

SNiFF+TM

Version 3.2 for Unix and Windows

Reference Guide



TakeFive Software, Inc.
Cupertino, CA
E-mail: info@takefive.com

TakeFive Software GmbH
5020 Salzburg, Austria
E-mail: info@takefive.co.at

Copyright

Copyright © 1992–1999 TakeFive Software Inc.

All rights reserved. TakeFive products contain trade secrets and confidential and proprietary information of TakeFive Software Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure.

Parts of SNIFF+:

Copyright 1991, 1992, 1993, 1994 by Stichting Mathematisch Centrum,
Amsterdam, The Netherlands.

Portions copyright 1991-1997 Compuware Corporation.

Trademarks

SNIFF+ is a trademark of TakeFive Software Inc.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Credits

The first version of Sniff was developed at the Informatics Laboratory of the Union Bank of Switzerland. Its development was considerably facilitated by the public domain application framework ET++.

Authors of the first version:

Walter Bischofberger (Sniff)

Erich Gamma (Sniffgdb)

Erich Gamma and André Weinand (ET++)

Table of Contents

Part I Guidelines

About this Manual	3
Conventions	3
Tool elements	4
Typography	5
Feedback and useful links	5
SNiFF+J for Java	7

Part II Tools Reference

Some Common Elements	11
Fast positioning in lists	11
Regular expression filters	11
Keyboard and mouse shortcuts	11
Status line	12
Frozen check box	12
Layout handle	12
Common Menus	13
Tools menu	13
File menu	15
Edit menu	16
Show menu	18
Target menu	19
Info menu	20
Class menu	22
View menu	23
History menu	23
Help (?) menu	23
Right-click context menus	24
Common Dialogs	25
Open Project dialog	26
Choose Symbol dialog	30
Find/Change dialog	30
Target dialog	32
Check In dialog	34
Check Out dialog	35
Differences dialog	36

Lock dialog	37
Unlock dialog	37
Directory Dialog (Unix)	38
Directory Dialog (Windows)	39
File Dialog (Unix)	40
Project File dialog (Unix)	41
Project File dialog (Windows)	43
Print dialog	44
Licenses dialog	45
Class Browser	47
Quick Reference	48
Basic components	50
Filters	50
Status Line	53
Menus	53
Configuration Manager	55
Quick Reference	56
Basic components	57
Status Line	59
Menus	60
Cross Referencer	65
Quick Reference	67
Basic components	69
Filters	70
Status Line	70
Menus	71
X-Ref Filter dialog	73
Debugger (Unix and Java)	75
Starting the Debugger	75
Multiple simultaneous Debugger sessions	76
Supported debuggers	76
Selecting a debugger back-end	77
Status line	77
Menus	78
Tabs	80
Dialogs	81
Diff/Merge tool	83
Quick Reference	84
Basic components	85

Status Line	86
Menus	86
Documentation Editor	89
Quick Reference	90
Modes of operation	91
Basic components	91
Menus	92
Documentation Synchronizer	93
Quick Reference — Synchronizer	93
Basic Components—Synchronizer	94
Filters—Synchronizer	95
Menus — Synchronizer	96
Hierarchy Browser	101
Quick Reference	102
Basic components	103
Filters.	103
Status Line.	104
Menus	105
Include Browser	107
Quick Reference	108
Basic components	109
Filters.	109
Check boxes	110
Status Line.	110
Menus	110
Launch Pad	113
Quick Reference	114
Basic components	115
Menus	115
Open Project dialogs	119
Log	121
Log window	121
Preferences	123
Preferences	123
Preferences dialog.	124
Appearance view	126
Tools view	130
Source Editor view.	132

Retriever view	137
Cross Referencer view	138
Documentation Editor view	140
Shell view	143
Working Environments view	144
New Project Setup view	146
Version Control System view	149
File Types view	153
Platform view	157
Others view	160
Project Attributes	163
SNIFF+'s Project Attributes	163
General view	164
General Advanced	167
Build Options view	169
Directives	170
Project targets - C/C++	172
Project targets - Java	173
Build Structure - C/C++	174
Build Structure - Java	175
Build Options Advanced	177
Parser view	179
Version Control System view	181
File Types view	182
Group Project Attributes	187
Project Editor	189
Quick Reference	190
Basic components	192
Filters	193
Status line	195
Menus	197
History window	204
Add/Remove Files dialog	206
Statistics dialog	208
Retriever	209
Quick Reference	210
Indexing and caching	211
Basic components	212
Files — Matches List	214
Navigation buttons	214
Modification control buttons	215

Undoing changes	215
Menus	216
The Retriever in “replace only” mode	217
Advanced Retriever Options dialog	217
Find and Replace Filters dialog	218
Locking Status dialog	221
Shell	223
Menus	223
Source Editor	225
Quick Reference	226
Shortcuts	227
Basic components	230
Menus	231
Debugging mode — extra buttons added to the Source Editor	232
Symbol Browser	233
Quick Reference	234
Basic components	234
Filters	235
Status line	236
Menus	237
Working Environments	239
Quick Reference	240
Basic components	241
Modifying Working Environments	243
Working Environments information	244
Menus	246
Modify/New Working Environment dialog	249
Users dialog	250
Part III Advanced Reference	
SNiFF+ Executables	253
sniff	253
Environment variables	254
Multiple simultaneous SNiFF+ sessions	255
SNiFF+ without display (batch mode)	256
sniff_arch	256
sniff_genproj	257

Sniffaccess	259
Invoking Sniffaccess	259
SNIFF+ external access communication protocol	261
Sniffaccess requests	264
Sniffaccess notifications	271
HP Softbench BMS bridge (Unix only)	274
Advanced Customization	279
Customizing the SNIFF+ <Meta> key (Unix only)	280
Template files	282
Parser config file	282
Filter file	282
Custom menus	283
Error formats	288
Setting SNIFF+'s look and feel (Windows NT/95 only)	290
Working with IDL Projects in SNIFF+	293
What the SNIFF+ IDL Parser does	293
Integration of the SNIFF+ IDL Parser with SNIFF+	293
Using the SNIFF+ IDL Parser without SNIFF+	294
Using SNIFF+'s Make Support for compiling IDL files	295
IDL Projects	296
Editing \$SNIFF_DIR/make_support/<platform>.mk	297
Creating a server project and configuring Make support for it	297
Creating a client project and configuring Make support for it	301
Default Makefile	306
Regular Expressions in SNIFF+	307
Quick Reference - Syntax	308
Literals and metacharacters	309
Character classes or lists	314
Groups, alternatives and back references	317
SNIFF+ - Generated Files	319
Generated files	320
Part IV Glossary and Index	
Glossary	325
Index	331

Part I

Guidelines



About this Manual

What this manual is

This manual is part of the SNIFF+ documentation set, which consists of:

- User's Guide
- Reference Guide
- C++ Tutorial
- C Tutorial
- Java Tutorial
- Fortran Tutorial
- Quick Reference Guide
- Release Notes, Installation Guide and Application Papers
- Online documentation of the above in HTML, PostScript and PDF formats

Conventions

One basic term

- **Symbol** — any programming language construct such as a class, method, etc.

Two conventions: menu references

For clarity and to avoid undue verbosity, the phrase:

“Choose the MenuCommand from the MenuName” is presented as follows:

- Choose **MenuName > MenuCommand**.

A context menu that appears when you click the right mouse button is referred to as:

Context menu, and consequently:

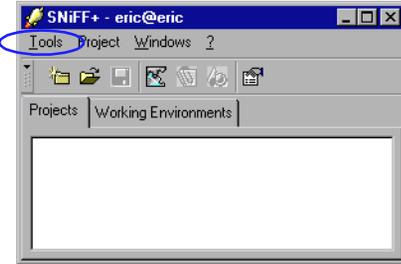
“Choose a menu command from the context menu that appears when you click the right mouse button” is presented as follows:

- Choose **Context menu > MenuCommand**

A note on Unix/Windows

The screenshots in this manual are all done on Windows NT. If you are working on Unix, what you see on your screen may look slightly different.

When you start SNIFF+, the first tool that appears is the Launch Pad. In this and other SNIFF+ tools, the first item in the menu bar is for launching tools.



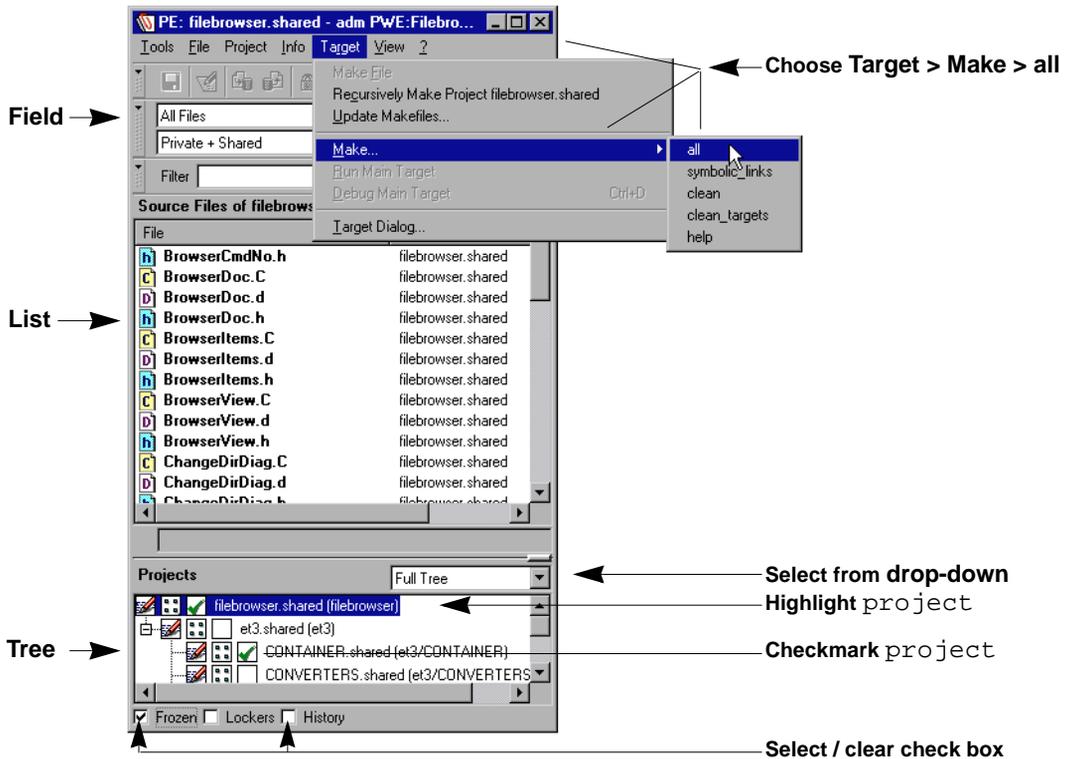
- On **Windows**, it is called **Tools**.
- On **Unix**, it is depicted by an **Icon**.

When we refer to this menu in order to launch a tool from the Launch Pad, or any other open SNIFF+ tool, we will use the notation:

Choose **Tools > ToolName**.

- On Unix a “check box” looks like a “button” (Motif Look), and a “drop-down” looks like a “pop-up”.

Tool elements



Typography

Capitalized Words	Names of tools, windows, dialogs and menus start with capital letters. Examples: Symbol Browser, Tools menu, File dialog.
<i>Italics</i>	Names of manuals and newly introduced terms are in italics. Examples: <i>User's Guide</i> , the <i>workspace</i> concept.
Boldface and Bold italics	Menu, field and button names and menu entries are printed in bold-face. Placeholders for symbols, selections or other strings in menus are in bold italics. Example: From the menu, choose Show > Symbol(s) <i>selection...</i>
Monospace	Code examples and symbol, file and directory names, as well as user entries are printed in monospace type. Examples: <code>.login</code> , <code>\$PATH</code> , <code>class VObject</code> . Type <code>abc</code> .
<Keys>	Special keys are printed in monospace type with enclosing '< >'. Examples: <code><CTRL></code> , <code><Return></code> , <code><Meta></code> .

Feedback and useful links

Your feedback is always very welcome. Please send feedback to one of our support e-mail addresses.

Europe:

sniff-support@takefive.co.at

USA:

sniff-support@takefive.com

Useful links

SNiFF+ web pages:

- SNiFF+ Users Mailing List
<http://www.takefive.com/support/sniff-list.html>
- SNiFF+ Users Mailing List Archive
<http://www.takefive.com/sniff-list>
- Frequently Asked Questions
<http://www.takefive.com/support/faq.html>
- Customer Newsletter
http://www.takefive.com/news/customer_newsletter.html

This manual relates to SNiFF+ in general.

- For Java-specific issues, please refer to the *SNiFF+ Java Tutorial*. To open the *Java Tutorial* online, choose **Help(?) > Tutorials > Java** from the Launch Pad's menu.
- Please also refer to the *Java Tutorial* to find out how to get started with the **Visaj GUI Builder** integration. SNiFF+ integration features are incorporated in the *Visaj User's Guide* under the Visaj Class Editor's **Help** menu.

Part II

Tools Reference



Some Common Elements

Introduction

This chapter introduces the following user interface elements that are common to many tools in SNIFF+:

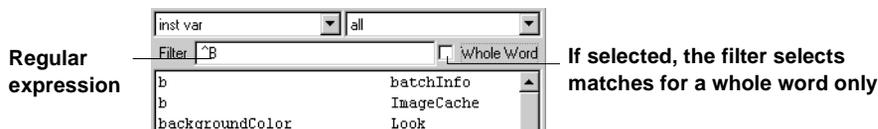
- [Fast positioning in lists — page 11](#)
- [Regular expression filters — page 11](#)
- [Keyboard and mouse shortcuts — page 11](#)
- [Status line — page 12](#)
- [Frozen check box — page 12](#)
- [Layout handle — page 12](#)

Fast positioning in lists

Click into a list and press a sequence of keys to position the list to the first entry that matches the sequence. By pressing the <ESC> key, you can restart the sequence.

Regular expression filters

Most tools offer a name filter for filtering the items according to a regular expression.



To filter the list, enter a regular expression in the **Filter** field. After you press <Return>, the list is filtered according to the regular expression. To remove the filter, delete the regular expression from the **Filter** field and press <Return> again. If you press the **Whole Word** button, only whole words are matched to the expression.

