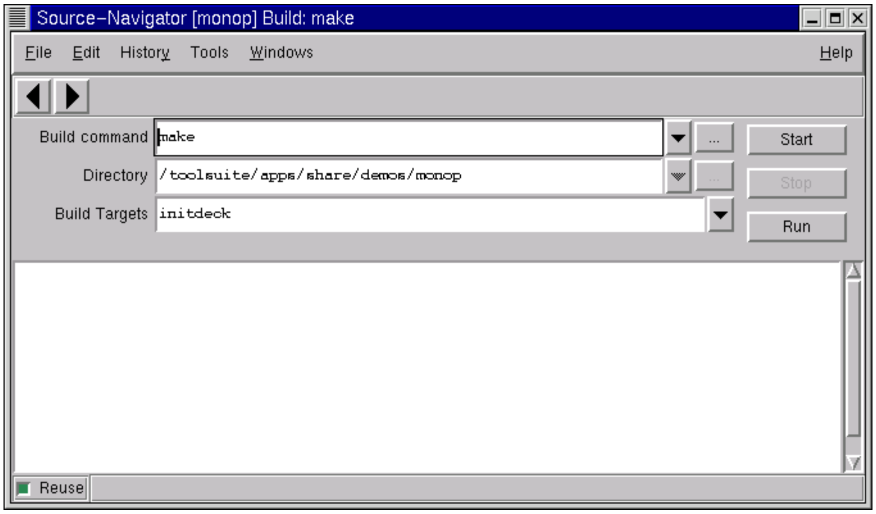
7. Click **OK** to close the **Edit Target** dialog.
8. Click the **Done** button to close the **Build Settings** dialog.

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** combo-box, select `initdeck`.

Figure 106: Selecting the initdeck Build Target


Source–Navigator [monop] Build: make
File Edit History Tools Windows Help
Start
Stop
Run
Reuse

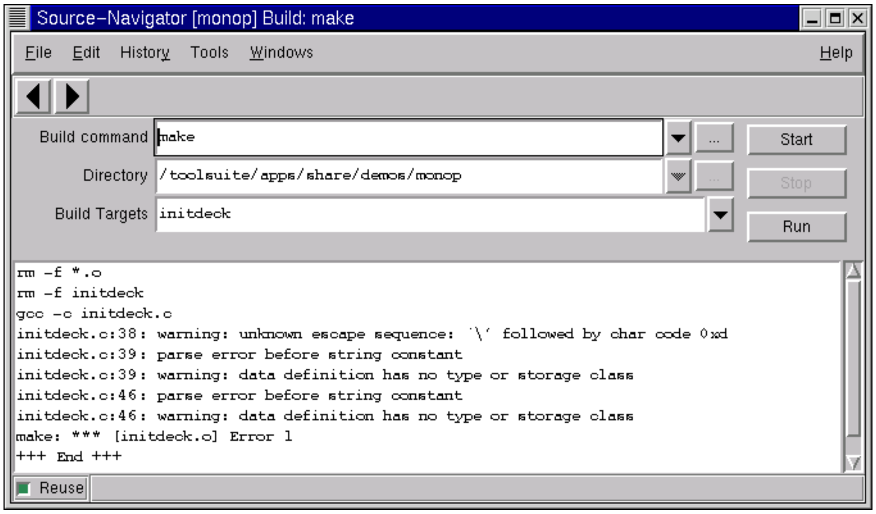
Build command: make

Directory: /toolsuite/apps/share/demos/monop

Build Targets: initdeck

3. Click the **Start** button.

Figure 107: Building initdeck


Source–Navigator [monop] Build: make
File Edit History Tools Windows Help
Start
Stop
Run
Reuse

Build command: make

Directory: /toolsuite/apps/share/demos/monop

Build Targets: initdeck

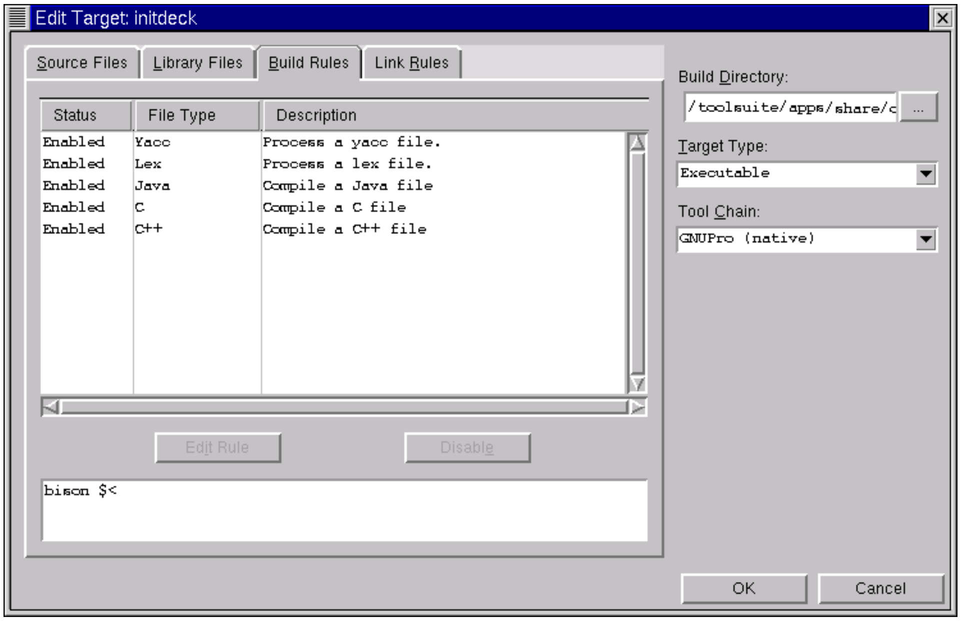
```
rm -f *.o
rm -f initdeck
gcc -c initdeck.c
initdeck.c:38: warning: unknown escape sequence: `\' followed by char code 0xd
initdeck.c:39: parse error before string constant
initdeck.c:39: warning: data definition has no type or storage class
initdeck.c:46: parse error before string constant
initdeck.c:46: warning: data definition has no type or storage class
make: *** [initdeck.o] Error 1
+++ End +++
```

Errors are generated 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.

Figure 108: Build Rules Tab



Status	File Type	Description
Enabled	Yacc	Process a yacc file.
Enabled	Lex	Process a lex file.
Enabled	Java	Compile a Java file
Enabled	C	Compile a C file
Enabled	C++	Compile a C++ file

```
bison $<
```

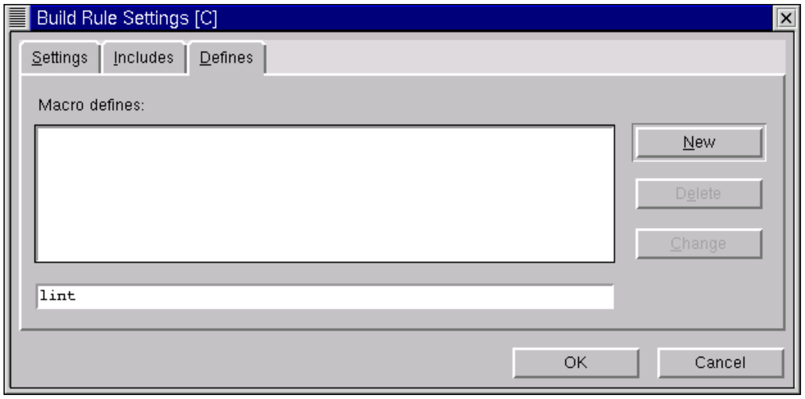
Build Directory: /toolsuite/apps/share/c

Target Type: Executable

Tool Chain: GNUPro (native)

4. Because `initdeck` is written in C, double-click the **C** rule. The **Build Rule Settings** dialog opens.
5. Click the **Defines** tab. Type `lint` in the text entry box.

Figure 109: Creating the lint Macro

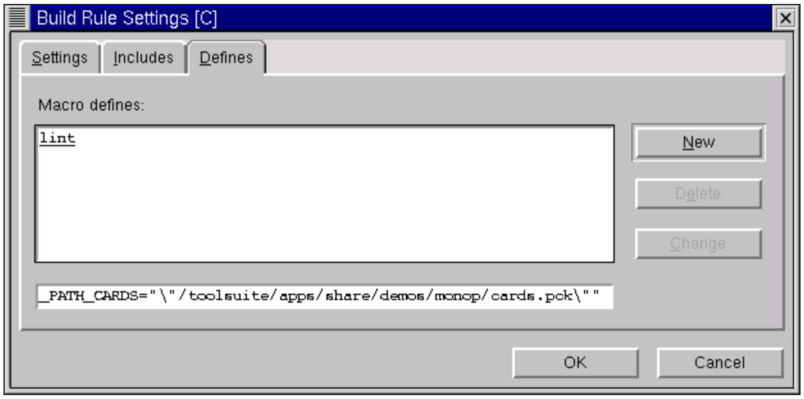


Macro defines:

```
lint
```

6. Click the **New** button to create the macro.
7. Type 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\""`

Figure 110: Creating the _PATH_CARDS Macro

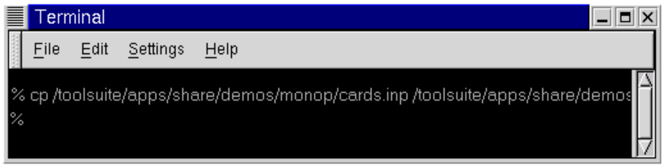


```
lint
_PATH_CARDS="\"/toolsuite/apps/share/demos/monop/cards.pck\""
```

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.
10. Click **OK** to close the **Edit Target** dialog.
11. Click the **Done** button to close the **Build Settings** dialog.
12. Open a console window and copy the `card.inp` file, located in the `demos/monop` directory, into the build directory.

Figure 111: Copying the card.inp File



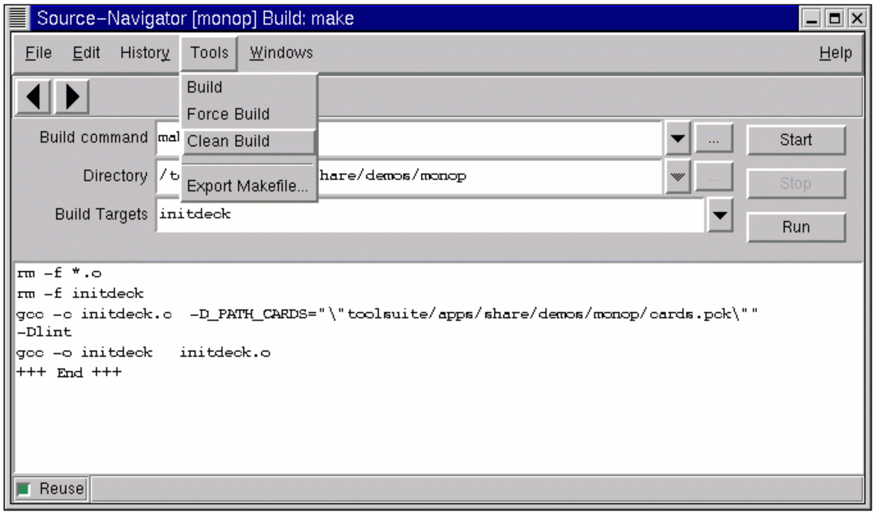
```
% cp /toolsuite/apps/share/demos/monop/cards.inp /toolsuite/apps/share/demos
%
```

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**.

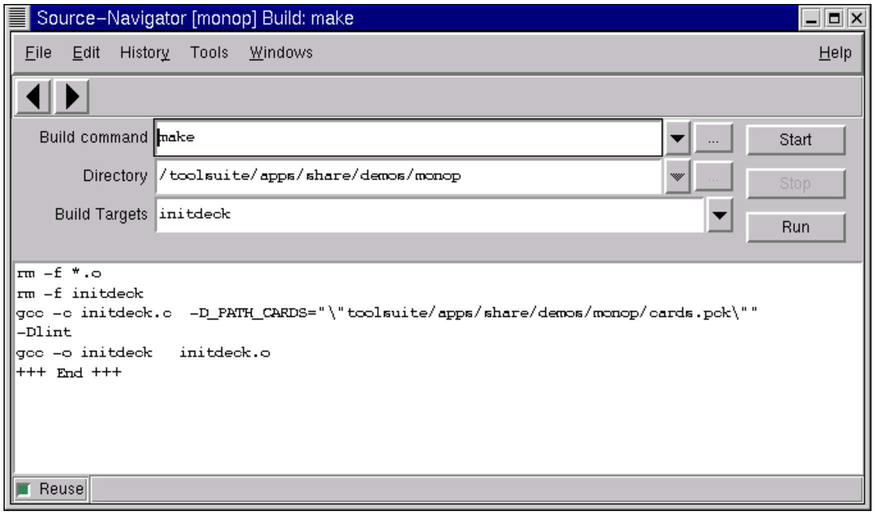
Figure 112: Selecting Clean Build



```
rm -f *.o
rm -f initdeck
gcc -c initdeck.c  -D_PATH_CARDS="\"toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -o initdeck   initdeck.o
+++ End +++
```

2. Click the **Start** button to perform the build.

Figure 113: Performing a Clean Build



Build command: make
Directory: /toolsuite/apps/share/demos/monop
Build Targets: initdeck

```
rm -f *.o
rm -f initdeck
gcc -c initdeck.c  -D_PATH_CARDS="\"toolsuite/apps/share/demos/monop/cards.pck\""
-Dlint
gcc -o initdeck   initdeck.o
+++ End +++
```

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

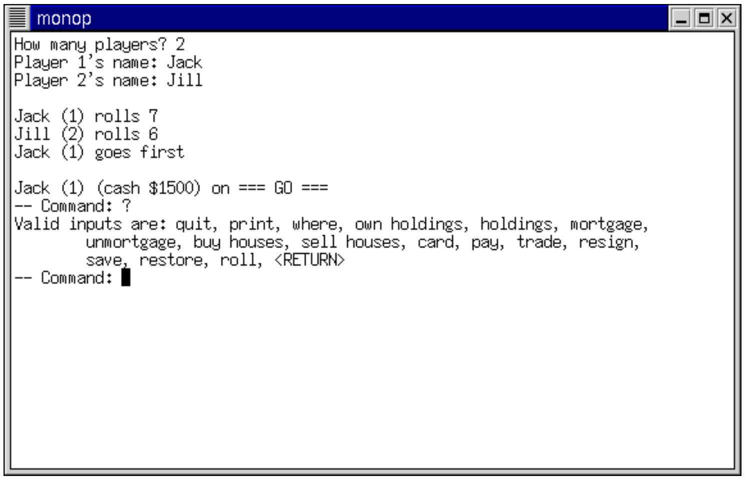
3. Click the **Run** button to run `initdeck`, which creates the cards used in the game.

   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.

Figure 114: Playing monop



```
How many players? 2
Player 1's name: Jack
Player 2's name: Jill

Jack (1) rolls 7
Jill (2) rolls 6
Jack (1) goes first

Jack (1) (cash $1500) on === GO ===
-- Command: ?
Valid inputs are: quit, print, where, own holdings, holdings, mortgage,
        unmortgage, buy houses, sell houses, card, pay, trade, resign,
        save, restore, roll, <RETURN>
-- Command: 
```

14

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 \
   --projectname /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.

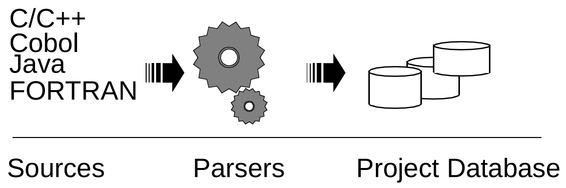
```
~bin/snavigator --projectname foo
```

Part II:
Programmer’s Guide

Introduction

Source-Navigator is based on two software components: a database engine and a graphical user interface for representing information about software projects which are held in the database. Source-Navigator has been designed to permit third parties to customize and extend Source-Navigator into specific problem domains.