For more information about regular expressions, see also [Regular Expressions in SNIFF+ — page 307](#).

Keyboard and mouse shortcuts

Keyboard and mouse shortcuts are available throughout SNIFF+. Each tool chapter in this manual has a **Quick Reference** section, accessible also via the **Help(?)** menu of each tool, where available shortcuts are described.

Please refer to [Customizing the SNIFF+ <Meta> key \(Unix only\)](#) to find out how to redefine the <Meta> key.

Status line

Every SNIFF+ tool has a status line at the bottom. Besides showing important status information, it contains a **Frozen** check box (see [Frozen check box — page 12](#)). The tool-specific status information is described in the corresponding tool chapters.

Frozen check box

The **Frozen** check box controls whether the tool can be reused for a new request. If the **Frozen** check box is selected and a request is sent to the tool, this tool will stay untouched and another tool of the same type is instantiated. If the **Frozen** check box is not selected, tools are reused.



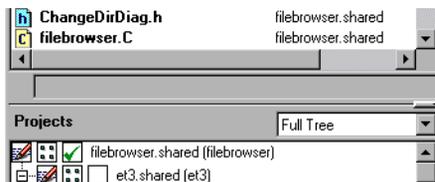
↑ Toggles whether tool is reusable

Layout handle

All SNIFF+ tools consisting of more than one view have a layout handle. The layout handle allows modification of the size ratio between two views.

Dragging the handle with the mouse changes the ratio. A view can be resized to zero, leaving all the space for the other view.

- A **double-click** on the handle shows/hides the less important view completely.



Click with the mouse and drag;
double-click to show/hide the view completely

Common Menus

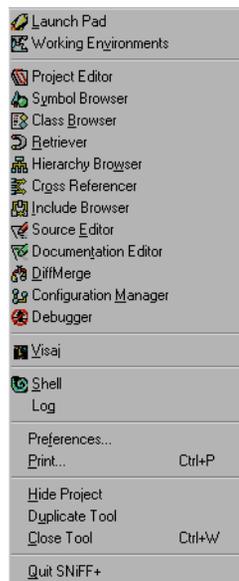
Introduction

The menu commands described in this section apply to the tools listed under each menu heading. Due to the specific functionality of a given tool, individual commands within a menu may differ. Where this is the case, tool-specific commands or variations are described in the relevant tool chapter.

Tools menu

The **Tools** menu is the first menu in the menu bar of every tool.

- **On Unix**, the **Tools** menu is represented by the **Icon** of the open tool. The icon used to represent each tool can be seen in the menu itself (below).



The tool that you choose from this list becomes the active tool. A new instance of the tool is opened if no reusable one is available

Tools menu command	Description
Tool name	Activates the named tool. If the tool is not already open or frozen (i.e. not reusable), a new instance of the tool is opened.

Tools menu command	Description
Debugger	If the Debugger is already open, it can be easily located by using this menu entry.
Visaj	Opens a dialog in which you can open an existing Visaj project in the Visaj Class Editor or in which you can create a new Visaj project.
Print...	Opens a Print dialog for printing the tool's main view. Before opening the Print dialog, some tools with multiple views (like the Configuration Manager) open a dialog in which you can select the view that you want to print. The Print... command is disabled in some tools.
Hide Project	Hides the selected project and all tools associated with it. When the project is hidden, the project name is displayed in italics in the Launch Pad. Show the Project again by double-clicking on its name in the Launch Pad.
Duplicate Tool	Opens a duplicate of the tool in which you executed this command. The duplicate then becomes the “active” tool—when you issue a command that makes use of this tool, the duplicate (the “active” tool) will be used. Note that the starting state of the duplicate tool is the same as the state of the original tool. For example, the History menu (if the tool has one) is the same in both the original and the duplicate tool. The Duplicate Tool command is disabled in the Tools menu of the Include Browser, Cross Referencer and Diff/Merge tools.
Close Tool	Closes the active tool.
Quit SNIFF+	Terminates the current SNIFF+ session.

When a project is open, all commands are available. If no project is open, only the following commands are available:

- Launch Pad
- Working Environments
- Log...
- Preferences...
- Quit SNIFF+

File menu

The commands described below apply to the **File** menu in the

- Source Editor
- Diff/Merge tool
- Documentation Editor

For a description of the **File** menu in other tools please refer to the appropriate chapters in this manual.

File menu command	Description
Load...	Opens a File dialog, in which you can enter the name of the file to be loaded.
Save	Saves the modified file. This command is enabled for modified files. The file is saved, and SNIFF+'s Symbol table is updated. Furthermore, all tools are automatically updated to reflect the changes made to the file. In the Preferences, you can set whether backups are made. If backups are made, these are named <i>filename%</i> .
Save As...	Opens a File dialog, where you can enter a new name for the file.
Revert	Returns the file to its last state before any modifications were made to it. The command is enabled for modified files.
Check Out...	Opens the Check Out dialog, where you can select the version of the file that you want to check out.
Check In...	Opens a Check In dialog for checking a file in to the repository.
Lock...	Opens a Lock dialog for locking the current file.
Unlock...	Opens an Unlock dialog for unlocking the current file.
Show History Info	Opens a Project Editor and a History dialog to display the version history information of the loaded file. If a Project Editor is already open (and its Frozen check box is not selected), it is used.
Show Differences...	Opens a Differences dialog, in which you can enter two versions of a file that you want to compare with each other.

Edit menu

The commands described below apply to the **Edit** menu in the

- Source Editor
- Diff/Merge tool
- Documentation Editor
- Retriever

For a description of the **Edit** menu in other tools please refer to the appropriate chapters in this manual.

Edit menu command	Description
Undo last command	Undoes the <i>change you made to the text</i> . You can specify undo levels in the Preferences.
Redo last command	Redoes the last undone change.
Cut	Cuts the current selection to the paste buffer.
Copy	Copies the current selection to the paste buffer.
Paste	Pastes the paste buffer contents to the current cursor position or selection. This command is enabled when the paste buffer is not empty.
Paste File	Opens a File dialog where you can select a file to be copied into the current file. The selected file is pasted to the cursor position.
Find/Change...	Opens a Find/Change dialog. You can use regular expressions in this dialog.
Find Again	Repeats the latest search triggered from Find/Change dialog.
Go To Line...	Opens the <i>Go To</i> dialog, where you can enter the line number you want to jump to.
Search Forward	Searches for the next sequence of characters that matches the current selection. If a match is found, the editor positions to the match and selects it.
Search Backward	Searches for the previous sequence of characters that matches the current selection. If a match is found, the editor positions to the match and selects it.
Select All	Selects the entire file.

Edit menu command	Description
Nest	Shifts the currently selected line(s) one tab width to the right.
Unnest	Shifts the currently selected line(s) one tab width to the left.
Comment	Inserts '/' comment characters at the beginning of the currently selected line(s).
Uncomment	Removes '/' comment characters from the beginning of the currently selected line(s).

Show menu

The commands that are described below apply to the **Show** menu in the

- Source Editor
- Retriever
- Documentation Editor

Show menu command	Description
Symbol(s) <i>symbol</i>...	Shows the declaration or implementation of <i>symbol</i> . If <i>symbol</i> is ambiguous, a dialog opens with a list of valid alternatives.
Baseclass <i>class</i>	Shows the declaration of the base class of the currently selected class. This entry is enabled when the cursor is positioned in the scope of a class that has a base class. If <i>class</i> has multiple base classes, you can choose one of them from a submenu.
Overridden Method <i>method</i>	Shows the overridden method of the closest base class that defines <i>method</i> into a Source Editor.
Declaration/Implementation of <i>method</i>	Shows the declaration/implementation of <i>method</i> .
Header/Implementation File	Shows the corresponding header/implementation file.
Next Match	If an Editor is requested from, e.g., the Cross Referencer, this command shows the next reference to the symbol. (Does not apply in the Retriever.)

Target menu

The commands that are described below apply to the **Target** menu in the

- Project Editor
- Source Editor
- Shell

You can use the **Target** menu to issue *make* commands or run and debug executables. The commands and targets that are executed and made in this menu are specified in the Project Attributes. See also [Build Options view — page 169](#).

Target menu command	Description
Make File <i>objectfile</i>	Spawns a Shell and issues <code>make objectfile</code> in the source directory of the project. <code>objectfile</code> is the object file of the selected source file.
Recursively Make target	Spawns a Shell and recursively starts <code>make</code> for all the subprojects for which <code>make</code> support files are generated. Finally, <code>make</code> for the default target of the root project is called. SNIFF+ generates Make support files for projects whose Generate Make Support Files attribute is enabled. You can enable this attribute in the Build Options view of the Project Attributes dialog. To learn about a project's default target, please refer to User's Guide — Specifying the targets of a project — page 95 .
Update Makefiles	Updates the <code>make</code> support files of all the projects in the Project Tree. There are three types of <code>make</code> support files: dependency file, macros file and include file. See also Build Options view — page 169 .
Make target	Spawns a Shell tool and issues <code>make target</code> in the source directory of the project. <code>target</code> is the default target of the project.
Make Submenu	The Make... submenu contains the targets entered in the Project Attributes dialog. A separator line separates helping targets from project targets. You can build the individual targets of the project by choosing the appropriate entry. The Make all command works in conjunction with SNIFF+'s Make Support feature. See also User's Guide — Building targets recursively — page 98 .

Target menu command	Description
Run target	Spawns a Shell tool and executes the default target of the project. When you execute this command, a Program Arguments dialog appears. You can enter arguments for the run command in this dialog.
Debug target	Starts the debugger and loads the <i>target</i> executable. This entry is only enabled when you have entered the target name in the Make view of the Project Attributes dialog and the target is executable. See also Build Options view — page 169 .
Target Dialog...	Opens the Target dialog, in which you can choose a target from the list of all the targets in the Project Tree.

Info menu

The commands that are described below apply to all of the SNIFF+ tools, except for the Include Browser, Working Environments tool and the Configuration Manager.

In SNIFF+'s browsing tools, symbols are unambiguously specified as items in a list. Ambiguities may arise in the Source Editor, since the cursor is positioned in plain text. Nevertheless, SNIFF+ is able to minimize ambiguity by means of the symbolic information in the Symbol table. SNIFF+ does this by employing the notion of the “scope” of the cursor position. You can unambiguously select a symbol by clicking directly on the symbol's signature, or in a reference (function body or comment) to this symbol. If the symbol is not unique in the given context, a dialog appears when you choose a command from the **Info** menu. This dialog contains a list of possible candidates that you can then choose from.

* The **Info** menu in the Project Editor only contains six of the commands described below. The commands that apply to the Project Editor are preceded by an asterisk (*).

Info menu command	Description
Show Declaration of selection	Opens the Source Editor and positions at the declaration of <i>selection</i> .
Retrieve selection From This File	This command is only available in the Source Editor. Opens a Retriever and retrieves all occurrences of <i>selection</i> from the current file.

Info menu command	Description
Retrieve selection From This Project	This command is only available in the Source Editor. Opens a Retriever and retrieves all occurrences of <i>selection</i> from the project to which the current file belongs.
* Retrieve selection From All Projects	Opens a Retriever and retrieves all occurrences of <i>selection</i> from all projects in the Project Tree.
* Retrieve selection (using Retriever settings)	Opens a Retriever and retrieves all occurrences of <i>selection</i> from the projects that are checkmarked in the Retrievers Project Tree. Also all other settings in the Retriever, except filters, are applied.
* Find Symbols Matching selection	Opens a Symbol Browser to search for symbols that match <i>selection as a whole word</i> .
* Find Symbols Containing selection	Opens a Symbol Browser to search for symbols that contain <i>selection as a substring</i> .
Symbol Refers-To	Opens a Cross Referencer and starts a Refers-To query on <i>symbol</i> . The settings of this Cross Referencer's Xref Filter are used for the query parameters.
Symbol Referred-By	Opens a Cross Referencer and starts a refers-by query on <i>symbol</i> . The settings of this Cross Referencer's Xref Filter are used for the query parameters.
Symbol Refers-To Components	Opens a Cross Referencer and starts a query for showing all symbols (classes and structures) that are components of <i>symbol</i> . If the current selection is a member of a class/structure, the class/structure is taken for this query.
Symbol Referred-By As Component	Opens a Cross Referencer and starts a query for showing all symbols that have <i>symbol</i> as a component. Note that you can also query primitive C data types with this command.

Info menu command	Description
* selection Includes	Opens an Include Browser and shows the files that <i>selection</i> includes.
* selection Is Included-By	Opens an Include Browser and shows the files that include <i>selection</i> .
Show Documentation of symbol	Opens a Documentation Editor and positions it to the documentation of <i>symbol</i> . An alert message appears if no documentation file exists for either the entire file or for <i>symbol</i> . You then have the option of creating a documentation file.
Documentation Synchronizer...	Opens the Documentation Synchronization dialog.

Class menu

The commands that are described below apply to all of the SNIFF+ tools, except for the Project Editor, Include Browser and the Configuration Manager.

You can use the **Class** menu to issue commands for obtaining class-specific information about the current selection. The entries are only enabled if the selection is a class. The following commands are available:

Class menu command	Description
Browse <i>class</i>	Shows the members of <i>class</i> in the Class Browser
Show class in Entire Hierarchy Show class Relatives	Opens a Hierarchy Browser and loads either the entire class graph or the graph of the base and derived classes. <i>class</i> is highlighted in the Hierarchy Browser.
Mark Definers of class in Entire Hierarchy Mark Relatives Defining class	Opens a Hierarchy Browser and loads either the entire class graph or the graph that consists of <i>class</i> and its base and derived classes. All classes that define <i>class</i> are displayed in boldface in the Hierarchy Browser.

View menu

The View menu of all tools that have a Project Tree offers the following commands:

View menu command	Description
Select Project Set >	Opens a submenu with a list of all saved project sets.
Save Project Set	Opens a dialog where you can enter a name for saving the current view of your Project Tree, that is, which projects are checkmarked and which nodes are expanded/collapsed. Note that, if the name exists already, it is simply overwritten.
Remove Project Set	Removes the selected project set.

History menu

You can use the **History** menu, available in most tools, to return the tool to a previous state (or to re-issue an earlier query). The type of entries depends on the tool.

You can specify the number of entries in the **History** menu in the Preferences.

Help (?) menu

Online help is displayed in your preferred HTML browser, which you can set in the Preferences.

Help(?) menu command	Description
Tool help	Opens the Online Reference at the description of the current tool
Context help	Changes the cursor to context help mode. Clicking into a specific context then opens the Online Reference at the description of the selected context.
Quick Ref	This command is available in all tools where icons and typeface provide graphical feedback. The Online Reference is opened at a summary of these elements as they apply to the current tool.

Right-click context menus

Right-click context menus are available throughout SNIFF+. The **Context menu** offers a selection of frequently used commands. Generally, these correspond to menu commands. An exception are the commands in the Project Tree's **Context menu**.

The Project Tree Context menu

The following commands are available in the Project Tree Context menu of all tools that have a Project Tree.

Context menu command	Description
Select from <i>Project</i> Only	Only the highlighted <i>Project</i> is checkmarked. Only the elements of the checkmarked project are shown in the main view of the tool.
Select From All Projects	All Projects are checkmarked. The elements of projects are shown in the main view of the tool.
Propagate Selection	The selection of checkmarked projects is propagated to all other open tools that have a Project Tree.
Collapse/Expand Subprojects of <i>Project</i>	Collapses/Expands the node of <i>Project</i> .
Collapse to Level...	Starts a dialog where you can enter the number of levels you want to expand the node to.
Expand all Projects	Fully expands all nodes. Fully expanded nodes are indicated by a “—” sign.
Propagate Expansion	Propagates the tree arrangement to all other open tools that have a Project Tree.
Total <i>number</i> / Selected <i>number</i>	Shows the total number of projects in the Project Tree and the number of those that are currently checkmarked.

Common Dialogs

Introduction

The dialogs described in this chapter can be opened from a number of different contexts within SNIFF+.

Dialogs with unique contexts, that is, those that are opened from only one tool, are described in the corresponding tool description chapter of this manual.

- [Open Project dialog — page 26](#)
- [Choose Symbol dialog — page 30](#)
- [Find/Change dialog — page 30](#)
- [Target dialog — page 32](#)
- [Check In dialog — page 34](#)
- [Check Out dialog — page 35](#)
- [Differences dialog — page 36](#)
- [Lock dialog — page 37](#)
- [Unlock dialog — page 37](#)
- [Directory Dialog \(Unix\) — page 38](#)
- [Directory Dialog \(Windows\) — page 39](#)
- [File Dialog \(Unix\) — page 40](#)
- [Project File dialog \(Unix\) — page 41](#)
- [Project File dialog \(Windows\) — page 43](#)
- [Print dialog — page 44](#)
- [Licenses dialog — page 45](#)

Open Project dialog

You can open projects in the current Working Environment with the Open Project dialog. The dialog appears when, in the Working Environments tool, you

- double-click on a Working Environment or
- choose the **File > Open Project** menu command.

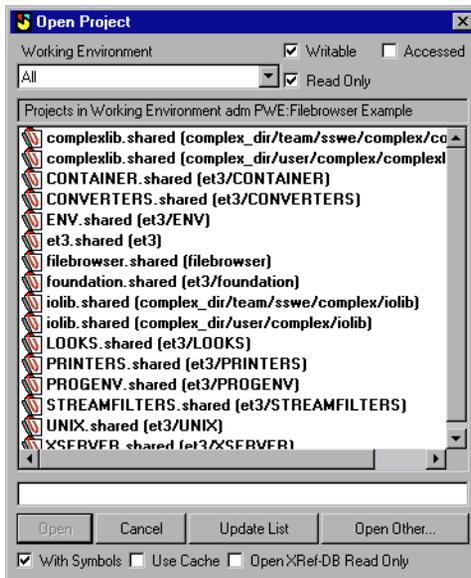
The dialog also appears when, in the Launch Pad's Working Environment tab, you

- double-click on a Working Environment

Projects in the Project List are listed alphabetically. The project name is followed (in parentheses) by the path relative to the root directory.

Note

When you open this dialog for the first time, it is initially empty. To populate the dialog with projects, press the **Update** button. Also changes (new or removed projects) will only be reflected after you press **Update**.



Typeface in the Project List

Typeface	Project is
Bold	writable in the current working environment
Non-bold	read-only in the current working environment
<i>Italics</i>	located in a shared working environment
Non-italics	located in the current Private Working Environment
Grayed	not at specified location

Mouse clicks

- **Double-click** on an item in the Project List to open the project in the working environment selected in the Working Environments tool.

Working Environment drop-down

Use the **Working Environment** drop-down to constrain the Project List.

List entry	Description
All	Shows all projects in the hierarchy tree of the current working environment. The displayed Project List is also subject to the selections in Filter check boxes.
<i>Other entries</i>	Constrains the list to individual working environments in the hierarchy tree of the current working environment. The displayed Project List is also subject to the selections in Filter check boxes.

Filter check boxes

Selected check boxes filter the entries in the Project List

- | | |
|------------------|--|
| Writable | Shows all projects that are writable in the current working environment. The displayed Project List is also subject to the selection in Working Environment drop-down and the other Filter check boxes. Default: selected. |
| Read only | Shows all projects that are read-only in the current working environment. The displayed Project List is also subject to the selection in Working Environment drop-down and the other Filter check boxes. Default: selected. |
| Accessed | Shows also the projects in accessed working environments. Note that the Accessed filter button is only effective when you choose All in the Working Environment drop-down. Default: not selected. |

Opening Mode check boxes

Selected Opening Mode check boxes determine what data is used in opening the project.

- | | |
|---------------------|---|
| With Symbols | Opens projects with symbol information. For browsing and day-to-day development work, enable this check-box. If you do not need symbol information, e.g. to open large projects for making structural changes, clear this check box. Also, if you are using the database-driven cross reference system, and X-Ref databases are locked, projects can still be opened without symbol information by deselecting this check box. Default: selected. |
|---------------------|---|

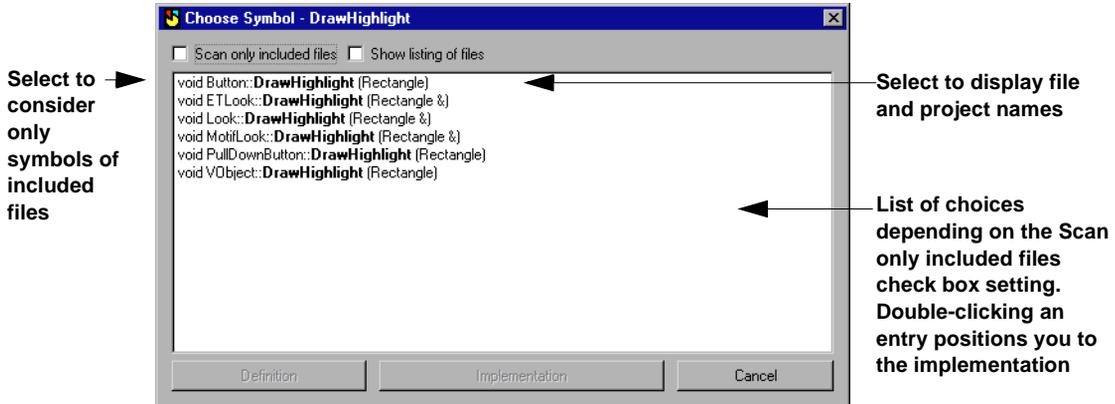
- Use Cache** When projects are closed, all necessary information for re-opening them is cached in a single file to speed up project opening. If **Use Cache** is selected, only this file is read. If not, all and the original Project Description Files (PDFs) are read, and all source files are checked.
Caution: The cached information can be **incorrect** if (1.) changes are made to projects and files between SNIFF+ sessions, that is, outside of SNIFF+, or (2.) if the preceding SNIFF+ session was terminated unexpectedly. Default: not selected.
- Open X-Ref-DB** Applies only if the database-driven cross reference system is used.
- Read Only** The option is only available for the first Project to be opened in a given Working Environment.
As soon as the first Project has been opened with **Read Only** database access, all subsequent Projects can also only be opened with **Read Only** database access in the current Working Environment and current SNIFF+ session. Default: not selected.

Buttons

- Open** Opens the project selected in the Project List.
- Update** Recursively checks for new projects, starting from the root directory of the current working environment and updates the Project List with the latest information. If the current working environment is a Private Working Environment, a dialog asks whether the accessed shared working environments should also be checked.
- Open Other** Opens the Project File dialog, in which you can open a project other than those listed in the Project List.

Choose Symbol dialog

The dialog appears when you choose **Show > Symbol(s) symbol...** or otherwise request a symbol, and multiple matches for *symbol* are found.



Check boxes

Scan only included files	Displays only those symbols that are defined in included files of the current file.
Show listing of files	Displays the file name and project name where the symbol is located.

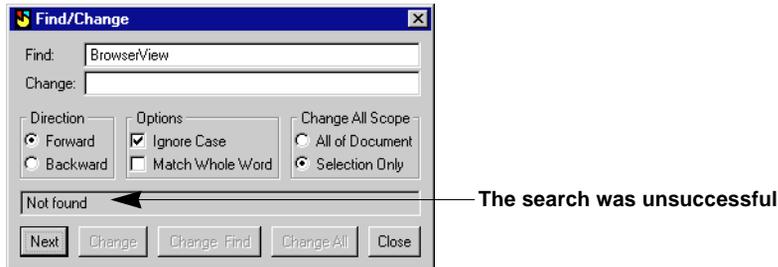
Buttons

Definition	Jumps to the definition of the selected symbol.
Implementation	Jumps to the implementation of the selected symbol. A double-click on a symbol has the same effect as pressing the Implementation button.

Find/Change dialog

To open the Find/Change dialog, choose the **Edit > Find/Change...** menu command. The Find/Change dialog allows finding and changing strings and supports regular expressions. An overview of regular expressions is provided under [Regular Expressions in SNIFF+ — page 307](#).

You cannot modify read-only files in the Find/Change dialog.



Edit fields

- | | |
|---------------|---|
| Find | Describes the text that is to be found. It may contain regular expressions. When the Find/Change dialog is started from a text with highlighted text, this selection is automatically entered into the Find field. |
| Change | Text that replaces a match when the Change and Find or Change All is pressed. |

Direction group

- | | |
|------------------------------|---|
| Forward/
Backward | Search direction. The search begins at the current cursor position. |
|------------------------------|---|

Options group

- | | |
|-----------------------------|--|
| Ignore Case | Specifies whether the search should be case-sensitive or not. |
| Match Whole
Word | Specifies whether the search string should match a whole word. |

Change All Scope group

- | | |
|--|--|
| All of
Document/
Selection Only | Specifies whether the scope of the search is the whole document or just the currently active selection. Option only applies to Change All . |
|--|--|

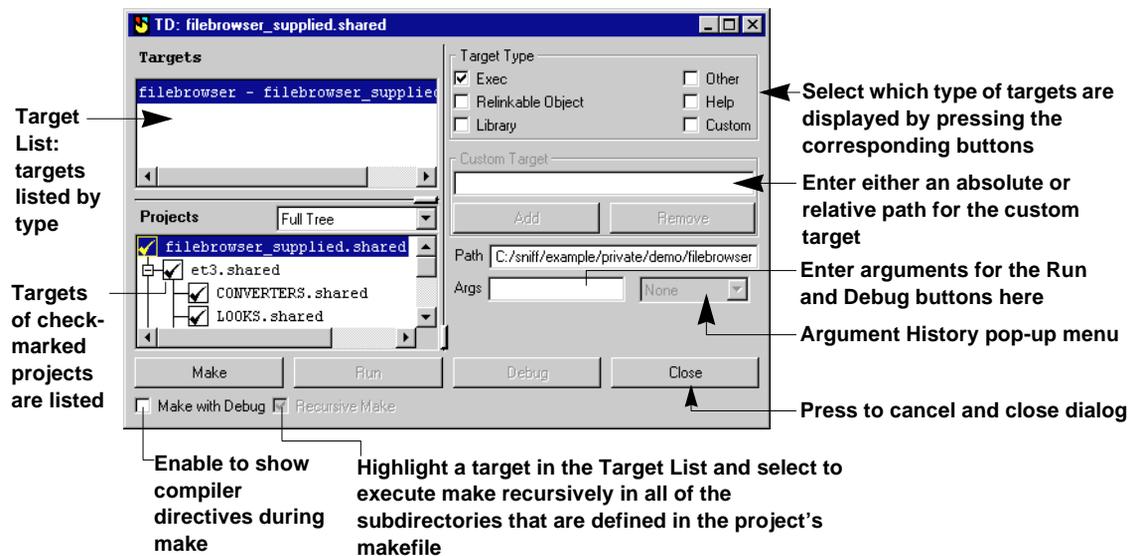
Buttons

Find Next	Triggers the search for the next match.
Change and Find	Replaces the current selection with the change string, then starts a new search.
Change All	Changes all occurrences of the find string in the current change scope to the text entered in the Change field.
Close	Closes the Find/Change dialog.

Target dialog

In the Target dialog, you can perform operations on custom or temporary targets, as well as on the targets that are specified in the project attributes.

To open the Target dialog, select **Target Dialog...** in the **Target** menu of the Source Editor, Project Editor or Shell.



Buttons

Make	Starts a Shell and makes the target that you selected from the Target List.
Run	Runs the target that you selected from the Target List. You can enter arguments for the Run command in the Args field or you can use the Argument History pop-up menu to select a previously used argument.
Debug	Starts the Debugger. You can enter arguments for the Debug command in the Args field or you can use the Argument History pop-up menu to select a previously used argument. Note: When you select a target from the Target List, the Run and Debug buttons are enabled only if the selected target exists in your file system and is executable.
Add	Adds the name of the target in the Custom Target field to the Target List. Note that the custom target is added to the root project of the Project Tree.
Remove	Removes the currently selected target from the Target List.
Close	Cancels the operation and closes the dialog.