Figure 115: Source-Navigator Component Overview



Software Development Kit

With the Software Development Kit (SDK), you can:

- modify the graphical user interface (see “Customization” on page 137).
- write new parsers that allow Source-Navigator to support additional programing languages (see “Adding Parsers” on page 153).
- query the database for specific information (see “Database API” on page 163).
- use other applications in conjunction with Source-Navigator (see “Interapplication Communication” on page 205).

The graphical user interface (GUI) is based on Tcl and Tk (v. 8.1). If you are not familiar with the Tcl programming language, please refer to a reference text such as ***Practical Programming in Tcl and Tk***[1] and ***Tcl and the Tk Toolkit***[2].

[1] Welch, Brent B. 1997. ***Practical Programming in Tcl and Tk***. 2nd ed. ISBN 0-13-616830-2.

[2] Ousterhout, John K. 1994. ***Tcl and the Tk Toolkit***. ISBN 0-201-63337-X.

1

Customization

This chapter describes how to change the start-up and runtime behaviors of Source-Navigator. You can also customize menu items and keyboard shortcuts for the **Editor**. For more information see “Editor” on page 45.

The `profile` File

A general configuration file, `~/share/etc/profile`, is a system-wide configuration file for GUI language and database cache size. These form the default values that all users inherit for all the tools.

If you would like to customize Source-Navigator for a specific user, you may create a file called `$HOME/.sn/profile` (on UNIX) or `%USERPROFILE%/.sn/profile` (on Windows) to contain the configuration for that user’s preferences.

Configurable Settings

The settings that may be customized are `sn_language` and `encoding`, which designates the language of the GUI, and `sn_mailhost`, which tells Source-Navigator how to connect to your mailer.

The format for a user's `profile` file is `setting name:value`, one entry per line.

Table 7: User-Specific Settings

Setting Name	*Default*	*Possible Values*
`sn_language`	`english`	`english`, `german`, or `japanese`
`encoding`	`ISO8859-1`	see the character set encoding combo-box in the **Project Preferences** dialog
`sn_mailhost`	`mailhost`	`mailhost`, DNS name, or IP address of the mailhost

An example `profile` file for the German user interface is:

```
sn_language:german
sn_mailhost:mailhost
```

The `rc.tcl` Start-up File

The information in the following section assumes that you are conversant with Tcl, which is used by Source-Navigator for program configuration tasks.

You can create a file called `rc.tcl`, which will automatically load when Source-Navigator starts. If you have either a `~/share/etc/rc.tcl` or a `$HOME/.sn/rc.tcl` file, Source-Navigator reads and executes the file with the Tcl `source` command. The system-wide configuration file is read first; your own configuration file is read afterwards and can override the site-local defaults.

When a Source-Navigator project opens, a Tcl procedure called `sn_rc` (if it exists) is called with no input parameters. This enables you to:

- install additional Tcl programs.
- customize keyboard shortcuts.
- launch specific applications at start-up.

Adding Events to the `rc.tcl` File

Many events in Source-Navigator have a corresponding Tcl procedure that is called when the event occurs. This provides the facility to control the appearance and behavior of each tool. You can choose whether or not to implement each of these Tcl procedures in `rc.tcl`, but it is not mandatory to do so.

Table 8: Events and Corresponding Tcl Procedures

Event	*Procedure called (with parameters)*
A new symbol browser is opened	`sn_rc_symbolbrowser{window menu}`
A new window is created	`sn_rc_mainwindow{window menu}`
A new editor view is created	`sn_rc_editor{view text}`
A new class view is created	`sn_rc_classbrowser{view classtree memberlist}`
A new grep window is opened	`sn_rc_grep{view list}`
A new include view is created	`sn_rc_include{view canvas}`
A new make window is opened	`sn_rc_make{view list}`
A new retriever view is created	`sn_rc_retriever{view list}`
A new cross-reference view is created	`sn_rc_xref{view canvas}`
The preferences dialog is opened	`sn_rc_preferences{win}`
The project editor dialog is opened	`sn_rc_projecteditor{win menu}`
A project is opened	`sn_rc_project_open{projectdb}`
The version control browser is started	`sn_rc_project_editor{win menu list}`
The retriever is started	`sn_rc_retrieve{win menu list}`

Table 9: Parameters for Tcl Event Procedures

Parameter	*Description*
`win`	The name of the Tk window widget. Referencing `$win.exp` returns the name of the Tk toolbar widget, and `$win.menu` returns the name of the Tk menu widget.
`view`	The name of the current view.
`list`	A window-specific list of data (for example, a list of filenames).
`canvas`	The name of the Tk canvas widget used to draw graphs in those tools where it is applicable.
`menu`	The path to the menus.
`classtree`	A path to a Source-Navigator tree widget.
`memberlist`	A path to a Source-Navigator tree table widget.

When a **Symbol Browser** window is created, a Tcl procedure called sn_rc_symbolbrowser, if it exists, is called with the following input parameters:

- the name of the **Symbol Browser** window.
- the path of the menu.

Changing Functionality Within the Symbol Browser

The following example script lists the names of the button widgets of a **Symbol Browser** toolbar:

```
proc sn_rc_symbolbrowser {top menu} {
    set toolbar_frame $top.exp
    puts stdout [join [winfo children \
    $toolbar_frame] "\n"]
}
```

which, when using the default toolbar, provides output similar to:

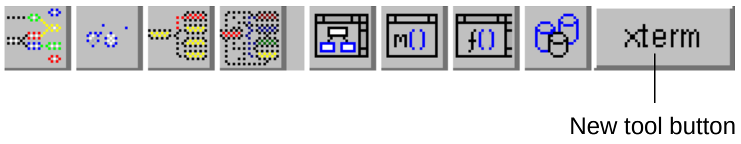
```
.multisymbr-1.exp.tree
.multisymbr-1.exp.class
.multisymbr-1.exp.xref
.multisymbr-1.exp.inc
.multisymbr-1.exp.space
.multisymbr-1.exp.retrfr
```

Example: adding a DOS shell or xterm toolbar button

The following example code adds a new button that starts a DOS shell on Windows and an xterm application on UNIX:

```
proc sn_rc_symbolbrowser {top menu} {
  global tcl_platform

  set tool_frame $top.exp

  # Set a variable that identifies the text widget on
  # the status bar.
  set info $top.msg.msg

  # Create a new button on the toolbar. Note that the
  # button’s command invokes the command line tool
  # in the background so that Source-Navigator is not
  # blocked while the tool is running. On Win32, we emulate
  # ‘xterm’ with ‘cmd’.

  if {$tcl_platform(platform) != "windows"} {
    set cmdline "exec xterm -T {Source-Navigator} &"
    set description xterm
  } else {
    set cmdline "exec cmd /c start cmd &"
    set description shell
  }
  button $tool_frame.xterm -text $description \
        -command $cmdline

    # Set the main window’s status bar to read "xterm"
    # or "shell" when the mouse pointer is over the region
    # of the button.
    balloon_bind_info $tool_frame.xterm \
        "Starts a new $description"
    bind $tool_frame.xterm <Leave> "set $top.msg {}"

    # Pack this button onto the toolbar.
    pack $tool_frame.xterm -side left
  }
```

Figure 116: Adding a new toolbar button



Adding Menus and Submenus

You can use the `rc.tcl` file to add new menus to the menu bar or new menu items to existing menus.

Example: adding an Extras menu

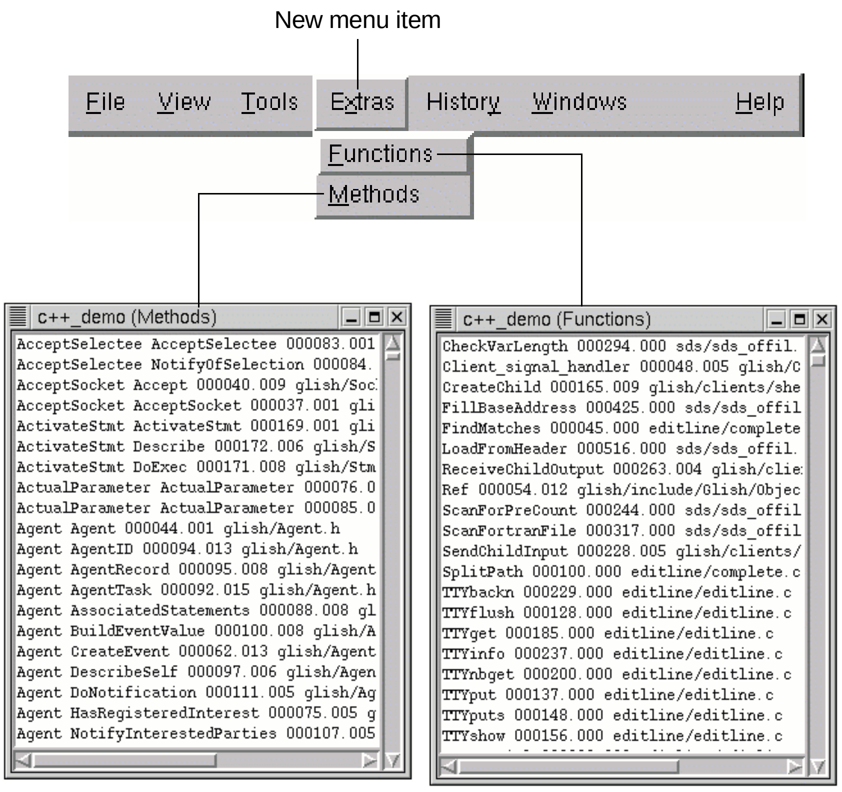
The following example shows how to add a new menu with two submenus. The first procedure creates the menu and submenu items, and the next two are the procedures that each of the submenu items actually execute.

```
proc sn_rc_symbolbrowser {win menu} {
  # Abbreviate the name of the menu to "extras".
  set extras $menu.extras
  # Create a new menu called "Extras".
  menu $extras -tearoff 0
  # Add a menu item to the menu. The second character is
  # the designated hot key. Place this menu fourth in
  # the menu bar.
  $menu insert 3 cascade -label Extras -underline 1 \
    -menu $extras
  # Add two items to the menu.
  $extras add command -label Functions -command \
    "custom_show_symbols Functions" -underline 0
  $extras add command -label Methods -command \
    "custom_show_symbols Methods" -underline 0
}

proc custom_show_symbols {symboltype} {
  # Set the scope appropriately.
  switch -- $symboltype {
    "Methods" {
      set scope "md"
```

```
                }
                default {
                  set scope "fu"
                }
              }

              if {[info commands paf_db_$scope] == ""} {
                sn_error_dialog \
                 "No symbols of the type <$scope> are available in the project."
                 return
              }

              # Generate a unique name for our new top-level window
              # widget.
              set w .custom_win_$scope
              # See if the window already exists.
              if {![info exists $w]} {
                # Create a new top-level window with a unique name.
                toplevel $w
                # Change its title.
                wm title $w "[sn_read_option project-name] ($symboltype)"
                # Put a frame around this window and add scrollbars
                # that scroll through a listbox.
                frame $w.frm
                scrollbar $w.frm.scrollx -orient horizontal \
                  -relief sunken -command "$w.frm.symlist xview"
                scrollbar $w.frm.scrolly -relief sunken \
                  -command "$w.frm.symlist yview"
                # Create a listbox to hold the symbol names.
                listbox $w.frm.symlist -height 20 -width 40 \
                  -xscrollcommand "$w.frm.scrollx set" \
                  -yscrollcommand "$w.frm.scrolly set"
                # If the user double-clicks in the list box, find
                # the nearest entry in the list and pass the name of
                # the symbol to custom_edit_file. This will launch
                # us into the source file where that symbol is
                # defined.
                bind $w.frm.symlist <Double-1> {
                  custom_edit_file [%W get [%W nearest %y]]
                }
                # Pack the widgets onto our new window.
                pack $w.frm.scrolly -side right -fill y
                pack $w.frm.scrollx -side bottom -fill x
                pack $w.frm.symlist -expand y -fill both
                pack $w.frm -fill both -expand y
              } else {
                # The window already exists, so just delete
                # everything from the listbox and we'll re-insert
                # the current items in the next step.
                $w.frm.symlist delete 0 end
              }
              # Put all of the database keys into the list.
              eval $w.frm.symlist insert end \
                [paf_db_$scope seq -data]
            }
```

```
proc custom_edit_file {key} {
  # We have been passed the key of a record from either
  # the function or method database tables. We need to
  # determine the filename and line number for the
  # editor to jump into the source code.
  # The key of a record takes the form:
  # { name, start_position, filename }
  if {[llength $key] == 3} {
    set pos [lindex $key 1]
    set filename [lindex $key 2]
  } else {
    set pos [lindex $key 2]
    set filename [lindex $key 3]
  }
  sn_edit_file {} $filename $pos
}
```

Figure 117: Adding a Menu Item to the Menu Bar



```
AcceptSelectee AcceptSelectee 000083.001
AcceptSelectee NotifyOfSelection 000084.
AcceptSocket Accept 000040.009 glish/Soc
AcceptSocket AcceptSocket 000037.001 gli
ActivateStmt ActivateStmt 000169.001 gli
ActivateStmt Describe 000172.006 glish/S
ActivateStmt DoExec 000171.008 glish/Stm
ActualParameter ActualParameter 000076.0
ActualParameter ActualParameter 000085.0
Agent Agent 000044.001 glish/Agent.h
Agent AgentID 000094.013 glish/Agent.h
Agent AgentRecord 000095.008 glish/Agent
Agent AgentTask 000092.015 glish/Agent.h
Agent AssociatedStatements 000088.008 gl
Agent BuildEventValue 000100.008 glish/A
Agent CreateEvent 000062.013 glish/Agent
Agent DescribeSelf 000097.006 glish/Agen
Agent DoNotification 000111.005 glish/Ag
Agent HasRegisteredInterest 000075.005 g
Agent NotifyInterestedParties 000107.005
```

```
CheckVarLength 000294.000 sds/sds_offil.
Client_signal_handler 000048.005 glish/C
CreateChild 000165.009 glish/clients/she
FillBaseAddress 000425.000 sds/sds_offil
FindMatches 000045.000 editline/complete
LoadFromHeader 000516.000 sds/sds_offil.
ReceiveChildOutput 000263.004 glish/clie
Ref 000054.012 glish/include/Glish/Objec
ScanForPreCount 000244.000 sds/sds_offil
ScanFortranFile 000317.000 sds/sds_offil
SendChildInput 000228.005 glish/clients/
SplitPath 000100.000 editline/complete.c
TTYbackn 000229.000 editline/editline.c
TTYflush 000128.000 editline/editline.c
TTYget 000185.000 editline/editline.c
TTYinfo 000237.000 editline/editline.c
TTYnbget 000200.000 editline/editline.c
TTYput 000137.000 editline/editline.c
TTYputs 000148.000 editline/editline.c
TTYshow 000156.000 editline/editline.c
```

Changing Functionality Within the Editor

All Source-Navigator **Editor** windows execute a Tcl procedure called `sn_rc_editor` (if it exists) at start-up. You can implement this procedure in the file `rc.tcl` as described in the section "Adding Events to the rc.tcl File" on page 139. The input parameters of this procedure are as follows:

- the name of the view.
- the name of the text widget.

Example: adding or changing Editor keyboard shortcuts

You can add or change keyboard shortcuts for the **Editor** using commands shown in the following example `rc.tcl` file. This example demonstrates how the built-in **Editor** can be customized to emulate other popular editors.

Note that the `bind` command assigns the key bindings **Ctrl+A** and **Ctrl+E** to move the cursor to the beginning and end of a line, respectively.

```
proc sn_rc_editor {view text} {
  set top [winfo toplevel $view]
  set menu_frame $top.menu
  set tool_frame $top.exp

  # Bind Control-a to jump to the start of the line
  # (like emacs).
  bind $text <Control-a> {
    %W mark set insert "insert linestart"
    %W yview -pickplace insert
    break
  }
  # Bind Control-e to jump to the end of the line
  # (like emacs).
  bind $text <Control-e> {
    %W mark set insert "insert lineend"
    %W yview -pickplace insert
    break
  }
}
```

For more information about bindings, see ***Practical Programming in Tcl and Tk*** by Brent B. Welch, page 285.