Adding arguments to the Argument History pop-up menu

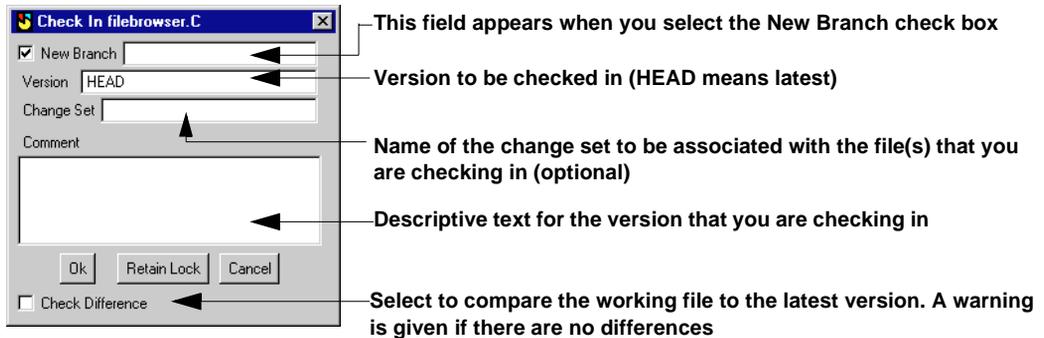
You can quickly add an argument to the **Argument History** drop-down by entering the argument in the **Args** field and then pressing either <Return> or <Tab>.

Recursive make switch

When you highlight a target in the Target List and select this check box, Make is recursively executed in all of the subdirectories that are defined in the `SUBDIRS` macro of the makefile of this target. By default the `SUBDIRS` macro is set to the `SNIFF_SUBDIRS` macro (defined in the `macros.incl` make support file).

Check In dialog

The Check In dialog appears when you choose **File > Check In...** menu command.



Check boxes

New Branch

When you select this check box, a field where you can enter the name of a new branch appears. SNIFF+ will automatically give the new branch's name the prefix HEAD_. The version number (or associated configuration name) you enter will be the point at which the new branch starts, and the file that you are checking in will be the first version of the new branch.

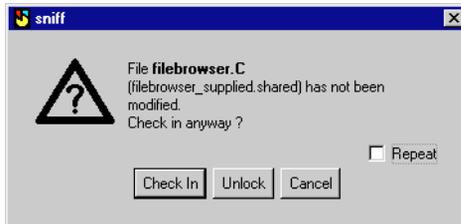
Note

For the **SCCS** version control system, you have to create branches in the Check Out dialog. As a result, the Check In dialog for SCCS does not contain the **New Branch** check box.

Some version control tools do not allow you to check in unmodified files. Please refer to your tool's documentation for details.

Check Difference

Press to check whether the contents of the file that you are checking in are different from the current HEAD version. If the file has not been modified, you will be informed by the following dialog.



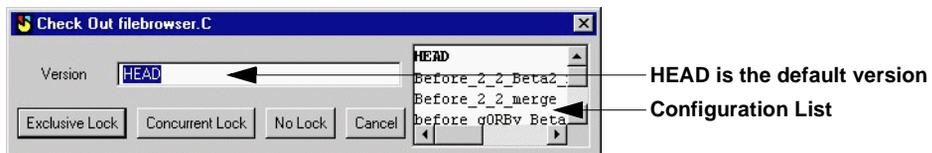
Check In button Checks in the unmodified file(s) with a new version number.

Unlock button Unlocks the unmodified file(s) to make them available to other team members without assigning a new version number.

Repeat check box If Repeat is selected, the command button action is applied to all affected files.

Check Out dialog

The Check Out dialog appears when you choose **File > Check Out...** menu command.



To check out a file, enter its version number. Alternatively, you can also click the associated configuration name of a version from the Configuration List on the right. The list also contains branch configuration names. Note that configuration names that begin with HEAD_ refer to branches.

SNiFF+ uses your Default Configuration to enter the default version of the file in the **Version** field. See also [Default Configuration — page 150](#)

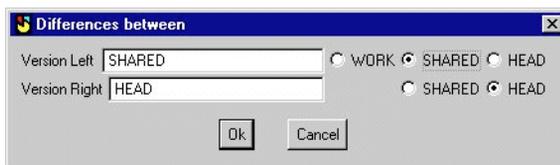
- Exclusive Lock** Checks out the specified version of the file to the Private Working Environment and puts an exclusive lock on the file in the version tool. An exclusive lock means that nobody else can lock the same version. The working file status is set to writable.
- Concurrent Lock** Checks out the specified version of the file to the Private Working Environment and locks the file in the version tool in such a way that others can also lock the same branch of the file. The working file status is set to writable. For systems like RCS that do not support concurrent locking, no lock is set at all. Only the first developer who checks in a concurrently locked file can directly check in the file — all others have to merge their versions back instead of checking them in.
- No Lock** Checks out the specified version of the file to the Private Working Environment. The status of the working file is set to read-only. No lock is set on the file.

Note

For the **SCCS** version control system, the Check Out dialog contains the **New Branch** check box for creating a branch from the version of the file that you are checking out. However, due to limitations in **SCCS**, you cannot attach a symbolic name to head versions of branches.

Differences dialog

The Differences dialog appears when you choose the **File > Show Differences...** menu command.



Radio buttons

WORK	Refers to a working file. If the working file exists in a Private Working Environment, WORK refers to it. Otherwise, WORK refers to the shared file in the shared working environment.
SHARED	Refers to the version of the shared file in the shared working environment.
Default Configuration (here HEAD)	Refers to the default configuration of the file. This version will not be the same as the SHARED version if a member of your team has checked out the file, modified it and then checked it in again.

Note

If you select the **SHARED** radio button for the **Version Right** field, you must select **WORK** for the **Version Left** field.

Lock dialog

The Lock dialog appears when you choose the **File > Lock...** menu command.



To lock a file, enter its version number. Alternatively, you can also click the associated configuration name of a version from the Configuration List on the right. The list also contains branch configuration names. Note that configuration names that begin with **HEAD_** refer to branches.

SNiFF+ uses your Default Configuration to enter the default version of the file in the **Version** field. See also [User's Guide — Default Configuration — page 150](#).

Unlock dialog

The Unlock dialog appears when you choose the **File > Unlock...** menu command.

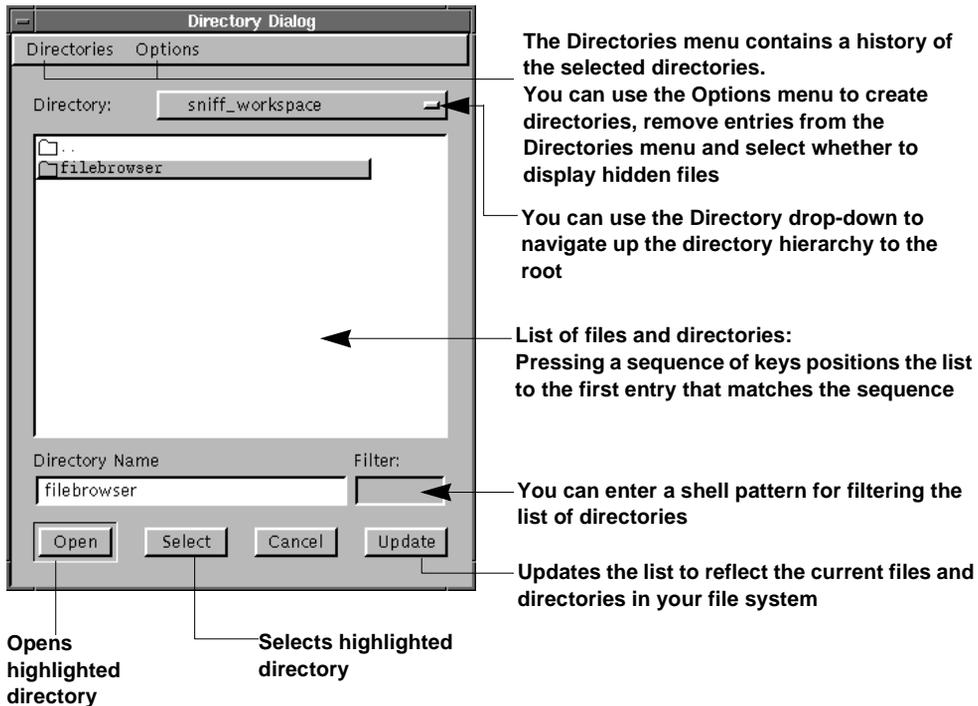


To unlock the file, enter its version number. Alternatively, you can enter the associated configuration name of a version.

SNiFF+ uses your Default Configuration to enter the default version of the file in the **Version** field. See also [User's Guide — Default Configuration — page 150](#).

Directory Dialog (Unix)

The Directory dialog allows the selection of a directory.



The **Directory Name** field expands shell metacharacters like '~' and \$variables. You can enter a directory by either pressing <Return> or the **Open** button.

Entering a C-shell regular expression in the **Filter** field and pressing the **Update** button filters the list to only those entries that match the filter.

Directories menu

The **Directories** menu lists the most recent active directories. Choosing an entry from the list opens that directory and displays its contents in the Directory List.

To remove entries, choose **Options > Edit Directories**.

Options menu

The **Options** menu lets you configure entries for the **Directories** menu, create directories, and select whether hidden files are displayed.

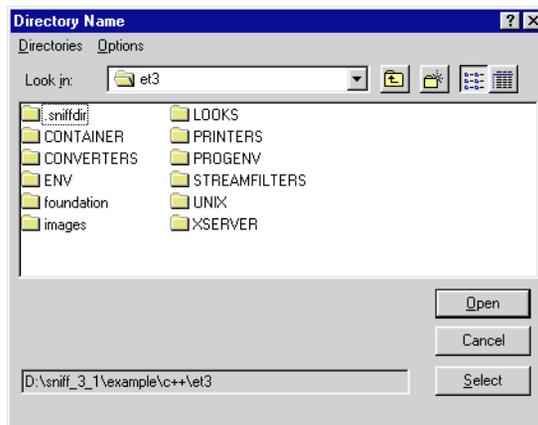
Directory drop-down

The **Directory** drop-down shows the list of directories in the path from the current directory to the root. You can click on any directory in the path to enter it.

Buttons

Open	Opens the highlighted directory and displays its contents in the Directory List.
Select	Selects the opened directory and closes the Directory dialog.
Update	Updates the File List (useful when new files are externally created or deleted while the Directory dialog is open or when a regular expression is entered into the Filter field).

Directory Dialog (Windows)

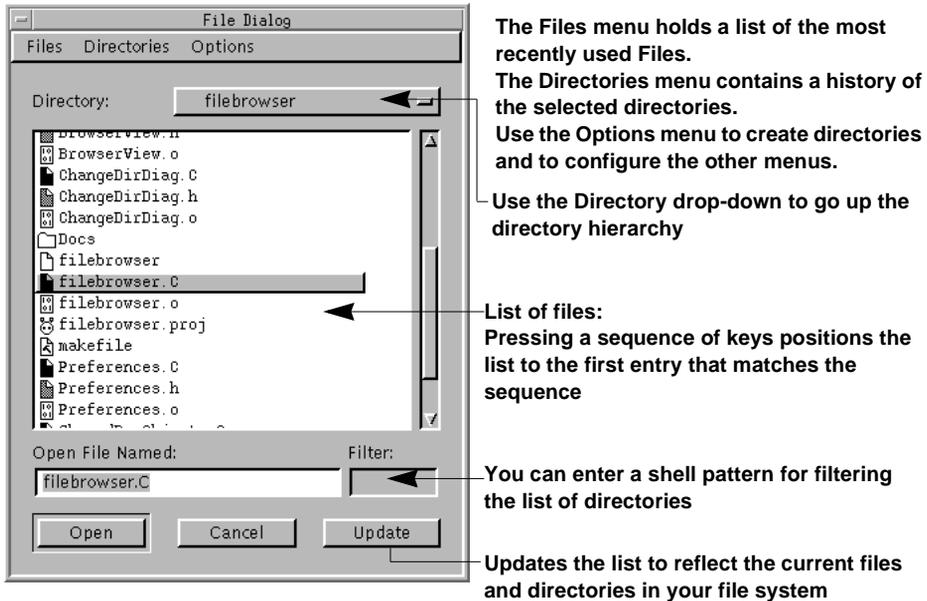


The **Options** menu lets you configure entries for the **Directories** menu, create directories, and select whether hidden files are displayed.

The **Directories** menu lists the most recent active directories. Choosing an entry from the list opens that directory and displays its contents in the Directory List. To remove entries, choose **Options > Edit Directories**.

File Dialog (Unix)

The File dialog is opened on save and load file operations.



The **Open File Named** field expands C-shell metacharacters like '~' and \$variables. Pressing <Return> in the text field selects the **Open** button.

Entering a C-shell regular expression in the **Filter** field and pressing the **Update** button filters the list to only those entries that match the filter.

Files menu

The **Files** menu lists the most recently opened files. Choosing a file from the list opens the selected file and then closes the File dialog. Files marked by a button remain in the list until they are removed in the options menu.

Directories menu

The **Directories** menu lists the most recent active directories. Choosing an entry from the list updates the directory.

Options menu

The **Options** menu serves to configure entries for the **Files** and **Directories** menus and allows creation of a new directory.

The **Create Directory** *directory* command creates *directory* in the current directory. This entry is only enabled if *directory* is entered in the editable text field.

Directory drop-down

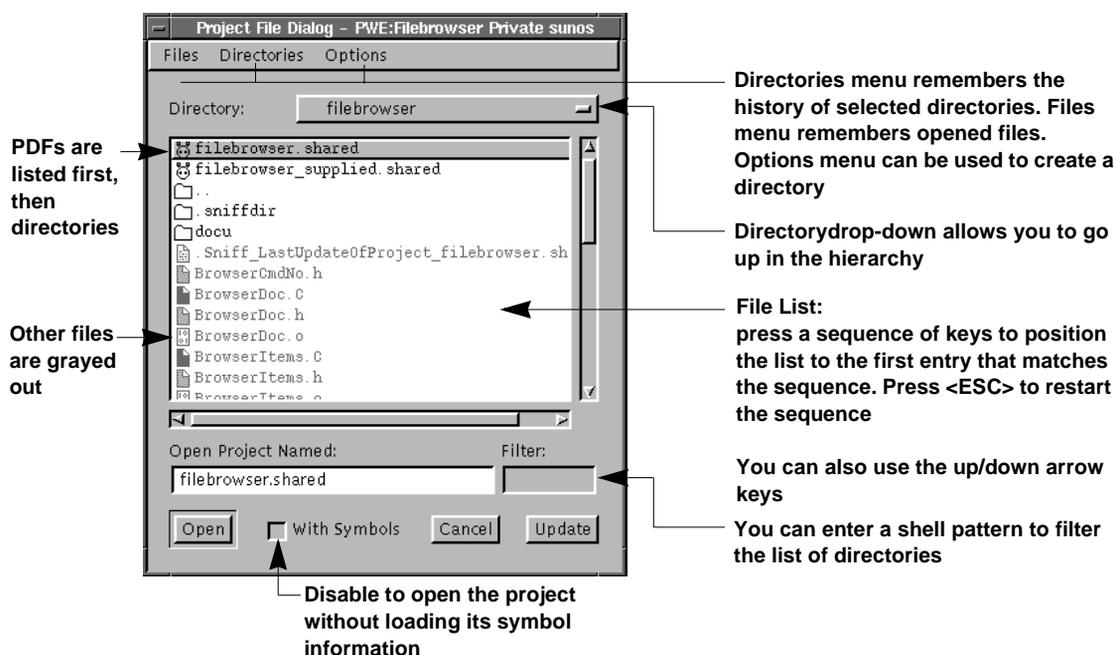
The **Directory** drop-down shows the list of directories in the path from the current directory to the root. You can click any directory in the path to open it.

Update Button

Updates the File List (useful when new files are created or deleted while the File dialog is open or when a regular expression is entered into the **Filter** field).

Project File dialog (Unix)

The Project File dialog appears whenever you want to open project description files (PDFs). It is very similar to the normal File dialog, except that it provides special options for opening PDFs.



Check boxes

- With Symbols** Opens projects with symbol information. For browsing and day-to-day development work, enable this check-box. If you do not need symbol information, e.g. to open large projects for making structural changes, clear this check box. Default: selected.
- Use Cache** When projects are closed, all necessary information for re-opening them is cached in a single file to speed up project opening. If **Use Cache** is selected, only this file is read. If not, all and the original Project Description Files (PDFs) are read, and all source files are checked.
Caution: The cached information can be **incorrect** if (1.) changes are made to projects and files between SNIFF+ sessions, that is, outside of SNIFF+, or (2.) if the preceding SNIFF+ session was terminated unexpectedly. Default: not selected.
- Open X-Ref-DB Read Only** Applies only if the database-driven cross reference system is used. The option is only available for the first Project to be opened in a given Working Environment. As soon as the first Project has been opened with **Read Only** database access, all subsequent Projects can also only be opened with **Read Only** database access in the current Working Environment and current SNIFF+ session. Default: not selected.

Buttons

- Open** Opens the selected project in the current working environment.
- Update** Updates the File List (which is useful when new files are created or deleted while the Project File dialog is open or when a regular expression is entered into the **Filter** field).

Files menu

The **Files** menu in the Project File dialog is slightly different from the one in the File dialog. In the Project File dialog, the **Files** menu displays the relative paths of (shared) projects.

When you select a PDF (Project Description File) in the **Files** menu, SNIFF+ first tries to open the project in your Private Working Environment. If the PDF is not located there, SNIFF+ searches for it in the shared source working environment. To override this default behavior of SNIFF+, type in the path of the PDF (in the field right above the **Open** button).

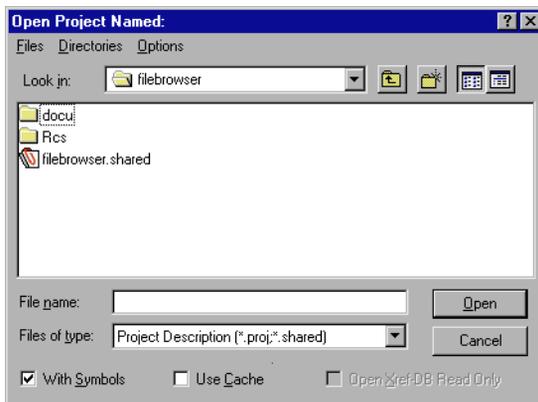
Directories menu

See [Directories menu — page 40](#).

Options menu

See [Options menu — page 40](#).

Project File dialog (Windows)



Check boxes

- With Symbols** Opens projects with symbol information. For browsing and day-to-day development work, enable this check-box. If you do not need symbol information, e.g. to open large projects for making structural changes, clear this check box. Default: selected.
- Use Cache** When projects are closed, all necessary information for re-opening them is cached in a single file to speed up project opening. If **Use Cache** is selected, only this file is read. If not, all and the original Project Description Files (PDFs) are read, and all source files are checked.
Caution: The cached information can be **incorrect** if (1.) changes are made to projects and files between SNIFF+ sessions, that is, outside of SNIFF+, or (2.) if the preceding SNIFF+ session was terminated unexpectedly. Default: not selected.
- Open X-Ref-DB Read Only** Applies only if the database-driven cross reference system is used. The option is only available for the first Project to be opened in a given Working Environment. As soon as the first Project has been opened with **Read Only** database access, all subsequent Projects can also only be opened with **Read Only** database access in the current Working Environment and current SNIFF+ session. Default: not selected.

Files menu

The **Files** menu in the Project File dialog is slightly different from the one in the File dialog. In the Project File dialog, the **Files** menu displays the relative paths of (shared) projects.

When you select a PDF (Project Description File) in the **Files** menu, SNIFF+ first tries to open the project in your Private Working Environment. If the PDF is not located there, SNIFF+ searches for it in the shared source working environment. To override this default behavior of SNIFF+, type in the path of the PDF (in the field right above the **Open** button).

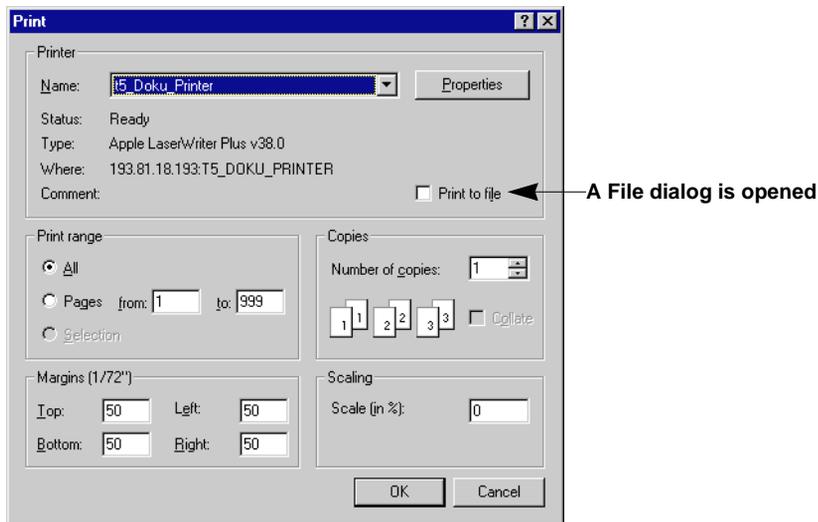
Directories menu

See [Directories menu — page 40](#).

Options menu

See [Options menu — page 40](#).

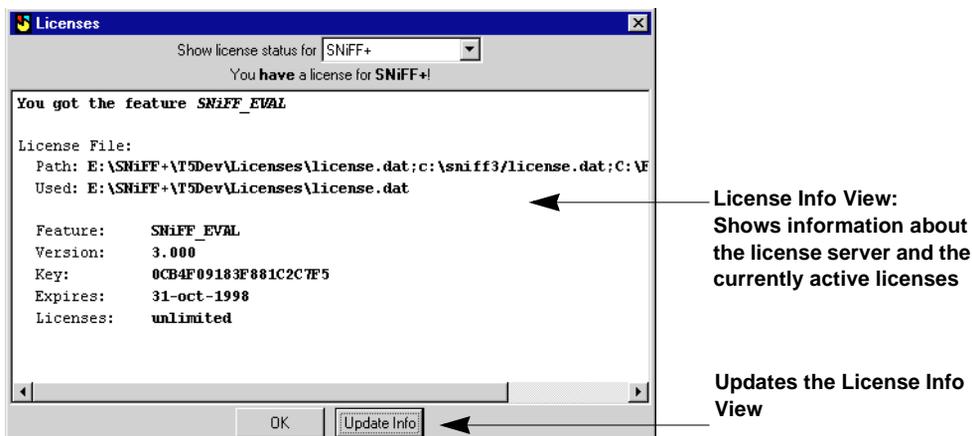
Print dialog



In the Print dialog, you can choose from among the available Windows printers. You can set the properties of the printers in the Print Manager (from the Windows Control Panel).

Licenses dialog

To open the dialog, choose the Launch Pad's **Help(?) > Licenses...** menu command. The Licenses dialog displays information about the floating license server and opens automatically when there is a problem in connecting to the license server process.



For information on obtaining and installing a licenses, please refer to the *SNIFF+ Installation Guide*.

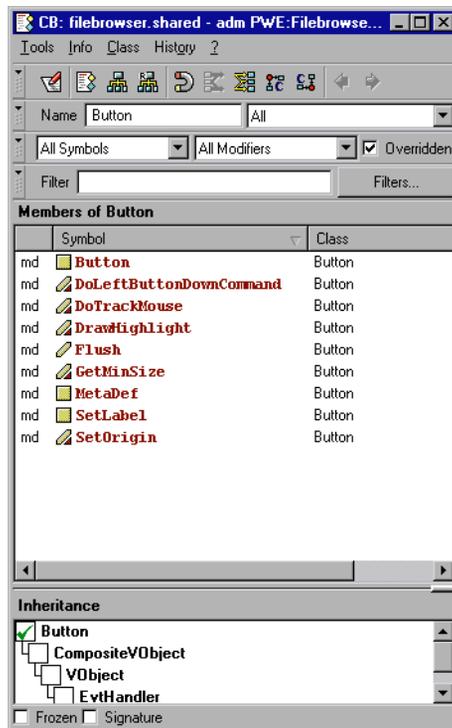
Class Browser

Introduction

The Class Browser shows the locally defined and inherited members of a class or structure. It provides a wide range of filtering possibilities based on the inheritance, visibility and type of the members.

The content of the list is determined by the settings of the Inheritance Graph view at the bottom of the tool, the **Filter** field and the various semantic filters.

- click on a class in an open tool and choose **Class > Browse class** from the menu, or
- choose **Tools > Class Browser** and enter the name of the class in the **Name** field.



Quick Reference

Icons in the Member List

Public	Member
	normal
	virtual
	is overridden
	overrides
	is overridden and overrides
	static

Icon colors in the Member List

Color	Visibility
 yellow	public
 blue	protected
 gray	private

Typeface in the Member List

- Typeface in the Member List corresponds to text highlighting in the Source Editor (you can set this in your Preferences).
- If members are shown in gray, this means that the member is not visible in the class being browsed.

Typeface in the Inheritance Graph

Typeface	Class
Normal	does not have members that fulfill current filter criteria
Bold	has members that fulfill current filter criteria
<i>Italics</i>	is abstract

Mouse clicks in the Inheritance Graph

- <CTRL>**click** on the name (not the checkbox) of a class lists members of that class only; members of all other classes are hidden.

Basic components

Name Field

Enter the name of a class you want to Browse.

Member List

For a description of the icons used in the Member List, please see [Quick Reference — page 48](#).

The Member List is determined by the settings of the various filters. The icons show attributes of class members.

Inheritance Graph

For a description of the typeface and mouse clicks in the Inheritance Graph, please see [Quick Reference — page 48](#).

The Inheritance Graph shows the graph consisting of the browsed class and its base classes. Select/clear the checkboxes in the graph to show/hide members of classes.

Filters

A number of filtering controls are provided at the top of the tool. Individual selections are possible in the various drop-downs and the **Overridden** check box, multiple combinations of filters can be selected using the Filters dialog.

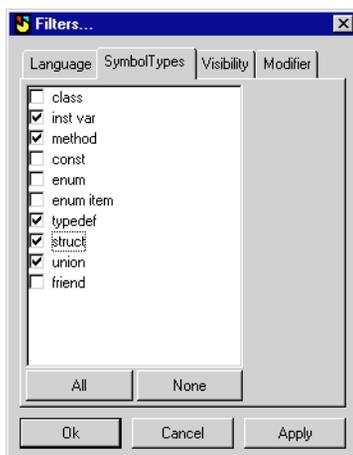
Filters... Button

The **Filters...** button opens the Filters dialog, where you can select multiple combinations of filters.

The Filters Dialog

- The **Apply** button applies the selected filters and leaves the dialog open.

- The **Ok** button applies the selected filters and closes the dialog.



The Class Browser's Filters dialog has four tabs, the elements in the tabs correspond to those in the drop-downs on the tools.

If multiple combinations are selected, the corresponding drop-downs show the entry, **Filtered...**

Selecting **Filtered...** in a drop-down opens the Filters dialog.

Overridden Check Box

Also overridden members are shown when the check box is selected.

Symbols drop-down

The **Symbols** drop-down specifies the type of symbols shown in the Member List.

All Symbols	Shows all members.
Filtered...	Means that multiple selections were made in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.

Visibility drop-down

The **Visibility** drop-down filters the symbol list according to the visibility of the members as seen from the class currently being browsed.

public	Shows only public members.
protected	Shows only protected members.
local private	Shows only local private members, that is, those visible within the class being browsed.
invisible private	Shows also private members of other classes, that is, also those that are not visible from the class being browsed.
All	Shows members of the classes checkmarked in the Inheritance Graph.
Filtered...	Means that multiple selections were made in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.

Modifiers drop-down

The **Modifier** drop-down filters the symbol list according to modifiers. Note that this includes also implicit modifiers, e.g., overriding or overloaded methods.

All Modifiers	Shows all members.
no modifiers	Shows only members without modifiers.
Filtered...	Means that multiple selections were made in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.
Language-specific modifiers	Shows only members modified by the selected modifier.

Filter field

Enter a Regular Expression here and hit <Return> to filter accordingly. See also [Regular expression filters — page 11](#)

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Signature

If the **Signature** check box is selected, the complete signature of each the member is displayed. This makes it possible, for example, to distinguish between overloaded methods.

Menus

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Please refer to [Class menu — page 22](#).

History menu

Please refer to [History menu — page 23](#).