Example: changing behavior of Editor toolbar buttons

The following example shows how to change the behavior of the **Compile** button so it compiles the current source file using the Java™ bytecode compiler rather than `gcc`.

NOTE This example only works when the extended toolbar buttons are enabled. To enable them, from the **File** menu, select **Project Preferences**. Select the **Edit** tab and click the **Extended toolbar buttons** checkbox.

```
proc sn_rc_editor {view text} {
  set top [winfo toplevel $view]
  set menu_frame $top.menu
  set tool_frame $top.toolbar.editfr

# MAKE SURE THAT THE EXTENDED TOOLBAR BUTTONS ARE ENABLED.

  # Reassign the command associated with the "compile" button.
  if {[winfo exists $tool_frame.compile]} {
    $tool_frame.compile config \
      -command "custom_compile $view"
  }
}

proc custom_compile {view} {
  # Save the preconfigured "make" command line; we need
  # to tamper with it in this procedure.
  set temp [sn_read_option make-command]
  # Extract the filename.
  set file [$view cget -filename]
  # Set the "make" command line appropriately.
  if {[string match {*.jav*} $file]} {
    sn_modify_option make-command "javac \"$file\""
  }

  # Byte-compile the source file.
  sn_make

  # Restore the "make" command line.
  sn_modify_option make-command $temp
}
```

Example: adding a button to the Editor’s extended toolbar

The following example shows how either the Windows Notepad editor or the UNIX vi editor can be integrated into **Editor** toolbar as an auxiliary editor.

NOTE This example only works when the extended toolbar buttons are enabled. To enable them, from the **File** menu, select **Project Preferences**. Select the **Edit** tab and click the **Extended toolbar buttons** checkbox.

```
proc sn_rc_editor {view text} {
  global tcl_platform

  set topw [winfo toplevel $view]
  set tool_frame $topw.toolbar
  # On Windows call Notepad to edit a file.
  if {$tcl_platform(platform) == "windows"} {
    # Create a new button to edit the file using Notepad.
    button $tool_frame.vi -text Notepad \
      -command "exec notepad \[$view getfilename\]"
      balloon_bind_info $tool_frame.vi \
        "Edit current file using the Notepad editor."
    } else {
      # Create a new button to edit the file using vi
      button $tool_frame.vi -text vi \
        -command "exec xterm -T vi -e vi \[$view getfilename\]"
      balloon_bind_info $tool_frame.vi \
        "Edit current file using vi"
    }
    # Pack this button onto the toolbar.
    pack $tool_frame.vi -side left
  }
```

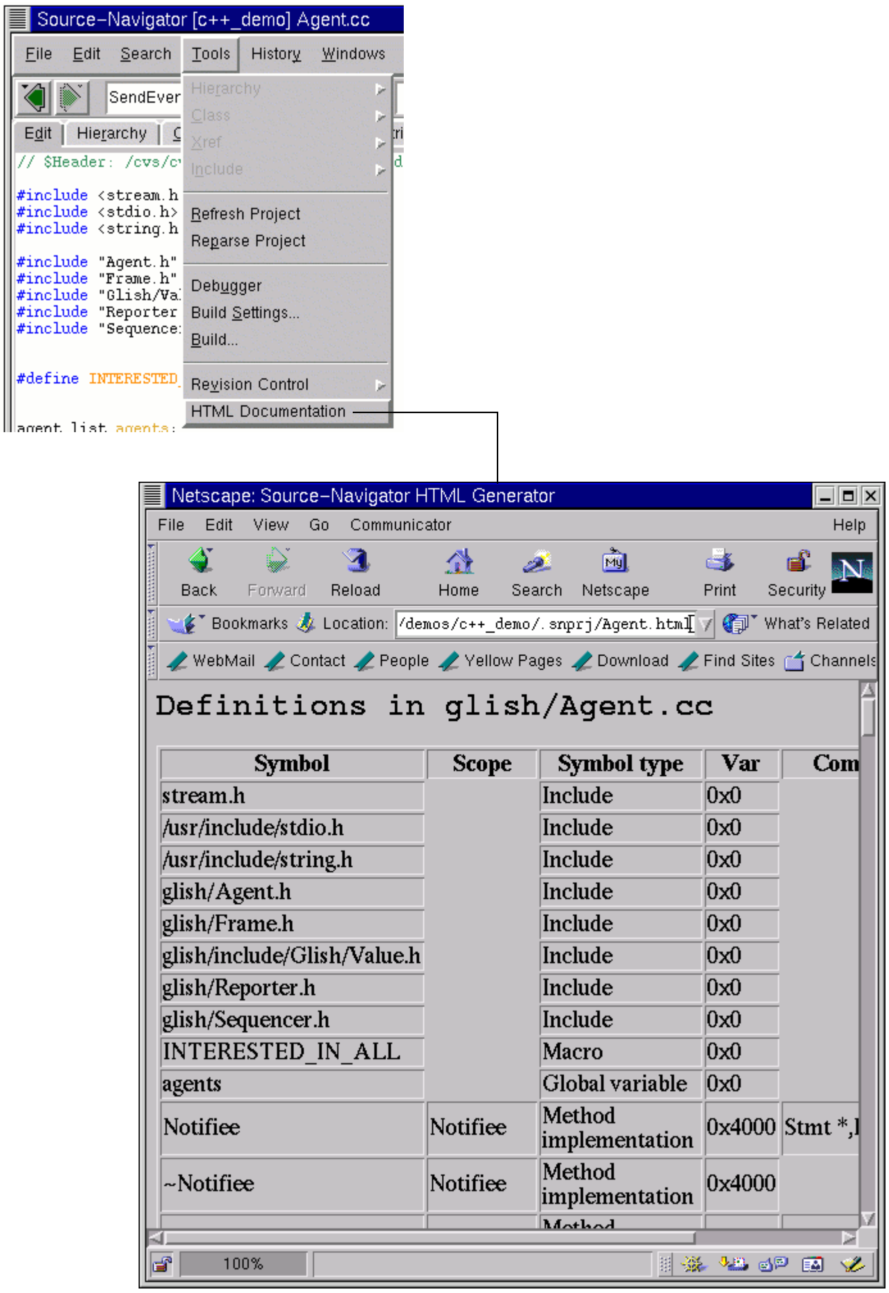
Example: generating an HTML file of the project database

This example shows how you can display project database information in a format that is different from what Source-Navigator usually provides.

```
proc html_doc {topw} {
  set actview [$topw ActiveWidget]
  set ed [MultiWindow&::list_find_editor $actview]
  if {$ed == ""} {
    bell; return
  }
  doc_start [list [$ed getfilename]]
}

proc sn_rc_editor {view editor} {
  # main window
  set topw [winfo toplevel $view]
  # add a menu entry for the html documentation
  set mn $topw.menu
  $mn.tools add command \
      -label "HTML Documentation" \
      -command "html_doc $topw"
}
```

In the **Editor**, from the **Tools** menu, select **HTML Documentation**, and Source-Navigator brings up a browser window containing definitions and cross-references of the symbols in your project.

Figure 118: Generating an HTML Representation of the Project Database


Source–Navigator [c++_demo] Agent.cc
File Edit Search Tools History Windows
SendEver
Edit Hierarchy
Hierarchy
Class
Xref
Include
Refresh Project
Reparse Project
Debugger
Build Settings...
Build...
Revision Control
HTML Documentation
Netscape: Source–Navigator HTML Generator
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: /demos/c++_demo/.snprj/Agent.html
What's Related
WebMail Contact People Yellow Pages Download Find Sites Channels
100%

```
// $Header: /cvs/c

#include <stream.h
#include <stdio.h>
#include <string.h

#include "Agent.h"
#include "Frame.h"
#include "Glish/Va
#include "Reporter
#include "Sequence

#define INTERESTED_

agent_list agents;
```

Definitions in glish/Agent.cc

Symbol	Scope	Symbol type	Var	Com
stream.h		Include	0x0	
/usr/include/stdio.h		Include	0x0	
/usr/include/string.h		Include	0x0	
glish/Agent.h		Include	0x0	
glish/Frame.h		Include	0x0	
glish/include/Glish/Value.h		Include	0x0	
glish/Reporter.h		Include	0x0	
glish/Sequencer.h		Include	0x0	
INTERESTED_IN_ALL		Macro	0x0	
agents		Global variable	0x0	
Notifiee	Notifiee	Method implementation	0x4000	Stmt *,I
~Notifiee	Notifiee	Method implementation	0x4000	
		Method		

Error Formats

Source-Navigator can receive and act upon messages from external tools such as a compiler or debugger through use of a configurable error format file that can be found in `~/share/etc/sn_cmp_g.pat`.

The contents of this error-format file is as follows; the comment above each regular expression illustrates the kind of text that the given expression can handle. For more information on regular expressions see “Grep” on page 83.

```
# Source-Navigator regular expressions for compiler,
# debugger and grep patterns.

# Source-Navigator supports also blanks, so the patterns
# must be so defined that blanks could be a part of the filename

# "filename.c", line 789
"([^" ]+)",[ ]+line[ ]+([0-9]+)

# filename.c, line 789
([^ ]+),[ ]+line[ ]+([0-9]+)

# line 789, filename.c,
line[ ]+([0-9]+),[ ]+([^, ]+)

# [filename.c:789]
\[([^\[: ]+):([0-9]+)\]

# filename.c:789
([^: ]+):[ ]*([0-9]+)

# filename.c(789)
([^ ]+\.[^ ]+)\(([0-9]+)\)

# filename.c(789,11) or filename.c(789.11)
([^ ]+\.[^ ]+)\(([0-9]+[,.][ ]*[0-9]+)\)

# /dir/filename.c, 789
([^/ ]+),[ ]+([0-9]+)
```

The lines beginning with the hash # symbol are comments showing matching patterns; the GNU regular expression describes the expressions. Every expression must contain two expressions enclosed in parentheses (for the file name and line number).

2

Predefined Language Conventions

Source-Navigator does not invoke a compiler to build its databases; instead, it has a plug-in parser for each language it supports. For details on how to add plug-in parsers to Source-Navigator, see “The Parser Toolbox Library” on page 155.

Predefined Parsers

Source-Navigator was designed to support not only the most common software development languages, but to support them together. With Source-Navigator you can follow references from a C++ method to a FORTRAN subroutine, and even to assembly language. To accomplish this task efficiently, Source-Navigator shares terms between multiple languages, even when the languages are defined in different terms.

For example, a C `struct` is represented in Source-Navigator as a `class`; there is no separate `struct` type for C. By unifying these terms, Source-Navigator greatly simplifies both the task of multi-language code comprehension and its own internal organization.

The C and C++ Parser

This parser understands C++, K&R C, and ANSI C languages, including the pre-processor directives. Pre-processing is not necessary to parse C or C++ source code. During project creation a default include search path is created that has the same role as the `-I` option for the C pre-processor. The include search path can be modified during project creation or from the **File** menu by selecting **Project Preferences** → **Include**.

The FORTRAN Parser

The FORTRAN parser understands the FORTRAN 77 syntax, plus extensions such as `record`, `structure`, and `include`. The usual FORTRAN extensions are understood by the parser as well. The include search path has the same role as the `-I` option for the C pre-processor. The include search path can be modified during project creation or from the **File** menu by selecting **Project Preferences** → **Include**.

The FORTRAN structure declaration is mapped by the tools to a class. Structure members are treated as instance variables.

The COBOL Parser

The COBOL parser understands these dialects: ANSI '74 Standard, ANSI '85 Standard (ANSI X3.23-1985), IBM OS/VS COBOL, IBM VS COBOL II, IBM SAA COBOL/370, IBM DOSVS COBOL, X/Open, and Micro Focus COBOL.

The Tcl and `[incr Tcl]` Parser

The Tcl parser understands all versions of Tcl/Tk through version 8.1. In addition, it understands `[incr Tcl]` versions 1.5 and 2.x.

The Java Parser

The Java parser understands Java 1.0.

The PowerPC Assembly Parser

The PowerPC assembly parser understands model number 601 assembly language.

NOTE The empty areas of Table 10 represent types that don't exist in each language.

Table 10: Type Definitions for Supported Languages

Type Abbrev.	*C/C++ and Java*	*FORTRAN*	*COBOL*	*Tcl*	*[incr Tcl]*	*PowerPC assembly*
cl	Class, Struct	Structure	Structure	Namespace	Class	
com		Common Block				
cov		Common variable				
con	#define[*] const[†] static final[‡]	Constant				Const
e	Enum					
ec	Enum value					
fd	Function declaration					
fr	Friend					
fu	Function	Function, Label	Function, Label	Procedure	Procedure	Function
gv	Global variable		Global variable	Global variable	Global variable	Global variable
iv	Instance variable	Instance variable[**]	Instance variable[††]	Namespace variable	Instance variable	
lv	Local variable			Local variable	Local variable	
ma	Macro					Macro
md	Method declaration					
mi	Method implemen-tation			Namespace procedure	Method implemen-tation	
su		Subroutine				
t	Typedef					
un	Union					

* In C and C++.
† In C and C++.
‡ In Java.
** For structure members.
†† For structure members.

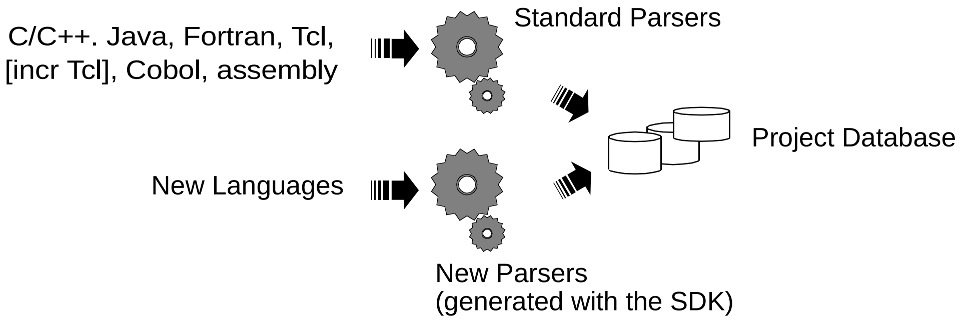
3

Adding Parsers

Each parser in Source-Navigator is capable of understanding source files written in a specific programming language. In the Source-Navigator suite, parsers are stand-alone executables which adhere to a consistent command line interface. This interface allows Source-Navigator to control the parser's behavior through command line switches that the parser is expected to observe.

The SDK provides a C-based application programming interface (API) which enables parsers to insert information into a project database.

Figure 119: Parsers Overview


C/C++. Java, Fortran, Tcl, [incr Tcl], Cobol, assembly
Standard Parsers
Project Database
New Languages
New Parsers (generated with the SDK)

All files listed in Table 11 can be found in the `~/share/sdk` directory.

Table 11: SDK-Related Files

Subdirectory	***Files***	***Description***
`/include/`	`sn.h` `snptools.h`	Include files

`/lib/`	`libdbutils.a`
	`libpafdb.a`
	`libsnptools.a`
	`libtcl8.1.a`
		Source-Navigator library files

`/parsers/examples/assembly`	`Makefile`
	`README`
	`a.c`
	`abrowser.l.in`
	`b.c`
	`build-macros`
	`linux-i486-elf.m4`
	`solaris-sparc.m4`
	`toolbox.m4`	

`/parsers/examples/elf`	`Makefile`
	`README`
	`blobsql.elf`
	`ebrowser.l`
		Makefile for `ebrowser.l`
		The latest release information
		Example source
		Parser (*lex*)

Parsers may be implemented in any programming language. Naturally, the task of writing a parser is significantly simpler when using compiler generation tools such as GNU flex. All of the examples provided with Source-Navigator are written using GNU flex.

A sample parser for the ELF language (an embedded SQL-like language) is provided with the SDK. To experiment with this parser, change to the `~/share/sdk/parsers/examples/elf` directory and type

```
make test
```

This will compile the parser and then parse a number of ELF source files, placing items of interest into a project database.