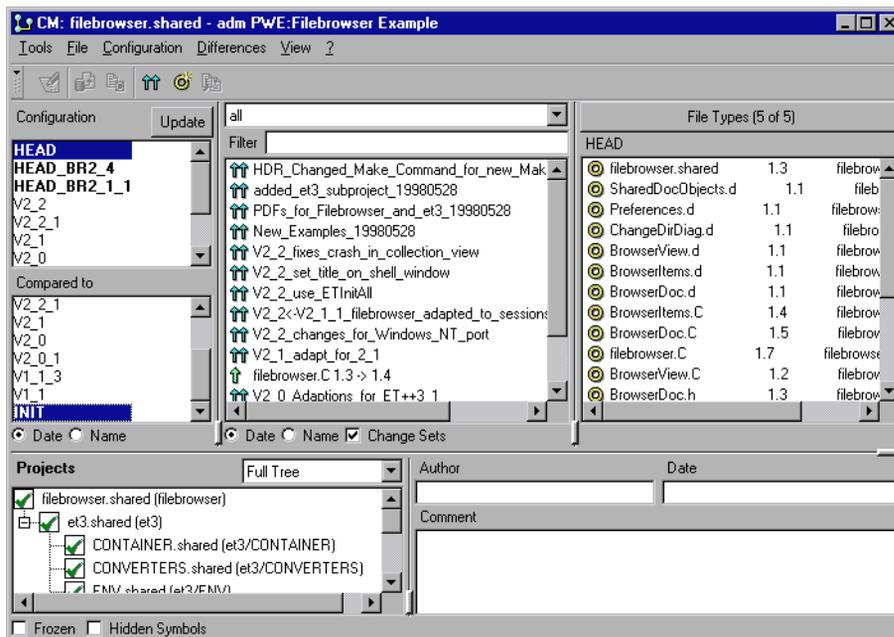
Help (?) menu

Please refer to [Help \(?\) menu — page 23](#).

Configuration Manager

Introduction

The Configuration Manager gives a structural and file-based overview of the changes between two configurations. You can also use it to create and manage branches.



Quick Reference

These icons are used to represent the types of changes between two configurations

Icon	Description
	A change set.
	A single unnamed version change on the branch selected in the Configuration List.
	The difference between two file versions, where each version is part of a configuration, and each configuration is on a different branch.
	A single unnamed version change on the branch selected in the Compared To List.
	A single unnamed version change on the branch selected in the Configuration List. The file has also been modified on the branch selected in the Compared To List.
	A new file added to a branch.
	A removed file.
	A version of a file that is part of the configuration selected in the Configuration List.
	A single unnamed backward version change on the branch that is selected in the Configuration List.
	A new version of a file that is the same as the last version.
	Head version of a branch.

Basic components

Configuration List and Compared To List

The Configuration List and Compared To List show all available configurations of the selected projects in the Project Tree. If only one configuration is selected in the Configuration List, the File List shows all the files and versions that are part of this configuration. If two configurations are selected (one in the Configuration List and one in the Compared To List) the Change List and the File List show the differences between those two configurations. The configurations can be ordered by date or name.

Change List

For a description of the icons used in the Change List, please see [Quick Reference — page 56](#).

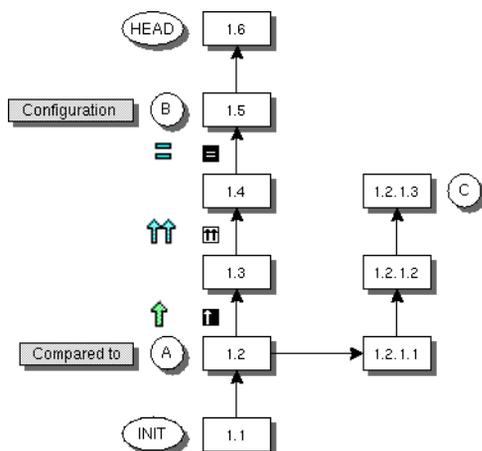
The Change List displays the changes between two configurations selected in the Configuration List and Compared To List. An entry in the Change List can have one of a number of forms, depending on the type of change it represents. For example, the following entry represents a single, unnamed change on a branch that is selected in the Configuration List:

```

↑ filebrowser.C 1.3 -> 1.4
├── Old version > new version
└── Change icon
  
```

All entries start with a change icon. This icon indicates the type of change made to files in the two configurations. The change icon is followed by the name of the file affected by the change. Depending on the type of change, the name of the file is followed by either two version numbers, one, or none.

The following diagram shows how the icons on [page 56](#) are used if two configurations on one branch are selected:



Change Type drop-down

You can select what type of changes to display with the **Change Type** drop-down

all	Displays all changes.
change sets	Displays only change sets.
different branches	Displays only the differences between two branched configurations (shown by the icon ).
new files	Displays files that have been added
removed files	Displays files that were removed to a branch. Note that SNiFF+ does not maintain information on files that have been physically removed from projects.

Change Sets check box

The **Change Sets** check box determines whether change sets are displayed in the Change List or not.

If the check box is enabled, change sets are displayed and the changes to the individual files that are part of the change sets are not displayed.

If the check box is disabled, file changes are displayed together with version numbers and change sets are not displayed.

History information

The lower right corner of the Configuration Manager displays information about the entry selected in the Change List or File List.

Author	Displays the author of the selected change.
Date	Displays the check-in date of the selected change/version.
Comment	Displays the comment that has been checked in together with the selected change/version. If a file is selected, the file description is displayed.

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Hidden Symbols check box

When this check box is enabled, deleted versions of files are displayed in italics in the Configuration Manager and Project Editor, and menu items relating to file versions are disabled for deleted file versions.

Menus

File menu

These commands are enabled only when you select a version of a file in the File List

Files menu command	Description
Show File	Loads the file into a Source Editor.
Check Out...	Opens the Check Out dialog, in which you can select the version of the file that you want to check out.
Delete Version...	Opens a Delete Version dialog, in which you can delete the version that is selected in the File List. If a version is selected in the Version Tree, this version is filled in as the default value of the dialog's Version field. If no version is selected in the Version Tree, SNIFF+ uses the Default Configuration of the working environment in which you opened the project (see <i>User's Guide</i>).
Replace Comment...	Opens a Replace Comment dialog, in which you can change the comment of the version that is selected in the File List.
Show History Info	Opens a Project Editor and shows the history information of the selected file. If a Project Editor is already open (and its Frozen check box is not enabled), it is used to show the history information of the selected file.

Configuration menu

Configuration menu command	Description
Show 3-Way Differences	Shows the three-way differences between the configurations selected in the Configuration and Compared To Lists with respect to the common ancestor configuration. The two selected configurations must be located on different branches
Merge Branch Configuration...	Merges the second configuration into the first one. This command checks out and locks the first configuration and merges the files of the second configuration into the newly checked-out files. The Diff/Merge tool is opened with a list of all files to be merged. This command checks whether the second configuration is a branch configuration of the first before checking out the files.
Check Out Configuration...	Opens a dialog for checking out the configuration that is selected in the Configuration List. After pressing one of the lock buttons, all files belonging to the configuration are checked out.
Freeze Head...	This menu entry appears in the Configuration menu when HEAD is selected in the Configuration List. When you choose this command, a Freeze Head dialog appears. After you press Ok , all HEAD versions of the main branch will have the configuration name that you entered in this dialog.
Freeze Default Configuration...	When you choose this command, a Freeze Default Configuration dialog appears. After you press Ok , the default configuration of all versions of the main branch will be saved under the configuration name that you entered in this dialog.

Configuration menu command	Description
Freeze Head Of...	<p>This menu entry appears in the Configuration menu when the HEAD configuration of a branch is selected in the Configuration List. When you choose this command, a Freeze Head Of dialog appears.</p> <p>After you press Ok, all HEAD versions of the branch will have the configuration name that you entered in this dialog. Note that only files that are part of the selected configuration will be frozen.</p> <p>The Freeze Head Of... command is disabled for the SCCS version control system.</p>
Freeze Configuration As	<p>This menu entry appears in the Configuration menu when a non-HEAD configuration is selected in the Configuration List. When you choose this command, a Freeze Configuration As dialog appears.</p> <p>After you press Ok, all configurations that are associated with the selected configuration will have the configuration name that you entered in this dialog. Note that only files that are also part of the selected configuration will be frozen.</p>
Rename Configuration...	<p>Opens a Rename Configuration dialog, in which you can rename the configuration that is selected in the Configuration List.</p> <p>After you press OK, the selected configuration is renamed.</p>
Delete Configuration Name...	<p>Opens a dialog in which you are asked to confirm the deletion of the configuration that is selected in the Configuration List. After you press Ok, the selected configuration name is deleted.</p> <p>Note that the Delete Configuration Name... command deletes only the configuration name from the repository and does not delete any versions or files that are associated with it.</p>

Differences menu

These entries are enabled when you select a change from the Change List

Differences menu command	Description
Show Differences...	<p>Opens a Show Differences dialog, in which you can specify the type of differences (2-way or 3-way) to be displayed between the entries selected in the Change List and the Configuration List:</p> <p>2-Way — Opens a Diff/Merge tool which shows the differences between the versions of the entry that is selected in the Change List.</p> <p>3-Way — Opens a Diff/Merge tool which shows three-way differences between the versions of the entry that is selected in the Change List with respect to their common ancestor.</p> <p>Cancel — Cancels the command and closes the dialog.</p>
Merge Differences...	<p>Opens a Merge Differences dialog, in which you can specify the type of differences (2-way or 3-way) between the selected entries in the Change List and the Configuration List that are to be merged:</p> <p>2-Way Merge — Opens a dialog for checking out the files of the configuration that is selected in the Configuration List. The Diff/Merge tool then appears and the differences between the working file (checked out file) and the entry that is selected in the Change List are displayed.</p> <p>3-Way Merge — Opens a dialog for checking out the files of the configuration that is selected in the Configuration List. The Diff/Merge tool then appears and the 3-way differences between the working file (checked out file), the entry selected in the Change List, and their common ancestor are displayed.</p> <p>Cancel — Cancels the command and closes the dialog.</p>
Show All Change Set	<p>Displays the change sets of all of the configurations in the Change List.</p>
Check Out...	<p>Opens a Check Out dialog for checking out the entry that is selected in the Configuration List. If the entry is a change set, all file versions that are part of the change set are checked out.</p> <p>After you press one of the lock buttons, the files that are part of the configuration are checked out.</p>

Differences menu command	Description
Rename Change Set...	Opens a Rename Change Set dialog, in which you can rename the selected change set. After you press Ok , the selected change set is re-named.
Delete Change Set Name...	Opens a dialog in which you are asked to confirm the deletion of the change set that is selected in the Change List. After you press Ok , the name of the selected change set is deleted. The Delete Change Set Name... command deletes only the name of the change set from the repository and not any versions or files that are associated with it.

View menu

Please refer to [View menu — page 23](#).

Cross Referencer

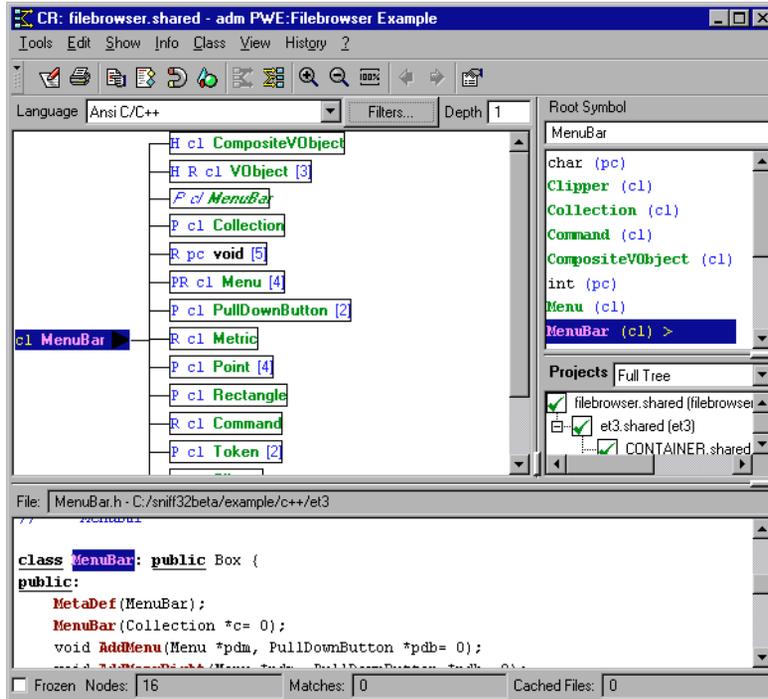
Introduction

The Cross Referencer provides symbol cross reference information. All kinds of cross references, including call graphing, can be visualized.

Basically, the Cross Referencer offers three kinds of symbol cross references:

- **Function body cross referencing** — Displays all symbols that a symbol refers to or displays all symbols that are referred to by another symbol. Call graphing is a special case of function body cross referencing in which the graph is filtered to show only functions and method calls. Another example of function body cross referencing is a query where all symbols are displayed that access a variable in write mode. Questions like: “Which symbols are referenced by this function?” or “Who calls this method?” or “Who accesses this variable in write mode” can be answered.
- **Component analyzing** — Displays all types that are components of other types, or displays all classes or structures that have a type as a component. An example of a special case component query is to show all types that have a primitive C data type as a component. Questions like: “What are the components of this class displayed to level 5?” or “Who has an instance variable that points to this type?” or “What classes have integer instance variables?” can be answered.
- **Interface cross referencing** — Displays all types that are part of a type's interface, i.e. that are either a return or a parameter type, or displays all types that use another type in their interface. Questions like: “Who returns a character pointer?” or “Who uses an instance of the object in its parameter list?” or “What are the return and parameter types of this class?” can be answered.

The Cross Referencer can be opened from the **Tools** menu or by using the four query commands in the **Info** menu.



Quick Reference

Typeface in the Graph view

Typeface	Symbol
Normal	appears once in the Graph view
<i>Italics</i>	appears more than once in the Graph view

Mouse clicks in the Graph view

- **Click** on a symbol node to show it in the integrated read-only Code view.
- **Double-click** on a symbol node to open a Source Editor positioned at the symbol.
- **<SHIFT>click** on a symbol node to show the reference in the integrated read-only Code view.
- **<SHIFT>double-click** on a symbol node to open a Source Editor positioned at the reference. The Source Editor can then step through subsequent references with the **Show > Next Match** menu command.
With a **<SHIFT>double-click** on the first node in a reference list, you can then step through all of the references in the Source Editor.
- **<CTRL>double-click** shows the declaration of the selected symbol in the Source Editor.
- **<SHIFT><CTRL>double-click**. If a Source Editor is already open, this command freezes the already opened editor and shows the reference in a new Source Editor.

Abbreviations

Access Indicators

Access indicator	Access mode
r	Read
w	Write
H	Used as component type (has-a relationship)
P	Used as a parameter type
R	Used as return value type

Type Indicators

Type indicator	Symbol type
cl	Class
cd	Constructor or destructor
iv	Instance variable
me	Method
f	Function
v	Global variable
co	Constant variable
ma	Macro
en	Enumeration
ei	Enumeration item
td	Typedef
te	Template
st	Structure
un	Union
pc	Primitive C data type (e.g., char, int, float)
ud	Undefined symbol (e.g., the <code>printf</code> function, which is not defined in one of the loaded projects)
bo	builtin operator
bf	builtin function

Basic components

Graph view

For a description of the abbreviations and typeface used in the Graph view, please see [Quick Reference — page 67](#).

The Graph view shows the results of cross reference queries. It always has one root symbol and extends from left to the right. The view shows either forward or backward references (indicated by arrow icons in nodes). The graph is filtered by the Filter settings. Each node in the Graph view represents a symbol. The type/access indicator string at the head of each node shows additional information about the type of the symbol and the kind of reference (see [Access Indicators — page 67](#) and [Type Indicators — page 68](#)).

Each symbol is expanded only once— the graph is therefore actually a tree. A node that appears repeatedly in the tree is shown in italics to denote that this node has a probable expansion somewhere else. You can navigate to the expanded node by clicking it in the Symbol List.

Code view

References to the symbol selected in the Graph view are displayed in the read only Code view. To jump to the reference in the Source Editor, <CTRL>double-click on the symbol in the Graph view.

Root Symbol field

You can directly enter a symbol name in the Root Symbol field. When you press <Return>, the existing graph is reset and the entered symbol is placed as the new root symbol in the graph. After that, choose the appropriate reference command from the **Info** menu.

Symbol List

The Symbol List lists all nodes in the graph in alphabetical order. Multiple nodes in the graph are only shown once in the list. When a node is selected in the graph, the corresponding entry in the list is also selected and vice versa.

Filters

Filters.. button

This evokes the X-Ref Filter dialog, please refer to [X-Ref Filter dialog — page 73](#).

Project Tree

The Project Tree can be used to limit the displayed references in a subsequent cross reference query.

Language drop-down

In the **Language** drop-down, you can choose the language you want to query.

Depth field

The (integer) value in the **Depth** field specifies how many levels will be expanded during the next query. Large values naturally result in long query times and large graphs.

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Nodes

Indicates the number of nodes in the Graph view.

Matches

Indicates the number of matches in the Graph view.

Cached Files

Indicates the number of cached files.

Menus

Edit menu

Edit menu	Description
Replace Referencers of <i>SymbolName</i>	Starts the Retriever in Replace Only mode where you can globally change the references.

Show menu

Show menu	Description
Next Match	Displays the next reference to the symbol in the Code view. You can use this command to step through subsequent references.
Reference	Opens a Source Editor and is positioned at the reference of the symbol.
Declaration of <i>SymbolName</i>	Opens a Source Editor and is positioned at the declaration of the symbol.
Implementation of <i>SymbolName</i>	Opens a Source Editor and is positioned at the implementation of the symbol.

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Please refer to [Class menu — page 22](#).

View menu

You can use the **View** menu entries to specify the layout of the Graph view:

Commands	Description
Select Project Set	The names of the project sets (defined in the Save Project Set dialog) are displayed. You can then select the project set that you want to work with.
Save Project Set	You can select a set of projects to view. When the Save Project Set command is chosen, a Save Project Set dialog is opened in which you enter a name for the project set. These project sets are then displayed in the Select Project Set sub-menu.
Remove Project Set	Removes the selected project set, displayed with a tick to the left of it, from the Select Project Set sub-menu.
Filter...	Opens the Filter dialog. See also X-Ref Filter dialog — page 73 .
Layout > LeftRight	Redraws the Graph view, displaying it from left to right (default).
Layout > TopDown	Redraws the Graph view, displaying it from top to bottom.
Layout > Indented	Redraws the Graph view as an indented list with rectangular connecting lines.
Layout > Direct	Draws the connecting lines directly (default).
Layout > Diagonal	Draws the connecting lines diagonally.
Layout > Orthogonal	Draws orthogonal connecting lines.
Start from <i>SymbolName</i>	Makes the currently selected symbol the root symbol of the graph. All other nodes are deleted from the graph.
Hide Subnodes of <i>SymbolName</i>	Hides all subnodes of the selected symbol.
Show Restricted Tree of <i>SymbolName</i>	Shrinks the graph in such a way that only parent and subnodes are displayed. All other nodes are deleted from the graph.
Zoom commands	Allows you to zoom in and out.

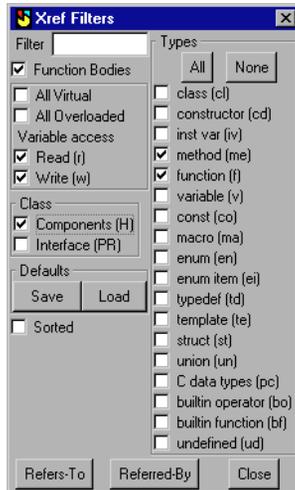
History menu

Please refer to [History menu — page 23](#).

X-Ref Filter dialog

You can use the X-Ref Filter dialog to set the scope of the query. The dialog is non-modal and can be opened by doing the following:

- choose **View > Filter...** or
- press the **Filter...** button:



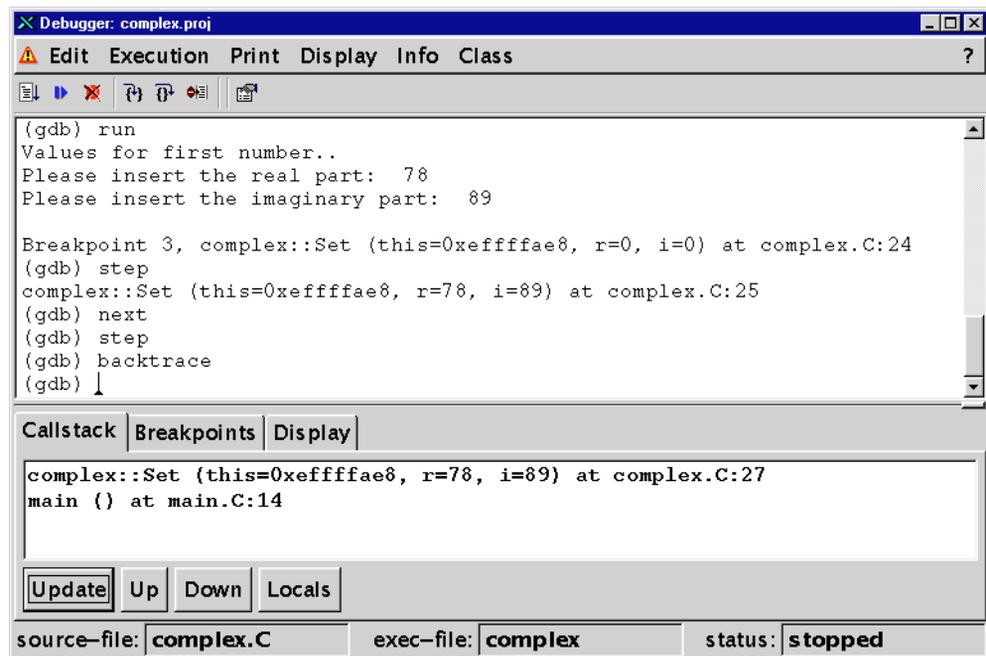
Elements	Description
Filter	Allows you to filter the graph with a regular++++ expression. The filter settings are used for the next query. For more information about regular expressions, see Regular Expressions in SNIFF+ — page 307 .
Function Bodies	Defines whether function bodies are to be included in queries. If enabled, the function body cross reference information is loaded by SNIFF+. Note that cross reference tables are loaded incrementally on demand in order to save memory. Function body cross reference information is not needed for component or interface cross referencing.
All Overloaded	Specifies whether each member of a set of overloaded items (methods or functions) should be referenced during the next query. If enabled, all overloaded items are taken into consideration. If disabled, only one specific overloaded item is referenced.

Elements	Description
Read (r)	Specifies whether read accesses to variables should be included in the next query. If disabled, no read accesses to variables will be shown.
Write (w)	Specifies whether write accesses to variables should be included in the next query. If disabled, no write accesses to variables will be shown.
Components (H)	Defines whether class component (has-a) relationships should be included in the next query. If enabled, component relationships will also be shown.
Interface (PR)	Defines whether class interface usage should be included in the next query. If enabled, Parameter and Return types will also be shown.
Save Default	Saves your current Filter Settings as the default setting. Note that if you work in projects with other languages, your settings will be incorrect.
Load Default	Loads the saved default setting. Note that if you work in projects with other languages, your settings will be incorrect.
Types	Specifies which types are to be included in the next query. Pressing All enables all check boxes; pressing None disables all check boxes.
Refers-To	Starts a Refers-to query.
Referred-By	Referred-by query.
Close	Closes the X-Ref Filter dialog.

Debugger (Unix and Java)

Introduction

The SNIFF+ Debugger is a graphical front-end to debuggers like gdb, dbx and SNIFF+s sniffjdb Java debugger. It is integrated into the environment by using the Source Editor to display the source code and the current stack frame. It forwards all commands to the underlying debugger and interprets its output.



Starting the Debugger

The Debugger can be started from the Target dialog and the **Tools** menu.

Note that the Debugger can be started only if the target name is specified in the project attributes and the target file exists in the project directory and is executable. See also [Build Options view — page 169](#)

Multiple simultaneous Debugger sessions

In SNIFF+, a *session* refers to a process on UNIX or Windows NT/95. You can have multiple Debugger sessions running in different SNIFF+ sessions.

Multiple Debugger sessions in different SNIFF+ sessions

If you want to select the same target name in multiple Debugger sessions, you will have to start new SNIFF+ sessions and launch the Debugger in each of them.

- Start a SNIFF+ session by executing

```
% sniff -s <sniff_session_name>
```

on the command line.

<sniff_session_name> is a *session id* that uniquely identifies each SNIFF+ session. The session id is a string and cannot contain any blank spaces within it.

If you start SNIFF+ without the **-s** option, the value of the `SNIFF_SESSION_ID` environment variable is used by default for the session id. If `SNIFF_SESSION_ID` is not set, `session0` is used for the session id. See also [SNIFF_SESSION_ID — page 255](#)

- In each SNIFF+ session, open a Source Editor tool and launch the Debugger from it.
- Select the target that you want to debug.

Supported debuggers

This SNIFF+ release supports the following debugging systems:

Platform	Debugger & Version
AIX 4.2	gdb 3.x gdb 4.x dbx 3.1
Alpha-DEC 3.0	gdb 3.x gdb 4.x dbx 3.11.8 decladebug 4.0-7
HP/UX 10.x	gdb 3.x gdb 4.x dde 3.25A.P1 xdb A.09.01
IRIX 5.3	gdb 3.x gdb 4.x dbx 3.19

Platform	Debugger & Version
Linux (ELF)	gdb 3.x gdb 4.x
Sinix 5.43 (Reliant Unix)	gdb 3.x gdb 4.x dbx 2.0C
Solaris 2.4 or newer	gdb 3.x gdb 4.x dbx
Unixware 7	gdb 3.x gdb 4.x
All platforms	Java debugging adaptor

Selecting a debugger back-end

You select the debugger back-end as well as edit debugger-specific preference settings in the Platform view of the Preferences. To open the Preferences dialog, choose **Tools > Preferences**. For more information, please refer to [Platform view — page 157](#).

Status line

The status line of the Debugger differs from the status lines of other tools and does not have a **Frozen** button. The Debugger status line has the following elements:

source file	Current source file.
exec file	Target file name of the executable.
status	State of the debugger.

The Debugger can be in one of four possible states:

not running	The back-end debugger is running, but the application being debugged is not running.
running	The application being debugged is running.
stopped	The application being debugged has stopped (e.g., due to a breakpoint).
idle	The Debugger has no debugger back-end process running.

Menus

Tools menu

For a description of the Tools menu, please refer to [Tools menu — page 13](#).

Edit menu

Edit menu command	Description
Cut	Cuts the current selection to the paste buffer.
Copy	Copies the current selection to the paste buffer.
Paste	Pastes the paste buffer contents to the current cursor position or selection. This command is enabled when the paste buffer is not empty.
Clear	Clears the complete Shell buffer.
Show Error	Filters the line containing the cursor. If it understands the error message format, it opens a Source Editor and displays the corresponding source code. The section Error formats — page 288 explains how to extend the list of understood message formats.

Execution menu

Execution menu commands	Description
Next	Single-steps over the next function/method.
Step	Single-steps into the next function.
Run	Runs the application being debugged from scratch. You can enter arguments for Run in the Program Arguments dialog that appears when you execute the command.
Cont	Continues execution (only enabled if the application is stopped).
Step Out	Finishes the current function or method.
Interrupt	Interrupts the debugged application process (entry is only enabled if the application is running). This command is not available for remote debugging.

Execution menu commands	Description
Attach	Opens the Process Picker dialog, in which you can choose a process from the list of running processes. <code>sniffgdb</code> then attaches to the chosen process. The Attach command only works if your debugger back-end supports attaching. This command is not available for remote debugging. Note: This command is not supported on Linux platforms.

Print menu

Print menu commands	Description
Print <i>selection</i>	Displays the value of the current selection in the main view. The selection must be known in the current stack frame
Print <i>*selection</i>	Displays the value of the memory location pointed to by the current selection in the main view. The selection must be known in the current stack frame and must point to a memory location for which the type is known to the Debugger.
Print <i>*this>selection</i>	Displays the value of the memory location pointed to by the current selection in the main view. The selection must be an instance variable of this object and must point to a memory location for which the type is known to the Debugger.
Print <i>this>selection</i>	Displays the value of the selection in the main view. The selection must be an instance variable of this object.
Print <i>*this</i>	Displays all instance variables of this object.

Display menu

Display menu commands	Description
Display <i>selection</i> Display <i>*selection</i> Display <i>*this>selection</i> Display <i>this>selection</i> Display <i>*this</i>	Continuously outputs the values of the displayed variables in the Display tab. Selections and semantics are the same as in the Print menu (see above). Outputs the values after the next single step command or program stop.

Display menu commands	Description
Undisplay selection	Deletes a variable from the list of displayed variables.