The Parser Toolbox Library

A library, implemented above the API, simplifies the task of writing new parsers. This library, known as the *parser toolbox*, allows programmers to focus on the issues of parsing source files in their chosen language and then storing the relevant information in the database.

The toolbox provides a number of C functions that can help you write your parser with less effort. These functions can be grouped as follows:

- functions to maintain line and column counts in your source files. Whenever the parser encounters an interesting symbol in the source program, it must know where in the source file the symbol occurred.

  That is, the function `sn_advance_line()` will increment a line counter maintained internally by the toolbox library. This function would be called along with other actions that the parser might perform when it encounters a newline character in the source text.
- a function to determine the name of the source file currently being parsed.
- miscellaneous text processing functions for counting the number of lines and columns consumed by a given block of text and so on.
- an entry point, called `sn_main`, which manages all of the details of interfacing with Source-Navigator and parsing each source file. In most circumstances, it is sufficient for the parser to call `sn_main` and allow this function to call back to your actual parsing function when required.

  This introduces a number of important concepts best explained with an example:

```
int sn_main(int argc, char *argv[], char *lang_string,
    FILE **infile_ref, int(*parse)(), void(*reset)());
char group[] = "java";
int main(int argc, char *argv[])
{
  return sn_main(argc, argv, group, &yyin, yylex, reset);
}
```

  When calling `sn_main`, it is necessary to pass:

  - the `argc` and `argv` variables as passed into your `main()` function. This allows the library to access the command line options given by Source-Navigator.
  - a pointer to a string identifying the language. In the example above, the string is called `group`.
  - a pointer to a `FILE *` stream variable. You must pass the address of this variable as the library will manipulate the stream when opening new source files. If you are using GNU flex/bison, then pass a pointer to the global variable `yyin`.

- A pointer to a function which takes no arguments and returns an `int`. This is a pointer to your actual parsing function. If you are using GNU flex/bison, then pass `yylex` or `yyparse`.
- A pointer to a function which takes no arguments and returns `void`. This function is expected to perform any actions prior to processing the next source file. Typically this function might look like:

```
void reset()
{
  sn_reset_line(); /* reset line count */
  sn_reset_column(); /* reset column count */
}
```

A detailed description of the functions available in the parser toolbox library can be found in `~/share/sdk/include/snptools.h`.

Unless the parser has to do something out of the ordinary, it should be possible to create a new parser by following these steps:

1. `#include snptools.h` in your program (or within the verbatim section of your lex specification).
2. Define a `main` function which calls `sn_main` with the appropriate arguments and returns the result of `sn_main` to the host environment.
3. Utilize the parser toolbox routines to simplify your work within the parser. For example, `sn_message` may be called to display messages in a dialog box that is shown to the user during the parsing process.
4. When your parser recognizes important language constructs such as comments, function declarations, or function invocations, call the appropriate `sn_insert` function to insert this information into the project database.
5. Link your final program with the `sntools` library. A typical command line for linking a parser can be found in the Makefile given in the `~/share/sdk/parsers/examples/elf` directory.

Project Database Calls

The toolbox library provides a number of functions for inserting information into the database that the parser will encounter in the source text. Each of these functions return an `int` with two possible return values:

0	success
-1	failure

sn_insert_symbol

```
int sn_insert_symbol(int id_type, char *classname,
  char *identifier, char *filename, int start_lineno,
  int startCol, int endLine, int endCol, unsigned long attrib,
  char *returnType, char *argTypes, char *argNames,
  char *comment, int highStartLine, int highStartCol,
  int highEndLine, int highEndCol);
```

`sn_insert_symbol` inserts a symbol into the project database.

`type` determines the type of the symbol. Possible values are:

`SN_TYPE_DEF`	type definitions (such as a C `typedef`)
`SN_CLASS_DEF`	class definition (particularly for object-oriented languages)
`SN_MBR_FUNC_DEF`	member functions
`SN_MBR_VAR_DEF`	member variables
`SN_ENUM_DEF`	enumerations
`SN_CONS_DEF`	constants
`SN_MACRO_DEF`	macros
`SN_FUNC_DEF`	functions
`SN_SUBR_DEF`	subroutines
`SN_GLOB_VAR_DEF`	global variables
`SN_COMMON_DEF`	common blocks
`SN_COMMON_MBR_VAR_DEF`	member variables within a common block
`SN_CLASS_INHERIT`	class inheritance
`SN_MBR_FUNC_DCL`	member function declarations
`SN_FUNC_DCL`	functions
`SN_ENUM_CONST_DEF`	enumeration constant
`SN_UNION_DEF`	definitions of unions or variant records
`SN_FRIEND_DCL`	friends (for C++)
`SN_NAMESPACE_DEF`	name spaces
`SN_EXCEPTION_DEF`	exceptions
`SN_LOCAL_VAR_DEF`	local variables
`SN_VAR_DCL`	variables
`SN_INCLUDE_DEF`	include files
`SN_REF_UNDEFINED`	reference to undefined symbols

`classname`	the name of the class, structure or common block of the symbol if the symbol's type is one of `SN_MBR_FUNC_DEF`, `SN_MBR_VAR_DEF`, `SN_COMMON_MBR_VAR_DEF`, or `SN_CLASS_INHERIT`. Otherwise, `classname` must be a NULL pointer. If the symbol's type is `SN_CLASS_INHERIT`, `classname` contains the name of the base class.
`identifier`	the name of the symbol to be inserted into the project database.
`filename`	the name of the source file in which this symbol was encountered.
`start_lineno`	the line number of the position where the symbol starts.
`startCol`	the column number of the position where the symbol starts.
`endLine`	the line number of the position where the symbol ends.
`endCol`	the column number of the position where the symbol ends.
`attrib`	contains attributes of the symbol definitions (see `sn.h`).
`returnType`	a string describing the return type of the function, subroutine or method. If the symbol is not one of these types, pass a NULL pointer.
`argTypes`	a string containing a comma-separated list of argument types for the argument list of functions, subroutines or methods. If the symbol is not one of these types, pass a NULL pointer.
`argNames`	a string containing a comma-separated list of argument names for the argument list of functions, subroutines or methods. If the symbol is not one of these types, pass a NULL pointer.
`comment`	a string containing the comment that often occurs after a definition in the source text. Note that eight-bit characters including `\n` may be used in the string. If there is no comment, pass a NULL pointer.
`highStartLine`	the line number of the position where the highlighting of the symbol starts.
`highStartCol`	the column number of the position where the highlighting of the symbol starts.
`highEndLine`	the line number of the position where the highlighting of the symbol ends.
`highEndCol`	the column number of the position where the highlighting of the symbol ends.

Examples

The following example inserts a class definition:

```
sn_insert_symbol(SN_CLASS_DEF, NULL, classname,
  sn_current_file(), sn_line(), sn_column(), sn_line(),
  sn_column() + strlen(classname), 0L, NULL, NULL,
  NULL, NULL, sn_line(), sn_column(), sn_line(),
  sn_column() + strlen(classname));
```

The following example inserts a method definition:

```
sn_insert_symbol(SN_MBR_FUNC_DEF, classname, methodname,
  sn_current_file(), sn_line(), sn_column(), sn_line(),
  sn_column() + strlen(methodname), 0L, NULL, NULL,
  NULL, NULL, sn_line(), sn_column(), sn_line(),
  sn_column() + strlen(methodname));
```

For a C function with the following prototype:

```
int transform(struct coord * p, int x, int y, unsigned att);
```

the following example inserts the definition into the database:

```
sn_insert_symbol(SN_FUNC_DEF, NULL, "transform",
  sn_current_file(), sn_line(), sn_column(), sn_line(),
  sn_column + len, 0L, "int", "struct coord *, int, int,
  unsigned", "p,x,y,att", NULL, NULL, sn_line(), sn_column(),
  sn_line(), sn_column() + len);
```

sn_insert_xref

`sn_insert_xref` inserts cross-referencing information.

One of the most useful aspects of Source-Navigator is its cross-referencing capabilities. For instance, it is possible to see which functions are used by other functions or which functions modify a particular global variable.

Where possible, a parser should attempt to collect information of this nature and insert it into the project database. In a language with a flat namespace such as C, this can be achieved by noting the name of the current function within the parser. If a function invocation is encountered in the source text, then the cross-referencing can be inferred based on the current function’s name and the function being called.

Cross-referencing information is added using:

```
int sn_insert_xref(int type, int scope_type, int scope_level,
  char *classname, char *funcname, char *argtypes,
  char *refclass, char *refsymbol, char *ref_arg_types,
  char *filename, int lineno, int acc);
```

`type`	describes the type of the referenced symbol. It must be one of the following:
	`SN_REF_TO_TYPEDEF`
	`SN_REF_TO_DEFINE`

SN_REF_TO_ENUM
SN_REF_TO_STRUCT
SN_REF_TO_UNION
SN_REF_TO_CLASS
SN_REF_TO_FUNCTION
SN_REF_TO_MBR_FUNC
SN_REF_TO_MBR_VAR
SN_REF_TO_COMM_VAR
SN_REF_TO_CONSTANT
SN_REF_TO_SUBROUTINE
SN_REF_TO_GLOB_VAR
SN_REF_TO_LOCAL_VAR
SN_REF_TO_TEMPLATE
SN_REF_TO_NAMESPACE
SN_REF_TO_EXCEPTION
SN_REF_TO_LABEL

`scope_type`	describes the type of the location where the cross-reference information is reported. It must be one of the following:
	`SN_FUNC_DEF`
	`SN_MBR_FUNC_DEF`
	`SN_SUBR_DEF`

`scope_level`	describes the scope level of the referenced symbol. It must be one of:
	`SN_REF_SCOPE_LOCAL`
	`SN_REF_SCOPE_GLOBAL`

`classname`	a string containing the class name of the method if `scope_type` is `SN_MBR_FUNC_DEF`; otherwise, it must be a NULL pointer.
`funcname`	a string containing the function, method or subroutine name in which the reference information is reported.
`argtypes`	a string containing a comma-separated list of argument types for the argument list of functions, subroutines or methods. Pass NULL if there are no arguments.
`refclass`	a string containing the class, structure or common block name of the referred symbol. If the symbol is not within a namespace, pass a NULL pointer.
`refsymbol`	the name of the referred symbol.

`ref_arg_types`	a string containing a comma-separated list of argument types if the referred symbol's type is a method, subroutine or function. Pass NULL if there are no arguments or the referred symbol is not one of these types.
`filename`	the name of the source file in which the reference information is reported.
`lineno`	the line number of the source file in which the reference information is reported.

`acc`	the level of access to the referenced symbol and is one of:
`SN_REF_READ`	symbol is read (such as `if (x) { ... }`)
`SN_REF_WRITE`	symbol is modified (such as `x = 10`)
`SN_REF_PASS`	variable is passed to a subroutine/function
`SN_REF_UNUSED`	symbol is never used

Examples

The following example inserts cross-referencing information for a function that is called from another function.

```
sn_insert_xref(SN_REF_TO_FUNCTION, SN_FUNC_DEF,
  SN_REF_SCOPE_GLOBAL, NULL, currentFunction, NULL, NULL,
  calledFunction, NULL, sn_current_file(), sn_line(),
  SN_REF_PASS);
```

The following example inserts cross-referencing information for a function that is called from a member function called `insert` which belongs to a C++ class called `Stack`.

```
sn_insert_xref(SN_REF_TO_MBR_VAR, SN_MBR_FUNC_DEF,
  SN_REF_SCOPE_GLOBAL, "Stack", "insert", NULL, "Stack",
  "i", NULL, sn_current_file(), sn_line(), SN_REF_READ);
```

sn_insert_comment

`sn_insert_comment` inserts comments into the project database.

When comments are encountered in the source text, the parser should call `sn_insert_comment` to add these comments to the project database. In some Source-Navigator projects, the user will choose to not include comments, but the parser should call this function regardless. The library function will decide whether or not to actually store the information in the database.

Comments are added to the database using:

```
int sn_insert_comment(char *classname, char *funcname,
  char *filename, char *comment, int beg_line, int beg_col);
```

`classname`	a string containing the name of the class or method where the comment was found or NULL.
`funcname`	a string containing the name of the function or method in which the comment was found, or NULL if the comment is outside any function or method scope.
`filename`	a string containing the name of the current file being parsed.
`comment`	a string containing the comment without the comment separators. For example, in Tcl this would exclude the leading # character.
`beg_line`	the line number of the source file where the comment begins.
`beg_col`	the column number of the source file where the comment begins.

Integration with Source-Navigator

A completed parser can be integrated into Source-Navigator by following these steps:

1. Copy the parser executable into the `~/bin` directory.
2. Edit `sn_prop.cfg` in the directory `~/share/etc`. The Tcl procedure `sn_add_parser` is used to add parsers to the configuration. The following example shows how to include support for Java:

```
sn_add_parser java -suffix {*.java} \
  -brow_cmd $odd_path(bindir)/jbrowser \
  -high_cmd $odd_path(bindir)/jbrowser \
  -high_switch "-h"
```

When Source-Navigator is restarted, support for the new language becomes available. When creating a project, the **Parsers** tab in the **Project Preferences** dialog shows the new language and its associated filename extensions.

This is all that is required to add new language support to Source-Navigator. If a project is created which contains files with any of the specified filename extensions, the new parser is invoked to process those files as a part of the overall parsing process.

4

Database API

This chapter provides information that enables programmers to build applications that make use of Source-Navigator's project databases. It covers the API, including available functions and their syntax.

This chapter assumes that you are familiar with the Tcl or C programming languages. For information about Tcl, see ***Practical Programming in Tcl and Tk***[1] and ***Tcl and the Tk Toolkit***[2], and for information about C, see ***The C Programming Language***[3].

Introduction

Source-Navigator project information is stored in a *database*. There is one database for each project and the database may contain one or more *views*. A view is a named subset of records; this might be the subset of all `.h` files, or the subset of all files in a given subdirectory. Views are a powerful and fast way to narrow down a large project without building multiple projects (see page 165).

[1] Welch, Brent B. 1997. *Practical Programming in Tcl and Tk*. 2nd ed. ISBN 0-13-616830-2.

[2] Ousterhout, John K. 1994. *Tcl and the Tk Toolkit*. ISBN 0-201-63337-X.

[3] Kernighan, Brian W., and Dennis M. Ritchie. 1988. *The C Programming Language*. 2nd ed. ISBN 0-13-110362-8.

A database consists of 15 to 25 files; each file consists of a table that contains symbol and index information. Database files are regular files in the operating system that you can share between UNIX and Windows operating systems.

When you create a project with Source-Navigator, a database is created in the `projectdir/.snprj` directory. By default, this location can be changed in the Project Preferences dialog when the project is created. See “General Project Preferences” on page 26 for more information. Each database file is created with a filename starting with *`projectname`*, where *`projectname`* is the project name you chose and *`projectdir`* is the project directory you chose. One or more databases may exist in the same directory.

Each file is a *table* that contains specific symbol information and indexes. A database table can be accessed via an *index*, which is handled internally by the database, or sequentially. To use the Tcl API, you do not need additional header files, compilers, or libraries. All of the Tcl commands you need to work with a Source-Navigator project database are built into the Source-Navigator Tcl interpreter called *hyper*.

NOTES In previous versions of Source-Navigator, fields in the database were separated by space characters. To accommodate filenames that contain spaces, the field separator has been changed to an internal value that can be referenced through the read-only Tcl variable `sn_sep`. See the scripts in “Cross-Reference Tables” on page 166 for more information on this variable.

The first three lines of the scripts in this chapter are “boilerplate” text to run these scripts on UNIX. On Windows you must delete these lines and run the remainder from a file using `SNsdk`.

To run stand-alone Tcl scripts (see page 188) on Windows NT, an execution command must be used. The usage of this command is:

```
SNsdk script-name arguments
```

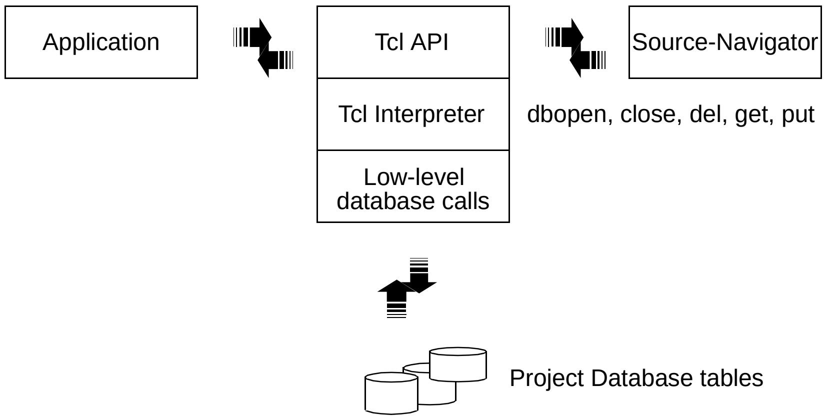
To run the `multicludes.tcl` example script on a project called `test1` in the `C:\test` directory the syntax would be:

```
SNsdk multicludes.tcl C:\test test1
```

Structure

The database may be accessed by a set of Tcl commands. The primary goals in designing the API were performance and flexibility; for high-level queries, the Tcl language provides a powerful and flexible solution.

Figure 120: Database API Overview



The database supports the `btree` and `hash` file formats. The `btree` format is a representation of a sorted, balanced tree structure. The `hash` format is an extensible, dynamic hashing scheme.

The `dbopen`, `close`, `del`, `get`, `put`, and `seq` routines are used to access the database. Optimal database tuning calls and parameters may also be configured in Tcl using the `cachesize` and `pagesize` properties (see “dbopen for Tcl” on page 168) to give optimum performance in specific applications.

Views

Views define sets of files to include or exclude from queries. The hidden files list is stored in a view. For example, in a project where you have both database- and GUI-specific source files, you can hide the database file `sql.c` and save the remaining project information as a view. Hiding this view means the records with references to the file `sql.c` must be skipped. The following example lists a view table:

```
set db_view [dbopen nav_view .snprj/cpl.2 RDONLY 0644 hash \
cachesize=300000]
puts stdout [join [ $db_view seq -data] \n]
```

The output is as follows:

```
sql.c
```

Using Views

Views have to be specified when a table is opened using the `dbopen` command; the application does not have to be changed. For more information on `dbopen` for Tcl, see page 168.

The example below uses a view. The view table in the following example (`.snprj/cpl.2`) is created using the **Editor**. The first view in a project has the suffix `.1`, the second `.2`. There is no limit for views.

```
#!/bin/sh
# Replace $HOME/snavigator with the Source-Navigator
# installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don’t forget the backslash before exec!
#
set db_view [dbopen nav_view .snprj/cpl.2 RDONLY 0644 \
    hash cachesize=300000]

set db_functions [dbopen nav_func .snprj/cpl.fu RDONLY \
    0644 btree cachesize=20000 $db_view]

# Output the list of matches with newline characters
# after each.
puts [join [$db_functions seq -data] \n]

# Force our way out of this event-driven shell.
exit
```

Cross-Reference Tables

Source-Navigator stores the cross-reference information in two tables with the suffixes `by` and `to`. They contain the same information, only their key format differs. The `to` table keeps the refers-to information and the `by` table the referred-by information. The following script opens the `to` cross-reference table of a project and lists its contents.

```
#!/bin/sh
# Replace $HOME/snavigator with the Source-Navigator
# installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don’t forget the backslash before exec!
#
set db_functions [dbopen nav_func .sn/cpl.to RDONLY 0644 btree \
  {cachesize=200000}]
```

```
puts [join [$db_functions seq -data] \n]
exit
```

The above script would generate these results:

```
# len fu # strlen fu p 000019 c.c
# main fu # glob_var gv w 000010 c.c
# main fu # len fu p 000013 c.c
# main fu # printf fu p 000013 c.c
# main fu # strcpy fu p 000012 c.c
```

The hash (#) characters mean that the symbols do not belong to any classes. To fetch only the references in the main function, modify the fetch instruction:

```
$db_functions seq -data "#${sn_sep}main${sn_sep}"
```

In the query, note that the first character of the key must be a hash (#) character because `main` is not a method but a function. To be sure that only the references of main are reported, a separator character (`$sn_sep`) has to be added to the string `main`. Without the separator, the query would report the references of all functions whose names begin with the string `main`.

An example of the result after the modification:

```
# main fu # glob_var gv w 000010 c.c
# main fu # len fu p 000013 c.c
# main fu # printf fu p 000013 c.c
# main fu # strcpy fu p 000012 c.c
```

If an application collects references to a function referred-by, it is better to use the `by` database table.

The script below opens the referred-by table and reports every reference to the global variable `glob_var` and the function `len`.

```
#!/bin/sh
# Replace $HOME/snavigator with the Source-Navigator
# installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don’t forget the backslash before exec!
#
global sn_sep
set db_functions [dbopen nav_func .snprj/cpl.by RDONLY \
     0644 btree cachesize=200000]
# Output the cross-references which match the following
# criteria, with newline characters between all of them.
puts [join [$db_functions seq -data \
     "#${sn_sep}glob_var${sn_sep}gv"] \n]
puts [join [$db_functions seq -data  \
     "#${sn_sep}len${sn_sep}fu"] \n]

# Force our way out of this event-driven shell.
exit
```

The result will be:

```
# glob_var gv # main fu w 000010 c.c
# len fu # main fu p 000013 c.c
```

The output above indicates that the symbols `glob_var` and `len` are only used by the function `main`.

Tcl API Functions

The Source-Navigator database tables can be accessed by means of Tcl commands created using the `dbopen` command. The database API can access Source-Navigator tables regardless of whether Source-Navigator is running, or a project is using the target tables.

dbopen for Tcl

The `dbopen` command opens a table for reading and/or writing. `dbopen` creates a new Tcl object (command) with the name `dbobject`.

SYNOPSIS

```
dbopen dbobject tableName access permission type ?openinfo?
```

`dbobject`	The desired name of the new command. `dbopen` will fail if there is already a command named *`dbobject`*.
`tableName`	The `dbopen` command opens a table for reading and/or writing. Files not intended for permanent storage on disk can be created by setting the `tableName` parameter to `NULL`.
`access`	The `access` argument is as specified for the Tcl `open` routine; however, only the `CREAT`, `EXCL`, `RDONLY`, `RDWR`, and `TRUNC` flags are significant. Refer to the Tcl documentation for further information on modes for opening files.
`permission`	If a new file is created as part of the process of opening it, `permission` (an integer) sets the permissions for the file. On UNIX, this is done in conjunction with the process' file mode creation mask.
`type`	The `type` argument must be either `btree` or `hash`. The `btree` format represents a sorted, balanced tree structure. The hash format is an extensible, dynamic hashing scheme.

`openinfo`	The `openinfo` optional argument must be a valid Tcl list and can be used to set type-specific properties of database tables. The syntax is as follows:
	`property1=value,property2=value,...\`
	`propertyn=value`
	See Table 12 and Table 13 for valid values for this parameter.

Table 12: Hash Table Properties

Hash table properties	***Meaning***
`bsize`	Defines the hash table bucket size, and is, by default, 256 bytes. It may be preferable to increase the page size for disk-resident tables and tables with large data items.
`factor`	Indicates a desired density within the hash table. It is an approximation of the number of keys allowed to accumulate in any one bucket, determining when the hash table grows or shrinks. The default value is 8.
`nelem`	An estimate of the final size of the hash table. If not set, or set too low, hash tables will expand gradually as keys are entered. Although a slight degradation in performance may be noticed, the default value is 1.
`cachesize`	Suggested maximum size, in bytes, of the memory cache. This value is only advisory, and the access method will allocate more memory rather than fail.

Table 13: btree Table Properties

btree table properties	***Meaning***
`flags`	The `flags` value is specified by applying the bitwise OR operation with any of these values: `R_DUP` permit duplicate keys in the tree; that is, permit insertion if the key to be inserted already exists in the tree. The default behavior is to overwrite a matching key when inserting a new key or to fail if the `R_NOOVERWRITE` flag is specified. The `R_DUP` flag is overridden by the `R_NOOVERWRITE` flag, and if the `R_NOOVERWRITE` flag is specified, attempts to insert duplicate keys into the tree will fail. If the database contains duplicate keys, the order of retrieval of key/data pairs is undefined if the get method is used; however, seq method calls with the `R_CURSOR` flag set will always return the logical “first” of any group of duplicate keys.
`cachesize`	Suggested maximum size (in bytes) of the memory cache. This value is only advisory, and the access method will allocate more memory rather than fail. Since every search examines the root page of the tree, caching the most recently used pages substantially improves access time. In addition, physical writes are delayed as long as possible, so a moderate cache can reduce the number of I/O operations significantly. Using a cache increases (but only increases) the likelihood of corruption or lost data if the system crashes while a tree is being modified. If cachesize is 0 (no size is specified), a default cache is used.
`minkeypage`	The minimum number of keys stored on any single page. This value determines which keys are stored on overflow pages; that is, if a key or data item is longer than the pagesize divided by the `minkeypage` value, it is stored on overflow pages instead of in the page itself. If `minkeypage` is 0 (no minimum number of keys is specified), a value of 2 is used.

Table 13: btree Table Properties (Continued)

btree table properties	***Meaning***
`psize`	Page size is the size (in bytes) of the pages used for nodes in the tree. The minimum page size is 512 bytes, and the maximum page size is 64K. If `psize` is 0 (no page size is specified), a page size is selected based on the underlying file system I/O block size.
`lorder`	The byte order for integers in the stored database metadata. The number should represent the order as an integer; for example, big endian order would be the number 4,321. If `lorder` is 0 (no order is specified), the current host order is used.

If the file already exists (and the `TRUNC` flag is not specified), the values specified for the parameter `flags`, `lorder`, and `psize` are ignored in favor of the values used when the tree was created.

Forward sequential scans of a tree are from the least key to the greatest.

Space freed up by deleting key/data pairs from the tree is never reclaimed, although it is normally made available for reuse. This means that the `btree` storage structure is grow-only. The only solution is to avoid excessive deletions, or to create a fresh tree periodically from a scan of an existing one (see “dbcp” on page 192).

The instruction below opens (for read-only) a `btree` database table using cachesize of two MB.

```
set db [dbopen nav_classes brow.cl RDONLY 0644 btree \
    {cachesize=2000000}]
```

The next example opens a hash table that is used later as a view.

```
set db_view [dbopen nav_view .snprj/progs.1 RDONLY \
    0644 hash "cachesize=300000"]
set db_functions [dbopen nav_func .snprj/progs.fu \
    RDONLY 0644 btree "cachesize=200000" $db_view]
```

Methods

With commands created by `dbopen`, these methods can be used: `close`, `del`, `exclude`, `get`, `isempty`, `put`, `reopen`, `seq`, and `sync`. In the following examples, *`dbobject`* represents the command returned from `dbopen`.

close

SYNOPSIS

```
dbobject close
```

This method flushes any cached information to disk, frees any allocated resources, and closes the underlying table. Since key/data pairs may be cached in memory, a database should be closed or synchronized before the application exits; this flushes the cache to disk. Failure to close or synchronize can cause data loss. As a final step, *`dbobject`* is also destroyed.

del

SYNOPSIS

```
dboject del ?-glob pattern? ?-beg pattern? ?-end pattern?
    ?-regexp pattern? ?-strstr pattern? ?key? ?flags?
```

This method removes key/data pairs from the table.

With `glob`, `beg`, `end`, `regexp`, and `strstr` switches (applicable only to the `btree` tables), a pattern must be specified to delete every record whose key matches the pattern. If `key` is specified, only those records are checked for deletion whose keys begin with `key`.

The parameter `flags` may be set to the value `R_CURSOR.` Delete the record referenced by the cursor. The cursor must have previously been initialized.

This method returns the number of the deleted records.

exclude

SYNOPSIS

```
dbobject exclude view
```

This method effectively sets the view of *`dbobject`* by excluding all symbols not in the view. `View` must be the name of an already existing object created using an earlier `dbopen` command.

get

SYNOPSIS

```
dbobject get key
```

If `key` is found, this method returns `key` and the associated data in separate Tcl lists; otherwise, it returns an empty string. For more information, see “Fetching Tables” on page 172.

isempty

SYNOPSIS

```
dbobject isempty
```

This method returns 1 if the table (with its current view) is empty, otherwise 0.

put

SYNOPSIS

```
dbobject put key data ?flags?
```

This method stores key/data pairs in the table.

The parameter `flags` may be set to one of these values:

The default behavior of the `put` routines is to enter the new key/data pair, replacing any previously existing key.

`R_CURSOR`	Replace the key/data pair referenced by the cursor. The cursor must have previously been initialized.
`R_NOOVERWRITE`	Enter the new key/data pair only if the key did not previously exist.
`R_SETCURSOR`	Store the key/data pair, setting or initializing the position of the cursor to reference it. (Applicable only to the `btree` tables.)

This method returns 0 on success, and 1 if the `R_NOOVERWRITE` flag was set and the key already exists in the table.

reopen

SYNOPSIS

`dbobject` `reopen`

This method closes and reopens the table. This flushes data to disk and resets any views.

seq

SYNOPSIS

`dbobject` `seq option`

See “Fetching Tables” on page 172.

sync

SYNOPSIS

`dbobject` `sync`

If the table is in memory only, this method has no effect and will always succeed. This method returns zero (0) on success.

Fetching Tables

Database tables can be fetched by the `get` and `seq` methods.

`get` returns only one record if the fully qualified key can be found. For more information, see the description of the `get` method on page 171.

`seq` can be used to fetch tables sequentially. If the `key` argument is not given, the whole table is fetched (records to be retrieved can be filtered with patterns). `key` limits the records that should be fetched. `seq` may begin at any time, and the position of the cursor is not affected by calls to the `del`, `get`, `put`, or `sync` methods. Use the optional filters `-end`, `-glob`, `-nocase`, `-regexp`, `-result_filter`, and `-strstr` to limit the retrieved records.

Using `key` assures the best performance because it already limits the records that should be fetched while the filters limit the results only after the records have been fetched.

By fetching, the view (if any) assigned to the table while it was open is always processed.

Fetch Methods

Database tables can be fetched sequentially using `seq` or with indexed `get` access.

seq

SYNOPSIS

```
dbobject seq ?-columns column_list? ?-data? ?-end pattern?
    ?-first? ?-format format_string? ?-glob pattern? ?-key?
    ?-nocase pattern? ?-regexp expression?
    ?-result_filter pattern? ?-strstr pattern? ?-uniq? ?-list?
    ?key? ?flags?
```

`-columns column_list`

This switch determines the order and format in which the fields of the records are to be retrieved. `column_list` is a Tcl list with the format:

```
{{col_num1 ?format?} {col_num2 ?format?}...
{column_numbern ?format?}}
```

where `format` may be one of the following:

`/`	Only the remainder of the field after the last `/` character is retrieved.
`#separator`	The remainder of the field after the last `/` character is retrieved, and then `separator` and the field contents, until the last `/`. If the field doesn't contain any `/`, only the field contents are retrieved.
`%`	The field is formatted with the `sprintf` C Programming function as if it were: `sprintf(result,format,field);`
`&`	The formatted result up to the current field is formatted as if it were: `sprintf(new_result, format, current_result);` The `&` character is interpreted as if it were `%`.
`:biteq:bitor`	Records are retrieved only if the current field contents fulfill the condition as follows: `(biteq & field) == biteq && (bitor & field)` Use this operator only for numerical fields.

`=value`	Records are retrieved only if the current field equals to `value`.
`-min:max`	Records are retrieved only if the current field is between `min` and `max`. The field is in the range if it equals to `min` or `max`. This operator can be used only for decimal numerical fields.
`<value`	Records are retrieved only if the current field is less than `value`. Use this operator only for numerical fields.
`>value`	Records are retrieved only if the current field is greater than `value`. Use this operator only for numerical fields.
`|suffix`	A suffix is added to the contents of the field.

If `format` does not match any of the special characters listed above, it will just be appended after the field contents.

In `column_list`, every field can be defined only once.

`-data`	When this switch is specified, the fetched data of the record is not retrieved.
`-end pattern`	Only those records are fetched whose keys end with `pattern`.
`-first`	When this switch is specified, only the first record to match the conditions is retrieved.
`-format format_string`	If this option is specified, every retrieved record is additionally formatted with the `sprintf` C Programming function as if it were: `sprintf(new_result,format_string,result);`
`-glob pattern`	The record is retrieved only if its key matches the `glob` pattern.
`-key`	If this key is specified, the keys of the selected records are not retrieved.
`-nocase pattern`	The record is retrieved only if its key matches the non-case-sensitive `glob` pattern.
`-regexp expression`	The record is retrieved only if its key matches the regular expression.
`-result_filter pattern`	The record is retrieved only if the formatted fields match the `glob` pattern.
`-strstr pattern`	The record is fetched only if the pattern can be found in the key of the fetched record. (For more information, see the `strstr` C library function).
`-uniq`	If this option is specified, duplicated lists in the result returned are eliminated.

`-list`	If neither the `column`, `data`, or `key` switch is used, the keys and data parts of the fetched records are retrieved in separated sub-Tcl list. If this switch is specified, the key and data parts are retrieved in the same lists.
`flags`	The *`flags`* value may be set to one of these values:

`R_CURSOR`	The data associated with the specified key is returned. This differs from the `get` routines in that it sets or initializes the cursor to the location of the key as well. (For the `btree` access method, the returned key is not necessarily an exact match for the specified key. The returned key is the smallest key greater than or equal to the specified key, permitting partial key matches and range searches.)
`R_FIRST`	The first key/data pair of the database is returned, and the cursor is set or initialized to reference it.
`R_LAST`	The last key/data pair of the database is returned, and the cursor is set or initialized to reference it (applicable only to the `btree` tables.)
`R_NEXT`	Retrieve the key/data pair immediately after the cursor. If the cursor is not yet set, this is the same as the `R_FIRST` flag. This value is taken if a flag is not given.
`R_PREV`	Retrieve the key/data pair immediately before the cursor. If the cursor is not yet set, this is the same as the `R_LAST` flag (applicable only to the `btree` tables). `R_LAST` and `R_PREV` are available only for the `btree` tables because they each imply that the keys have an inherent order that does not change.

C Programming API Functions

This section provides the equivalent of manual pages for the C Programming API functions.

NOTE You must `#include <db.h>` when using the C API, and you need to link the final executable with `libpafdb.a` and `libtcl8.1.a` to locate the database library routines.

dbopen for C

SYNOPSIS

```
#include <db.h>

DB *dbopen(const char *file, int flags, int mode, DBTYPE type, const
void *openinfo);
```

`dbopen` is the library interface to database files.

`dbopen` opens *`file`* for reading and/or writing. Files that are never intended to be preserved on disk may be created by setting the file parameter to `NULL`.

The `flags` and `mode` arguments are as specified to the `open(2)` routine; however, only the `O_CREAT`, `O_EXCL`, `O_EXLOCK`, `O_NONBLOCK`, `O_RDONLY`, `O_RDWR`, `O_SHLOCK`, and `O_TRUNC` flags are meaningful. (Note that opening a database file `O_WRONLY` is meaningless.)

The `type` argument is of type `DBTYPE`, defined in the `<db.h>` include file. It may be set to `DB_BTREE` or `DB_HASH`.

The `openinfo` argument is a pointer to an access method specific structure described in the `access` method's manual page. If `openinfo` is NULL, each access method will use defaults appropriate for the system and the access method.

The `dbopen` routine returns a pointer to a DB structure on success and NULL on error. The DB structure is defined in the `<db.h>` include file, and contains at least these fields:

```
typedef struct {
    DBTYPE type;
    int (*close)(const DB *db);
    int (*del)(const DB *db, const DBT *key, u_int flags);
    int (*fd)(const DB *db);
    int (*get)(const DB *db, DBT *key, DBT *data, u_int flags);
    int (*put)(const DB *db, DBT *key, const DBT *data, u_int flags);
    int (*sync)(const DB *db, u_int flags);
    int (*seq)(const DB *db, DBT *key, DBT *data, u_int flags);
} DB;
```

The elements of this structure specify the database type and a set of functions required to perform operations on a database of this type. These functions take a pointer to a structure as returned by `dbopen`, and sometimes one or more pointers to key/data structures and a flag value.

The structure elements are:

`type`	The type of the underlying access method (and file format).
`close`	A pointer to a routine to flush any cached information to disk, free any allocated resources, and close the underlying file(s). Since key/data pairs may be cached in memory, failing to sync the file with a `close` or `sync` function may result in inconsistent or lost information. `close` routines return -1 on error (setting `errno`), and 0 on success.

`del`	A pointer to a routine to remove key/data pairs from the database.

The parameter `flags` may be set to:

`R_CURSOR`	Delete the record referenced by the cursor. The cursor must have previously been initialized.

`del` routines return -1 on error (setting `errno`), 0 on success, and 1 if the specified `key` was not in the file.

`fd`	A pointer to a routine that returns a file descriptor representative of the underlying database. A file descriptor referencing the same file is returned to all processes that call `dbopen` with the same filename. This file descriptor may be safely used as an argument to the `fcntl(2)` and `flock(2)` locking functions. The file descriptor is not necessarily associated with any of the underlying files used by the access method. No file descriptor is available for use in memory databases. `fd` routines return -1 on error (setting `errno`), and the file descriptor on success.
`get`	A pointer to a routine that is the interface for keyed retrieval from the database. The address and length of the data associated with the specified `key` are returned in the structure referenced by `data`. `get` routines return -1 on error (setting `errno`), 0 on success, and 1 if the `key` was not in the file.
`put`	A pointer to a routine to store key/data pairs in the database.

The parameter `flags` may be set to one of these values:

`R_CURSOR`	Replace the key/data pair referenced by the cursor. The cursor must have previously been initialized.
`R_NOOVERWRITE`	Enter the new key/data pair only if the key does not previously exist.

`R_SETCURSOR` is available only for the `DB_BTREE` access method because it implies that the keys have an inherent order that does not change.

The default behavior of the `put` routines is to enter the new key/data pair, replacing any previously existing key.

`put` routines return -1 on error (setting `errno`), 0 on success, and 1 if the `R_NOOVERWRITE` flag was set and the key already exists in the file.

`seq`	A pointer to a routine that is the interface for sequential retrieval from the database. The address and length of the key are returned in the structure referenced by `key`, and the address and length of the data are returned in the structure referenced by `data`.

Sequential key/data pair retrieval may begin at any time, and the position of the “cursor” is not affected by calls to the `del`, `get`, `put`, or `sync` routines. Modifications to the database during a sequential scan are reflected in the scan, that is, records inserted behind the cursor are not returned while records inserted in front of the cursor are returned.

The `flags` value must be set to one of these values:

`R_CURSOR`	The data associated with the specified key is returned. This differs from the `get` routines in that it sets or initializes the cursor to the location of the key as well. (For the `DB_BTREE` access method, the returned key is not necessarily an exact match for the specified key. The returned key is the smallest key greater than or equal to the specified key, permitting partial key matches and range searches.)
`R_FIRST`	The first key/data pair of the database is returned, and the cursor is set or initialized to reference it.
`R_LAST`	The last key/data pair of the database is returned, and the cursor is set or initialized to reference it. (Applicable only to the `DB_BTREE` access method.)
`R_NEXT`	Retrieve the key/data pair immediately after the cursor. If the cursor is not yet set, this is the same as the `R_FIRST` flag.
`R_PREV`	Retrieve the key/data pair immediately before the cursor. If the cursor is not yet set, this is the same as the `R_LAST` flag. (Applicable only to the `DB_BTREE` access method.)

`R_LAST` and `R_PREV` are available only for the `DB_BTREE` access method because they each imply that the keys have an inherent order that does not change.

`seq` routines return -1 on error (setting `errno`), 0 on success, and 1 if there are no key/data pairs less than or greater than the specified or current key.

`sync`	A pointer to a routine to flush any cached information to disk. If the database is in memory only, the `sync` routine has no effect and always succeeds.

The `flags` value may be set to this value:

`sync` routines return -1 on error (setting `errno`), and 0 on success.

Key/Data pairs

Access to all file types is based on key/data pairs. Both keys and data are represented by this data structure:

```
typedef struct {
  void *data;
  size_t size;
} DBT;
```

The elements of this structure are defined as follows:

`data`	A pointer to a byte string.
`size`	The length of the byte string.

Key and data byte strings may reference strings of essentially unlimited length although any two of them must fit into available memory at the same time. It should be noted that the access methods provide no guarantees about byte string alignment.

Errors

The `dbopen` routine may fail and set `errno` for any of the errors specified for the library routines `open(2)` and `malloc(3)` or the following:

`[EFTYPE]`	A file is incorrectly formatted.
`[EINVAL]`	A parameter, such as hash function or pad byte, has been specified that is incompatible with the current file specification, or which is not meaningful for the function (for example, use of the cursor without prior initialization), or there is a mismatch between the version number of the file and the software.

The `close` routines may fail and set `errno` for any of the errors specified for the library routines `close(2)`, `read(2)`, `write(2)`, `free(3)`, or `fsync(2)`.

The `del`, `get`, `put`, and `seq` routines may fail and set `errno` for any of the errors specified for the library routines `read(2)`, `write(2)`, `free(3)`, or `malloc(3)`.

The `fd` routines will fail and set `errno` to `ENOENT` for any of the errors specified in memory databases.

The `sync` routines may fail and set `errno` for any of the errors specified for the library routine `fsync(2)`.

Limitations

None of the access methods provides any form of concurrent access, locking, or transactions.

`btree` Database Access Method

SYNOPSIS

```
#include <db.h>
```

The routine `dbopen` is the library interface to database files. One of the supported file formats is `btree` files. For a general description of the database access methods, see `dbopen(3)`; this describes only the `btree`-specific information.

The `btree` data structure is a sorted, balanced tree structure storing associated key/data pairs.

The `btree` access method specific data structure provided to `dbopen` is defined in the `<db.h>` include file as follows:

```
typedef struct {
  u_long flags;
  u_int cachesize;
  int maxkeypage;
  int minkeypage;
```

```
    u_int psize;
    int (*compare)(const DBT *key1, const DBT *key2);
    size_t (*prefix)(const DBT *key1, const DBT *key2);
    int lorder;
  } BTREEINFO;
```

The elements of this structure are as follows:

`flags`
: The `flags` value is specified by applying the bitwise OR operation with the following value:

  `R_DUP`
  : Permit duplicate keys in the tree, that is, permit insertion if the key to be inserted already exists in the tree. The default behavior, as described in `dbopen(3)`, is to overwrite a matching key when inserting a new key or to fail if the `R_NOOVERWRITE` flag is specified. The `R_DUP` flag is overridden by the `R_NOOVERWRITE` flag, and if the `R_NOOVERWRITE` flag is specified, attempts to insert duplicate keys into the tree will fail.
  If the database contains duplicate keys, the order of retrieval of key/data pairs is undefined if the `get` routine is used; however, `seq` routine calls with the `R_CURSOR` flag set will always return the logical “first” of any group of duplicate keys.

`cachesize`
: A suggested maximum size (in bytes) of the memory cache. This value is only advisory, and the access method will allocate more memory rather than fail. Since every search examines the root page of the tree, caching the most recently used pages substantially improves access time. In addition, physical writes are delayed as long as possible, so a moderate cache can reduce the number of I/O operations significantly. Using a cache slightly increases the likelihood of corruption or lost data if the system crashes while a tree is being modified. If `cachesize` is 0 (no size is specified), a default cache is used.

`maxkeypage`
: The maximum number of keys stored on any single page. Not currently implemented.

`minkeypage`
: The minimum number of keys stored on any single page. This value determines which keys are stored on overflow pages; that is, if a key or data item is longer than the pagesize divided by the `minkeypage` value, it is stored on overflow pages instead of in the page itself. If `minkeypage` is 0 (no minimum number of keys is specified), a value of 2 is used.

`psize`
: `psize` is the size (in bytes) of the pages used for nodes in the tree. The minimum page size is 512 bytes and the maximum page size is 64K. If `psize` is 0 (no page size is specified) a page size is chosen based on the underlying file system I/O block size.

`compare`
: `compare` is the key comparison function. It must return an integer less than, equal to, or greater than zero if the first key argument is considered to be respectively less than, equal to, or greater than the second key argument. The same comparison function must be used on a given tree every time it is opened. If `compare` is NULL (no comparison function is specified), the keys are compared lexically, with shorter keys considered less than longer keys.

`prefix`	`prefix` is the prefix comparison function. If specified, this routine must return the number of bytes of the second key argument, which is necessary to determine that it is greater than the first key argument. If the keys are equal, the key length should be returned. Although the usefulness of this routine is very data dependent, in some data sets it can produce significantly reduced tree sizes and search times. If `prefix` is NULL (no prefix function is specified) *and* no comparison function is specified, a default lexical comparison routine is used. If `prefix` is NULL and a comparison routine is specified, no prefix comparison is done.
`lorder`	The byte order for integers in the stored database metadata. The number should represent the order as an integer; for example, big endian order would be the number 4,321. If `lorder` is 0 (no order is specified), the current host order is used.

If the file already exists (and the `O_TRUNC` flag is not specified), the values specified for the parameters `flags`, `lorder`, and `psize` are ignored in favor of the values used when the tree was created.

Forward sequential scans of a tree are from the least key to the greatest.

Space freed up by deleting key/data pairs from the tree is never reclaimed, although it is normally made available for reuse. This means that the `btree` storage structure is grow-only. The only solutions are to avoid excessive deletions, or to create a fresh tree periodically from a scan of an existing one.

Searches, insertions, and deletions in a `btree` will all complete in `O (ln N)` where `N` is the average fill factor. Often, inserting ordered data into a `btree` results in a low fill factor. This implementation has been modified to make ordered insertion the best case, resulting in a much better-than-normal page fill factor.

Errors

The `btree` access method routines may fail and set `errno` for any of the errors specified for the library routine `dbopen(3)`.

Hash Database Access Method

SYNOPSIS

```
#include <db.h>
```

The routine `dbopen` is the library interface to database files. One of the supported file formats is hash files. For a general description of the database access methods, see `dbopen(3)`; this describes only the hash specific information.

The hash data structure is an extensible, dynamic hashing scheme.

The access method specific data structure provided to `dbopen` is defined in the `<db.h>` include file as follows:

```
typedef struct {
  u_int bsize;
  u_int ffactor;
```

```
    u_int nelem;
    u_int cachesize;
    u_int32_t (*hash)(const void *, size_t);
    int lorder;
  } HASHINFO;
```

The elements of this structure are as follows:

`bsize`	`bsize` defines the hash table bucket size, and is, by default, 256 bytes. It may be preferable to increase the page size for disk-resident tables and tables with large data items.
`ffactor`	`ffactor` indicates a desired density within the hash table. It is an approximation of the number of keys allowed to accumulate in any one bucket, determining when the hash table grows or shrinks. The default value is 8.
`nelem`	`nelem` is an estimate of the final size of the hash table. If not set or set too low, hash tables will expand gracefully as keys are entered, although a slight performance degradation may be noticed. The default value is 1.
`cachesize`	A suggested maximum size, in bytes, of the memory cache. This value is *only* advisory, and the access method will allocate more memory rather than fail.
`hash`	`hash` is a user-defined hash function. Since no hash function performs equally well on all possible data, the user may find that the built-in hash function performs poorly on a particular data set. User-specified hash functions must take two arguments (a pointer to a byte string and a length) and return a 32-bit quantity to be used as the hash value.
`lorder`	The byte order for integers in the stored database metadata. The number should represent the order as an integer; for example, big endian order would be the number 4,321. If `lorder` is 0 (no order is specified), the current host order is used. If the file already exists, the specified value is ignored, and the value specified when the tree was created is used.

If the file already exists (and the `O_TRUNC` flag is not specified), the values specified for the parameters `bsize`, `ffactor`, `lorder`, and `nelem` are ignored and the values specified when the tree was created are used.

If a hash function is specified, `hash_open` attempts to determine if the hash function specified is the same as the one with which the database was created, and will fail if it is not.

Backwardly compatible interfaces to the routines described in `dbm(3)` and `ndbm(3)` are provided; however, these interfaces are not compatible with previous file formats.

Errors

The `hash` access method routines may fail and set `errno` for any of the errors specified for the library routine `dbopen(3)`.

Simple Query Tool

The example below shows a simple query tool written in the C programming language. Note that it works only for `btree` tables and that views are not supported.

```
#include <db.h>

main(int argc,char *argv[])
{
  DB    *db;
  DBT   data,key;
  int   flag,len;
  char  *pattern;
  if (argc != 3)
  {
    printf("usage: %s database pattern\n",argv[0]);
    exit(1);
  }
  if (!(db = dbopen(argv[1],O_RDONLY,0644,DB_BTREE,NULL)))
  {
    fprintf(stderr,"Could not open \
        \"%s\",%s\n",argv[1],
      strerror(errno));
    exit(2);
  }
  pattern = argv[2];
  len = strlen(pattern);
  key.data = (void *)pattern;
  key.size = len;
  for(flag = R_FIRST;
    db->seq(db,&key,&data,flag) == 0 &&
    strncmp(key.data,pattern,len) == 0; flag = R_NEXT)
  {
    printf("key:  %s\n",key.data);
    printf("data: %s\n",data.data);
  }
  db->close(db);
  exit (0);
}
```

To compile and link you can use the following Makefile:

```
SDK=/export/home/tom/snavigator/sdk

CFLAGS=  -I$(SDK)/include
LIB=     -L$(SDK)/lib -lpafdb

dbqry:    dbqry.c
  $(CC) -o $@ $< $(LIB)
```

Database Table Structures

Source-Navigator stores information about source files in project (database) tables to assure high performance with flexible query possibilities.

With the exception of the project file (that is itself also a hash database table), every table normally relies on the `.snprj` sub-directory of the project and can be accessed like any other database table. The following example shows what the table structure of a project database looks like on a UNIX system. It was produced using the shell command `ls -l .snprj`.

```
-rw-r--r--   1 user sys        16384 Aug 12 12:19 cpl.1
-rw-r--r--   1 user sys        16384 Aug 12 12:34 cpl.2
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.by
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.cl
-rw-r--r--   1 user sys        16384 Aug 12 12:19 cpl.f
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.fil
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.fu
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.gv
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.iv
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.md
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.mi
-rw-r--r--   1 user sys         8192 Aug 12 12:19 cpl.to
```

In the following table, the symbol ‡ represents the `sn_sep` separator character. See the scripts in “Cross-Reference Tables” on page 166 for more information on this variable. Additionally, all of the following keys must be on one line.

The hash # character in class names means that the symbol does not belong to any classes, and semicolon (`;`) separates the key and data parts. Positions consist of line and column numbers separated by a comma (`,`).

Table 14: Database Table Structures

File Suffix	*Table Type*	*Table Description*	*Record Format*
0	hash	Ignored words	word;#
1	hash	Default view	filename;#
2	hash	Second view	filename;#
by	btree	Referred by	ref-class‡ref-symbol-name‡ref-type‡class‡symbol‡type‡access‡position‡filename;\ {caller_argument_types}‡{ref_argument_types}
cl	btree	Classes	name‡start_position‡filename;end_position‡attributes‡{}‡{class template}‡{}‡{comment}
com	btree	Common blocks	name‡start_position‡filename;end_position‡attributes‡{}‡{}‡{}‡{comment}
con	btree	Constants	name‡start_position‡filename;end_position‡attributes‡{dec_type}‡{}‡{}‡{comment}
cov	btree	Common value	common-block‡name‡start_position‡filename;end_position‡attributes‡{}‡{}‡{}‡{comment}
e	btree	Enumerations	name‡start_position‡filename;end_position‡attributes‡{}‡{}‡{}‡{comment}
ec	btree	Enum-constants	name‡start_position‡filename;end_position‡attributes‡{}‡{}‡{}‡{comment}
f	btree	Project files	name;group‡parsing-time‡highlight-file
fd	btree	Function	name‡start_position‡filename;end_position‡attributes‡{ret_type}‡{arg_types}‡{arg_names}‡\ {comment}

Table 14: Database Table Structures (Continued)

File Suffix	*Table Type*	*Table Description*	*Record Format*
fil	btree	Symbols of files	filename‡start_position‡class‡identifier‡type;end_position‡high_start_pos‡high_end_pos‡\ arg_types
fr	btree	Friends	name‡start_position‡filename;end_position‡attributes‡{ret_type}‡{arg_types}‡{arg_names}‡\ {comment}
fu	btree	Functions	name‡start_position‡filename;end_position‡attributes‡{ret_type}‡{arg_types}‡{arg_names}‡\ {comment}
gv	btree	Variables	name‡position‡filename;attributes‡{type}‡{template‡parameter}‡{comment}
in	btree	Inheritances	class‡base-class‡start_position‡filename;end_position‡attributes‡{}‡{class template}‡\ {inheritance‡template}‡{comment}
iu	btree	Include	included_file‡start_position‡include_from_file;0.0‡0x0‡{}‡{}‡{}‡{}
iv	btree	Instance variables	class‡variable-name‡start_position‡filename;end_position‡attributes‡{type}‡{}‡{}‡{comment}
lv	btree	Local variables	function‡variable-name‡start_position‡filename;end_position‡attributes‡{}‡{type}‡{}‡\ {comment}
ma	btree	Macros	name‡start_position‡filename;end_position‡attributes‡{}‡{}‡{}‡{comment}
md	btree	Method definitions	class‡name‡start_position‡filename;end_position‡attributes‡{ret_type}‡{arg_types}‡\ {arg_names}‡{comment}
mi	btree	Method implementations	class‡name‡start_position‡filename;end_position‡attributes‡{ret_type}‡{arg_types}‡\ {arg_names}‡{comment}
rem	btree	Remarks	filename‡position‡class‡method_or_function;comment
su	btree	Subroutines	name‡position‡filename;attributes‡{}‡{}‡{comment}
t	btree	Typedefs	name‡position‡filename;attributes‡{original}‡{}‡{comment}
to	btree	Refers to	class‡symbol-name‡type‡ref-class‡ref-symbol‡ref-type‡access‡position‡filename;\ {caller_argument_types}‡{ref_argument_types}
un	btree	Unions	name‡position‡filename;attributes‡{}‡{}‡{comment}

Database API Program Examples

The Tcl script below opens a table for a fictitious Source-Navigator project named `pure`.

```
#!/bin/sh
# Replace $HOME/snavigator with the Source-Navigator
# installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don’t forget the backslash before exec!
#
set db_functions [dbopen nav_func .snprj/pure.fu RDONLY \
    0644 btree cachesize=200000]
# Output the list of matches with newline characters after
# each.
puts [join [$db_functions seq] \n]
```

This shell script produces the following result:

```
{chk 000011.012 chk.c} {17.1 0x8 {void} {int} {size} {}}
{fnc1 000019.012 chk.c} {33.1 0x8 {void} {int,int,char *}
    {i,size,str} {}}
{keys 000026.005 keybind.tcl} {28.1 0x0 {} {} {k} {}}
{main 000035.000 chk.c} {38.1 0x0 {int} {} {} {}}
```

Each record contains two Tcl lists: the first is the key part, the second is the data part. If `-key`, `-data`, or `-columns` is used, the key and the data parts are always retrieved in separate Tcl lists.

If you use the `-data` switch in the `fetch` command (such as `$db_functions seq -data`) in the example script above, only the key is fetched and the result is the following:

```
chk 000011.012 chk.c
fnc1 000019.012 chk.c
keys 000026.005 keybind.tcl
main 000035.000 chk.c
```

To restrict the result to functions whose names begin with `main`, use the following command:

```
$db_functions seq -data "main"
```

To fetch only the functions with the name `main`, you should add a blank to the key value:

```
$db_functions seq -data "main$sn_sep"
```

The `-columns` switch can be used to change the order of fields. The query below retrieves the name of the files, then the name of the functions and their positions:

```
$db_functions seq -data -columns [list 2 0 1]
```

The result is:

```
chk.c chk 000011.012
chk.c fnc1 000019.012
keybind.tcl keys 000026.005
chk.c main 000035.000
```

As described on page 173, the Tcl list following `-columns` contains sub-lists. The first element of a sub-list identifies (offset number beginning from 0) which field should be retrieved, and the second element is an optional format that controls formatting. In the example below, the `""` appends a blank after every retrieved field. Sometimes it is useful to use `\t` (tab) instead of blanks. In Source-Navigator, types are often shown in parentheses.

For example, to obtain a listing of every function, indicated as (`fu`), the following command

```
$db_functions seq -data -columns [list {2 "(fu) "} 0 1]
```

produces the result:

```
chk.c(fu) chk 000011.012
chk.c(fu) fnc1 000019.012
keybind.tcl(fu) keys 000026.005
chk.c(fu) main 000035.000
```

The format characters `:`, <, >, and = can be used to make comparisons of database table field contents. When a condition is not true, the record is not retrieved.

For the purposes of this example, the source file where the following C++ class `TEST` is defined will be called `test.cc`.

```
class TEST {
private:
  int inm()
  {
    return 0;
  }
  outside(int x,int y);

public:
  static void copy(){}

protected:
  int var;
};

TEST::outside(int x,int y)
{
}
```

The Tcl script below queries the project for all of the public methods of classes that have been defined in `test.cc`. The value four used in the example maps to the constant `PAF_PUBLIC` from `sn.h`.

```
#!/bin/sh
# Replace $HOME/snavigator with the Source-Navigator
# installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don't forget the backslash before exec!
#

set db_prefix .sn/doc.md
set db [dbopen methods_db $db_prefix RDONLY 0444 btree]

set res [$db seq -col [list 0 1 3 2 "5 :0:4"] -end "test.cc"]
# 4 -> 1 for private methods, 4 -> 5 for public and private methods

puts [join $res "\n"]
```

The script produces this output:

```
TEST copy test.cc 000009.014 0x200c
```

To query the private methods, change the value four to one (the value of `PAF_PUBLIC`).

This modified script produces this output:

```
TEST inm test.cc 000003.006 0x2001
TEST outside test.cc 000007.002 0x1
```

To query the public and private methods, use the value five. This is the bitwise OR of the values for `PAF_PRIVATE` and `PAF_PUBLIC`.

To query all static (`SN_STATIC`) methods defined in `test.cc`, change the script as follows:

```
set res [$db seq -col [list 0 1 3 2 "5 :8:7"] -end "test.cc"]
```

To query every method between lines seven and nine in `test.cc`, make the following query:

```
set res [$db seq -col [list 0 1 3 \
"2 <10" "2 >6"] -end "test.cc"]
```

Database Application Examples

The Source-Navigator installation contains a number of larger examples for useful tools that can be quickly realized using the database API. They are located in `~/share/sdk/api/tcl/database/examples`.

Source-Navigator can assist in a wide variety of software engineering and re-engineering tasks and these examples tend to address the common scenario of bringing under control inherited bodies of source code that may be poorly written and poorly understood.

These examples are all written in the Tcl programming language. Some examples utilize the Tk toolkit. None of the examples require that Source-Navigator be running in order to use them. They work on the database directly using the database API provided by the `hyper` interpreter that comes with Source-Navigator.

NOTE At the top of each script is a path to the interpreter that may need to be edited to locate `hyper` on your system.

Most of the examples require at least two command line arguments: the path to the Source-Navigator project directory and the name of the project you're interested in. More details can be found in the comment block at the top of each script file, and each script is quite heavily documented.

Scripts

The example scripts are described below.

multicludes.tcl

This tool reports on redundant header files. By reducing `#include` complexity in a source file, compilation time can be reduced. This tool locates simple duplication, whereby `foo.c` may include `bar.h` (such as `#include "bar.h"`) and then `bar.h` again later. By optionally specifying a `-transitive` command line argument to the script, a more thorough search through the header file graph is performed, such that includes of `stdio.h` may be detected as unnecessary if another included header file includes it on your behalf.

diamonds.tcl

This tool locates multiple inheritance “diamonds” in the class hierarchy of a project written in an object-oriented language like C++. In his book, ***Effective C++***[4], Scott Myers points out the dangers associated with class hierarchies in which two classes derived from the same superclass are inherited by a fourth derived-most class. Diamonds are universally considered to be poor C++ programming practice and this tool can locate them in a Source-Navigator project.

call-freq.tk

This tool plots the caller/callee frequencies for all functions and methods in a project. Functions appearing to have called many functions or that are called by many functions may be ones requiring coverage testing, additional documentation, optimization, etc.

Each function is represented as a point on a graph. Clicking on a point opens a list box showing the name of the function and the caller/callee statistics.

clobber.tcl

This tool shows the names of all functions/methods in a project that modify a particular global variable.

constants.tcl

This tool identifies global variables in projects which are accessed as read-only objects. These variables are therefore candidates for becoming constants.

unimp-methods.tcl

This tool locates class method definitions that are surplus to a class, for which there is no method implementation. This tool is not always accurate as it will also suggest that methods that are defined inline are not implemented when they actually are.

unused.tcl

This tool determines where unused global variables exist in a project.

[4] Meyers, Scott. 1997. ***Effective C++: 50 Specific Ways to Improve Your Programs and Designs***. 2nd ed. ISBN 0-20-192488-9.

5

Database Utilities

The Source-Navigator utilities `dbdump`, `dbcp`, and `dbimp` access and maintain databases that can be started from user shells or applications. These utilities are located in the `~/bin` directory.

NOTE The scripts in this chapter are written to work on the UNIX operating system. When running on DOS, the scripts must be changed.

dbdump

The `dbdump` utility provides a complete listing, or a dump, of the contents of a database table. Its usage is:

```
dbdump ?-separator char? file
```

`dbdump` separates the key and data parts with a semicolon (`;`). The hash character (`#`) and brackets (`{}`) indicate an empty field. The following shell script lists the name and locations of the project class.

```
#!/bin/sh

dbdump=$HOME/snavigator/bin/dbdump
project=tmp
```

```
$dbdump -s ’ ’ $project.cl | awk ’{printf
    "%s,%s,%s\n",$1,$3,$2}’
```

dbcp

The `dbcp` utility copies and compacts the database. Space freed up by deleting key/data pairs from `btree` tables is never reclaimed, although it is normally made available for reuse. When copying a database with `dbcp`, deleted records are not copied, resulting in a much better page-fill factor and reduced disk space requirements. Its usage is:

```
dbcp input_table output_table
```

The following shell script compresses the tables of a Source-Navigator project.

```
#!/bin/sh

execdir=$HOME/snavigator/bin
dbcp=$execdir/dbcp

project=TEST
cd .paf

for i in $project.[a-z]*
do
  $dbcp $i $$.tmp
  mv $$.tmp $i
done
```

You can use this script even for currently-running Source-Navigator projects because compressing database tables does not affect performance. Hash tables may also be copied with `dbcp`, but there will be no space savings.

dbimp

The `dbimp` utility inserts, updates, and deletes records in a project database. It reads commands from its standard input.

Its usage is:

```
dbimp ?-c cache_size? ?-C cross_cache_size?
          ?-l? ?-f file? db_prefix
```

References to local variables are stored only if the `-l` flag is specified.

`db_prefix` contains the name of the database directory, for example, `.sn` and the base name of the project file. If the project file is called `test.proj`, `db_prefix` could be called `.snprj/test`.

The format of the commands (read from standard input) must follow this syntax:

```
COMMAND;KEY;DATA
```

If COMMAND is greater than or equal to 0, dbimp inserts key/data pairs. The value of COMMAND must be between PAF_FILE and PAF_COMMENT_DEF, inclusive. (For the numerical values see sn.h.)

The following example inserts a function (strcopy) definition.

```
8;strcopy 000001.004 x.c;4.1 0x0 {int} {} {} {}
```

The following example inserts a method definition (pro3) of the class xharom.

```
17;xharom pro3 000036.005 x.C;36.11 0x2 {int} {} {} {}
```

The instructions below insert cross-references (refers-to and referred-by) into the project database. The ‡ symbol represents the sn_sep separator character in the instructions below.

```
16;#‡abc‡fu‡abc‡var‡lv‡w‡000004‡x.c;#
15;abc‡var‡lv‡#‡abc‡fu‡w‡000004‡x.c;#
16;#‡abc‡fu‡#‡hello‡fu‡p‡000005‡x.c;#
15;#‡hello‡fu‡#‡abc‡fu‡p‡000005‡x.c;#
```

See “Database Table Structures” on page 184 for information on the structure of the Source-Navigator database tables.

- If COMMAND is 0, KEY must contain a known source file name of the project, and dbimp deletes the definitions of, but not cross-references to, the file.
- If COMMAND is -1, KEY must be 0 and DATA must contain a known source file name of the project, and dbimp deletes the definitions of, but not cross-references to, the file.
- If COMMAND is -2, KEY must be 0 and DATA must contain a known source file name of the project, and dbimp deletes the cross-references to, but not definitions of, the file.

The following example deletes the cross-reference information for the file x.c:

```
-2;0;x.c
```

Limitations

Undefined results occur if the COMMAND does not have a legal value.

In the commands, you may use only single blank spaces and no tabs.

6

Integrating with Version Control Systems

Source-Navigator's user interface provides a standard set of version control operations. The SDK allows third parties to integrate version control packages so that some or all of these operations can be performed from within Source-Navigator. This chapter assumes a basic understanding of the Tcl programming language and familiarity with the version control system that is to be integrated with Source-Navigator.

Basics

The version control interface in Source-Navigator assumes that there is a basic set of version control operations in the context of source code comprehension:

- obtaining different versions of a source file from a repository or archive maintained by the version control system. This operation will be referred to as a *check-out*.
- examining the differences between the current working version of a source file and another version held under version control. This operation will be referred to as a *diff*.

- making changes to the most current version of a file and placing those changes under version control. This operation will be referred to as a *check-in*.
- deleting versions from the repository.
- acquiring exclusive access to a particular version of a file so that no other developer can make changes to that version. This operation will be referred to as *locking*.
- relinquishing exclusive access to a particular version of a file so that other developers may make modifications. This operation will be referred to as *unlocking*.
- discarding any changes to the current working version and reverting to the most recent version in the repository.
- obtaining the history of a file from the version control system. This includes:
    - descriptive text about the changes made for each version.
    - a list of symbolic names for the versions associated with a file. In GNU RCS, for example, these names are known as *tags*.

Version Control Operations

Source-Navigator takes a universal approach to interfacing with external version control systems: each operation is performed by executing a particular command line and capturing the output from that command.

For some operations, the output of the command is significant and only information relevant to the user must be extracted from the output. Examples of this include obtaining a history of changes made to a file, obtaining the names of symbolic tags, and obtaining all of the version numbers associated with a file. The SDK provides a mechanism based on Tcl regular expressions for extracting the relevant text from command output.

The Configuration File

The Source-Navigator configuration file, `~/share/etc/sn_prop.cfg`, is read at start-up and can be used to customize Source-Navigator. In particular, new version control systems may be integrated from within this file. New systems are added by calling a Tcl procedure called `sn_add_version_control_system`.

The signature of this procedure is:

```
sn_add_version_control_system {ident args}
```

where

`ident` is an internal identifier used by Source-Navigator. This should be a logical

derivation of the name of the version control system and consist only of alphanumeric characters and start with an alphabetic character. Examples are:

```
rcs
cvs
ccase3
```

`args` is a variable number of arguments and may be any of the options described in the Options section, followed by a suitable value. For example:

```
-checkout co
```

Options

For options that have simple values, the possible values are outlined and the default is given. If this default value is satisfactory, the option may be omitted from the call.

For options that specify command lines, the usage of the command is shown. If the option is not applicable to a particular command, omitting it from the call is permissible, but may result in some version control operations being unavailable to the user.

Using the `-checkout` example on page 198, the `checkout` command line is illustrated in the following notation:

`${checkout}` *`filename`*

At runtime, the command line that is executed might be:

```
co main.c
```

`args` may be any of the following options:

`-checkin`

This specifies the command line to check a modified file back into the repository.

Command usage:

`${checkin}` *`filename`* or

`${checkin}` *`comment-text filenames`* or

`${checkin}` *`comment-filename filenames`*

Which command is used depends on the value of `-checkin-comment-via`.

`-checkin-comment-via`

This option specifies how comments about changes made to a file must be passed to the `-checkin` command. The three possible values are:

`stdin`	The comment is issued on the checkin command’s standard input.
`file`	The command is placed into a temporary file and the filename is passed on the command line prior to the names of source files.
`cmdline`	The comment is placed on the command line prior to the filename(s).

Default:

None

`-checkin-exclusive`

This option specifies the command line to check a file into the repository and to acquire a lock once the check-in is complete. This is useful when you want to make successive changes to a file.

Command usage:

```
{$checkin-exclusive} filenames
```

`-checkout`

This option specifies the command line to check out the latest version of a file from the repository.

Command usage:

```
${checkout} filenames
```

`-checkout-exclusive`

This option specifies the command line to check out the latest version of a file from the repository with exclusive access that prevents other developers from modifying the file. Some version control systems require that a file be checked out exclusively before a modified version of it may be checked in.

Command usage:

```
${checkout-exclusive} filenames
```

`-checkout-individual`

This option specifies the command line to check out a particular version of a file from the repository.

Command usage:

```
${checkout-individual} version-num filenames
```

`-checkout-individual-to-stdout`

This option specifies the command line to check out a particular version of a file from the repository and echo the contents to standard output.

Command usage:

`${checkout-individual-to-stdout}` *`version-num filenames`*

`-checkout-with-lock`

If this option is set to `yes`, then checkout operations will by default attempt to acquire a lock to prevent other developers from making concurrent modifications. This option determines whether the `with lock` check button is initially set in the `Checkout` dialog box.

Default:

None.

`-default`

If this option is set to `yes`, then this version control system entry will be the default for new projects. If this option is set to `yes` for more than one entry in the configuration file, then the entry which appears last in the file will be the default.

Default:

None.

`-delete-revision`

This option specifies the command line to delete a particular version of a file from the repository.

Command usage:

`${delete-revision}` *`version-num filename`*

`-diff-command`

This option specifies the name of the command used to compare two files (or two versions of the same file).

Default:

`diff`

`-diff-ignore-case`

This option specifies any additional command line options to the `diff` command to cause it to perform case insensitive comparisons. For almost all implementations of the traditional `diff` command, this will be `-i`.

Default:

`-i`

`-diff-ignore-whitespace`

This option specifies any additional command line options to the `diff` command to cause it to perform comparisons that are insensitive to whitespace. For almost all implementations of the traditional `diff` command, this will be `-w`.

Default:

```
-w
```

`-discard`

This option specifies a command line to discard any modifications made to the working version of a file and revert it to the current version in the repository.

Command usage:

`${discard}` *`filenames`*

`-history`

This option specifies a command line to retrieve the history of a file. Typically this includes comments made by developers at each version of the file, the set of version numbers for the file and the time and date of each change.

Command usage:

`${history}` *`filename`*

`-history-pattern`

This option specifies a pattern to extract the version history from the output of `${history}`. It should extract all the relevant information about all versions of a file. See “Patterns” on page 202 for more detailed information about possible values for this option.

Default:

None.

`-history-individual`

This option specifies the command to retrieve the history for a particular version of a file.

Command usage:

`${history-individual}` *`version-num filename`*

`-history-individual-pattern`

This option specifies a pattern to extract the version history from the output of `${history-individual}`. It should extract all the relevant information about the particular revision of a file.

Default:

None.

`-history-replacements`

This option specifies a set of textual replacements to perform on text extracted by the patterns `${history-pattern}` and `${history-individual-pattern}`. This allows text to be manipulated so that it may be cosmetically improved before being shown in the **History** window.

See “Replacements” on page 203 for details about possible values for this option.

Default:

None.

`-ignore-dirs`

This option specifies one or more directories to ignore when presenting directory trees to the user. Such directories may include a repository subdirectory for each directory maintained by the version control system. When specifying more than one subdirectory, use the Tcl list notation. For example: `{foo bar}`

Default:

None.

`-lock`

This option specifies the command line to lock a file such that no other developer may make concurrent modifications.

Command usage:

`${lock}` *`filename`*

`-lock-individual`

This option specifies the command line to lock a particular version of a file such that no other developer may make concurrent modifications to that version of the file.

Command usage:

`${lock-individual}` *`version-num filename`*

`-revision-number-pattern`

This option specifies a pattern to extract all of the available revision numbers from the output of the `${history}` command.

Default:

None.

`-symbolic-tags-pattern`

This option specifies a pattern to extract the names of a file’s symbolic tags from the output of the `${history}` command.

Default:

None.

`-symbolic-tags-replacements`

This option specifies a set of textual replacements to perform on text that is extracted by the `${symbolic-tags-pattern}` option. This allows text to be manipulated so that it may be cosmetically improved before being shown in the **Symbolic tags** window.

Default:

None.

`-title`

This option specifies a descriptive label that will be presented to the user in the project preferences window. This option is useful for showing a mixed-case product name or one that contains whitespace.

Default:

the `ident` parameter (in upper-case).

`-unlock`

This option specifies the command line needed to unlock a file so that other developers may make concurrent modifications.

Command usage:

`${unlock}` *`filename`*

`-unlock-individual`

This specifies the command line to unlock a particular version of a file so that other developers may make concurrent modifications to that version of the file.

Command usage:

`${unlock-individual}` *`version-num filename`*

Patterns

Patterns are used to extract text resulting from certain version control commands. It is necessary to use patterns to filter relevant lines of text from the output.

A pattern is specified as either:

- A list of regular expression pairs specifying where to start and stop extracting text. The two keywords `start` and `end` are reserved and refer to the start and end of the body of text respectively, or

- An atomic regular expression specifying lines that will be extracted if the regular expression matches any text in the line.

Some examples of patterns are:

Extract the names of all the directories in `ls -l` output:

`^d` (an atomic regular expression)

Extract the names of all the files and directories in `ls -al` output, but exclude the current and parent directories (“.” and “..”):

`{ {".." end} }` (a list of one regular expression pair)

More sophisticated examples exist in the configuration example on page 204.

Replacements

Replacements may be used to modify or delete text that is extracted from the result of certain version control commands.

Where replacements are mentioned in the option descriptions above, legal values are a list of zero or more pairs. A pair is defined to be a list of exactly two elements. The left-hand side of the pair is a Tcl regular expression and the right-hand side is a literal string to replace the text matched by the regular expression.

An example of a replacement list is:

```
{ {"\t" " "} {"foo" "bar"} {"^-+$" ""} }
```

When applied to output from a version control command, this would cause:

- tabs to be replaced by spaces.
- all occurrences of `foo` to be replaced by `bar`.
- separation lines consisting of hyphens to be replaced by a blank line.

Scripts

Situations may arise in which a version control system will seem incompatible with Source-Navigator’s approach to constructing command lines and examining the output of the command.

In most cases, these issues are overcome by writing shell scripts that provide a wrapper interface that Source-Navigator works with, but internally issues commands that work with the version control system.

Consider a version control system that does not adhere to the convention of specifying the version number of a file prior to the filename on the command line. Systems such as RCS would expect a command line such as:

```
co -r1.5 main.c
```

But other systems might expect a command line like:

```
foocs checkout main.c/1.5
```

This could be handled by writing a small shell script that maps the command lines accordingly:

```
#!/bin/sh
foocs checkout $2/$1
```

You would then instruct Source-Navigator to work via the shell script:

```
-checkout-individual "foocs-wrapper"
```

Shell scripts can also be useful in situations where a particular operation requires more than one command to be issued to the version control system.

Example

The following example is based on the GNU RCS version control system, which many developers are familiar with. Red Hat has integrated Source-Navigator with RCS and this configuration example is taken directly from the `sn_prop.cfg` configuration file.

```
# GNU Revision Control System (RCS)
sn_add_version_control_system rcs -default yes \
  -checkin "ci" \
  -checkin-comment-via stdin \
  -checkin-exclusive "ci -l" \
  -checkout "co -f" \
  -checkout-exclusive "co -f -l" \
  -checkout-individual "co -f -r" \
  -checkout-individual-to-stdout "co -p" \
  -checkout-with-lock yes \
  -delete-revision "rcs -o" \
  -discard "unco" \
  -history "rlog" \
  -history-pattern { {"^-----" "^====="} } \
  -history-individual "rlog -r" \
  -history-individual-pattern { {"^-----" "^====="} } \
  -history-replacements { {"[ \t]+" " "} } \
  -ignore-dirs RCS \
  -lock "rcs -l" \
  -lock-individual "rcs -l" \
  -revision-number-pattern "^revision (\[0-9.\]+)" \
  -symbolic-tags-pattern { {"^symbolic names" "^keyword"} } \
  -symbolic-tags-replacements { {"\t" ""} } \
  -unlock "rcs -u" \
  -unlock-individual "rcs -u"
```

7

Interapplication Communication

This chapter addresses communication between Source-Navigator and external applications, and provides examples of how to control Source-Navigator from other applications.

These examples use the Source-Navigator Tcl interpreter called *hyper*. To use the standard Tcl/Tk interpreter, replace the `~/snavigator/bin/hyper` string with the appropriate name; for example `/usr/local/bin/wish`.

After an external application connects to a Source-Navigator project, it may be controlled remotely using the Tk `send` command.

The Tk `send` Command

The communication between external applications and Source-Navigator can be achieved using the Tk `send` command. Communicating with Source-Navigator from another application requires that the application know the name of the interpreter running Source-Navigator. The Tk command `winfo interps` is useful for determining the name of all the interpreters active on a host.

Further information on the Tk `send` command can be found in one of the Tcl/Tk texts mentioned on page 136.

Multiple Source-Navigator Interpreters

An application may need to query every Source-Navigator interpreter name in the network. The example below queries every Source-Navigator interpreter name and fetches the name of the Source-Navigator projects. Then if the string `c++_demo` is found, the project name reloads files specified in the command line.

```
#!/bin/sh
# REPLACE $HOME/snavigator/bin/hyper with the
# Source-Navigator installation directory! \
exec $HOME/snavigator/bin/hyper "$0" "$@"
#
# Don’t forget the backslash!
#
# This program instructs all Source-Navigator applications
# running on this host to reload the files specified on the
# command line. The project currently opened by each
# Source-Navigator application must contain the string
# "c++_demo".
# The C++ demo project, c++_demo.proj, is optionally generated
# during the Source-Navigator installation process.
# We don't need the top-level window for this program.
wm withdraw .
# Scan through all the Tcl-based interpreters on this system
# and try and find running Source-Navigators.
foreach interpreter [winfo interps] {
  if {![string match "*navigato*" $interpreter]} {
    continue ;# not S-N, keep looking.
  } else {
    set sn $interpreter ;# found one.
  }
  set msg ""
  # Set the command that we're going to send to the
  # application (using "set" on a temporary variable to
  # examine the name of the currently opened project).
  set cmd {set tmp sn_options(ProjectName)}
  # Send the command to the other application and capture the
  # results.
  catch {set result [send $sn $cmd]} msg
  if {$msg != ""} {
    puts $msg
  }
  # See if the project name contains the substring "c++_demo".
  # If so, then issue the command "paf_parse_uptodate" within
  # the remote interpreter.
  if {[string match "*c++_demo*" $result]} {
    send $sn_parse_uptodate [list $argv]
  }
}

# Force our way out of this event-driven shell.
exit
```

If the script above is executable and the path to the interpreter is correct, the script will reparse and reload all files specified on the command line. If a listed file is being edited in a Source-Navigator editor window, the contents of the editor window will be updated.

NOTE File names have to be used relative to the project root directory if the file is in a subdirectory of the project; otherwise, the file names must be specified with absolute path names (`./` and `../` cannot be used in file names).

Index