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Please refer to [Class menu — page 22](#).

Tabs

Callstack tab

When the Callstack tab is selected, the following buttons appear

:

Update	Lists all stack frames beginning with the current one and going up to the main function entry (the list is also called call hierarchy). Frames for which symbol information is loaded are displayed in boldface.
Up	Goes one stack frame up in the call hierarchy. A reusable Source Editor is automatically positioned at the source location of the new stack frame.
Down	Goes one stack frame down in the call hierarchy. A reusable Source Editor is automatically positioned at the source location of the new stack frame.
Locals	Displays all local variables of the current stack frame in the main view.

Breakpoints tab

The Breakpoints tab displays a list of currently active breakpoints and allows their deletion or loading of their source code into a Source Editor. When the Breakpoints tab is selected, the following buttons appear:

Show	Loads the source containing the selected breakpoint into a Source Editor.
-------------	---

Clear	Deletes the selected breakpoint.
Status	Displays status information about the breakpoints in the main view.

Display tab

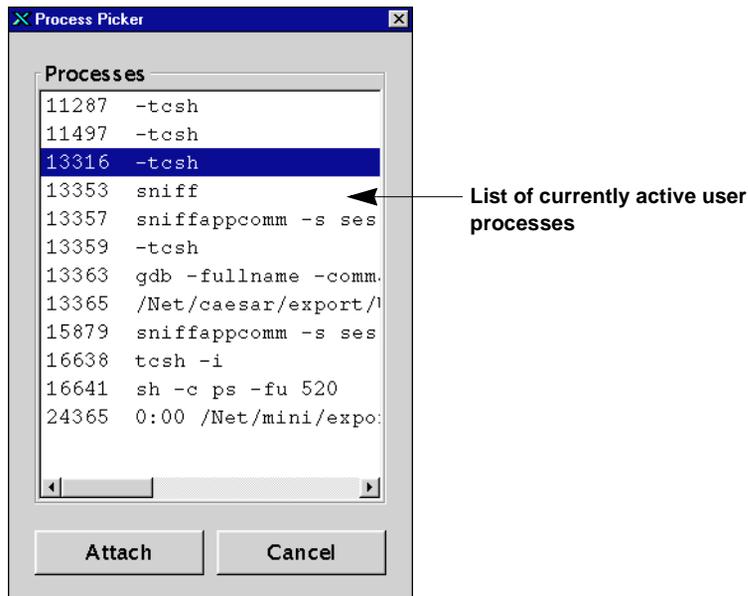
When the Display tab is selected, the following buttons appear:

Undisplay	Deletes a variable from the list of displayed variables. The selection must refer to a displayed variable.
Status	Displays status information about the displayed variables in the main view.

Dialogs

Process Picker dialog

You can open the Process Picker dialog by choosing **Execution > Attach**. In the dialog, you can select a process from the list of currently running user processes. After you have selected a process, the debugger back-end attaches to it. This command is not available for remote debugging.

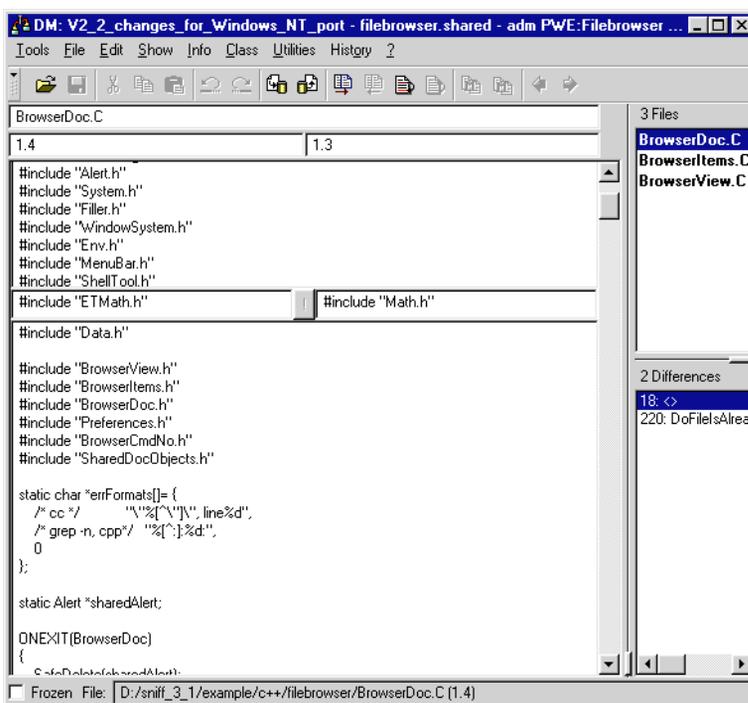


Diff/Merge tool

Introduction

The Diff/Merge tool allows you to show and merge differences between files and versions of files. The Diff/Merge tool offers two- or three-way differences. Three-way differences are important for investigating changes made by two developers to the same file. In such a case you want to look at the two file versions compared to a common ancestor.

The following illustration shows the Diff/Merge tool displaying the differences between two versions of a file:



The Diff/Merge tool can also be used for editing files. However, we suggest using the Source Editor for editing, as it supports many more editing features than the Diff/Merge tool.

Quick Reference

Merge Button symbols

The following table describes the symbols found on the **Merge** button.
(In the table, `HEAD` refers to the latest version of a file as maintained by your version control tool, and `INIT` refers to the initial version of the file.)

Merge button symbol	Description
=	The changes are the same in both versions. There is no need to merge anything.
?	Here is a conflict. The same lines in both versions have been changed.
<	The version in the right column has changed. The version in the left column has not changed. If you click on it, the version in the left column is restored.
:	The version in the left column has changed. The version in the right column has not changed. If you click on it, the version in the right column is restored (normally you don't want this since you will restore the original).
!	If the left most file is not writable or if you look at a difference between versions in the repository, ! is the read only version of <. This means that clicking on the button has no effect.
#	If the left most file is not writable or if you look at a difference between versions in the repository, # is the read only version of ?. This means that clicking on the button has no effect.

Symbol background color

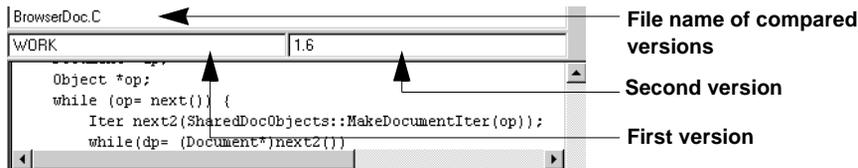
In addition, the following background colors have been added to emphasize the above.

Symbol background color	Description
red	CAUTION: Here is a conflict!
green	It is advisable to click on the merge button.
black	No need to merge anything.

Basic components

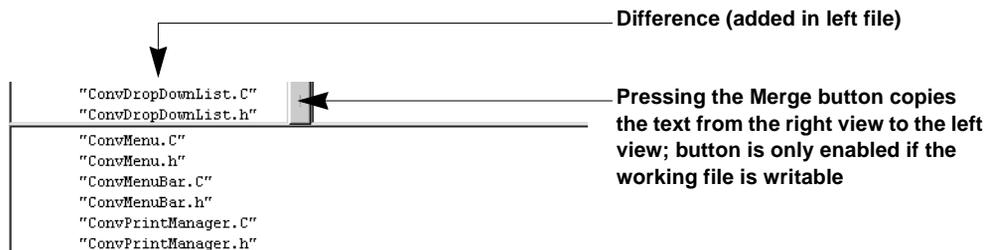
Header

The Header shows which files and which versions of the files are being compared. If two files are compared, the header displays the two filenames; if versions are compared, the filename and the two versions are displayed (as shown in the illustration below). The special version `WORK` denotes the current working file; the version `HEAD` denotes the latest version of the file. A third version that can be used for comparison is `SHARED`, which refers to the version of the shared file (in the Shared Source Working Environment). Please refer to the Glossary for a definition of working file and shared file.



Differences View

The Differences View displays the differences in the compared files/versions. Areas of code containing differences are split into two views and separated by a **Merge** button. An area where the two files/versions are the same is not split.



The Differences View shows two- or three-way differences. The **Merge** button between the working file and the compared file(s)/version(s) allows you to merge differences into the working file. If the working file is not writable, or if two versions of the same file are being compared, the **Merge** button is disabled.

Merged differences are shown in italics in the Differences List.

File List

The files shown in the File List depend on your selections in other views. The list contains only one file if the Diff/Merge tool compares two versions of a single file.

Clicking on a file loads the file into the Diff/Merge tool and shows the differences between the corresponding versions.

Differences List

The Differences List shows all the differences between the files/versions. The line number and the name of the symbol that contains the difference are provided.

Entries in *italics* indicate a difference that has already been merged.

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Menus

File menu

Please refer to [File menu — page 15](#).

Edit menu

Please refer to [Edit menu — page 16](#).

Show menu

Please refer to [Show menu — page 18](#).

Info menu

Please refer to [Info menu — page 20](#)

Class menu

Please refer to [Class menu — page 22](#).

Utilities menu

Utilities menu command	Description
Merge	Merges the current difference into the working file. For three-way merges, the first file is merged into the working file. If you want to merge the second file, you have to use the Merge buttons.
Merge All	Merges all differences into the working file. For three-way merges, all differences in the first file are merged into the working file. If you want to merge the second file, you have to use the Merge buttons.

History menu

Please refer to [History menu — page 23](#).

Documentation Editor

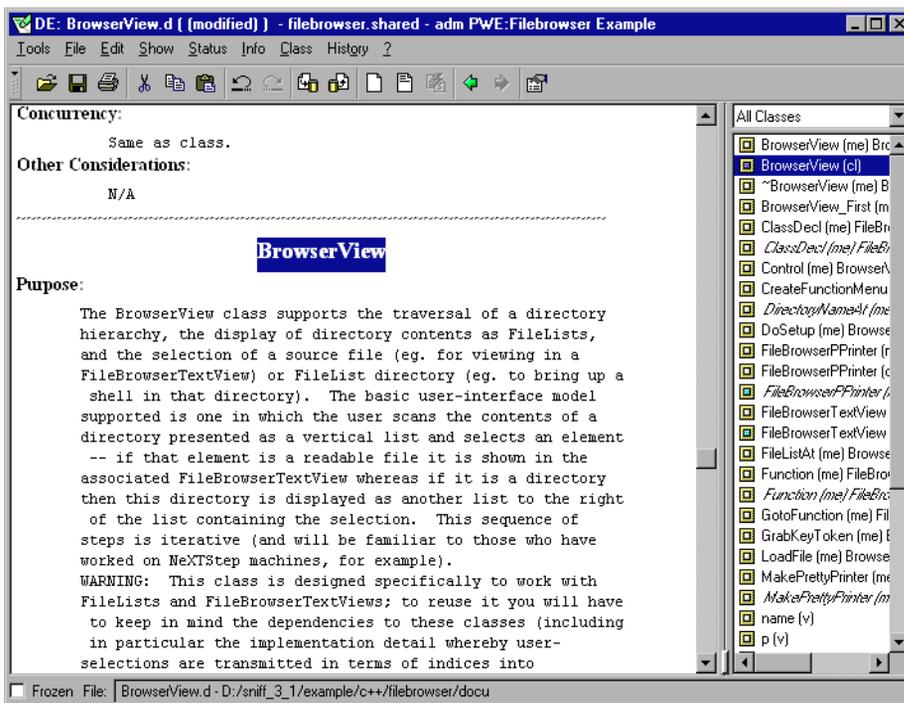
Introduction

The Documentation Editor supports the iterative and incremental generation, writing and maintenance of source code documentation. You can use its hypertext-like browser to quickly navigate between source and documentation. Furthermore, you can freely define the structure and contents of the generated documentation.

You can open an instance of the Documentation Editor by:

- selecting a documentation file from the File List in the Project Editor and double-clicking it, or by
- choosing the **Info > Show Documentation of *symbol*...** in any of the browsing tools (except for the Project Editor)

In the following illustration, the documentation file `BrowserView.d` has been loaded into the Documentation Editor. You can see the documentation frame for the class `BrowserView` in the tool's Documentation View.



Quick Reference

Icons in the Symbol List

These icons show the status of a symbol in the Documentation Editor. They are also used in the Documentation Synchronizer's File List and Project Tree where they apply to the documentation status of all the symbols in a file and in a project, respectively.

Icon	Symbol is
 (white)	undocumented
 (green)	partially documented
 (blue)	fully documented

Typeface

The typeface indicates the following about a symbol:

Typeface	Symbol
normal	can be browsed, documentation is valid
<i>italics</i>	cannot be browsed, documentation is obsolete

Mouse clicks

- **Click** on a symbol name to jump to the Documentation Frame.
- **Double-click** on a file name to jump to the source file.
- **Triple-click** in the Documentation View to select the entire documentation for a symbol.
- **Triple-click** on a section identifier to select a section.
- **Triple-click** in the section text to select the section text.

Modes of operation

The Documentation Editor operates in one of two modes (set in your Preferences — [Documentation Editor view — page 140](#)):

- **Browsing mode (read-only)** — Usual mode for simply browsing documentation files. When the Documentation Editor is in browsing mode, no changes can be made to existing documentation, and obsolete documentation will not be displayed.
- **Editing mode (read/write)** — Mode for documenting a software project. In this mode, you can generate documentation files from your source code.

Basic components

Documentation View

The Documentation View contains the documentation of source code at the symbol level. The documentation of a particular symbol is contained within *separators*. A symbol's documentation starts with the *symbol name* and *symbol signature* (if it has one). The actual documentation of the symbol follows in the *documentation body*, which is split into a series of *sections*. Each section begins with a *section identifier*. Its *section text* follows on the next line.

Symbol List

For a description of the icons used in the Symbol List, please see [Quick Reference — page 90](#).

The Symbol List displays the list of symbols that are documented in the documentation file. The list contains one or more symbols, depending on whether the documentation file has been generated for one symbol or an entire file.

The icons that precede the symbols in the Symbol List and the typeface of the symbols indicate what their documentation status is.

A typical entry in the Symbol List looks like this:

```
📄 BrowserView (me) BrowserView
```

It consists of a status icon, the symbol name and, in parentheses, the symbol type. For a list of the various symbol types, please refer to [Graph view — page 69](#).

Class drop-down

You can use the **Class** drop-down to specify which classes of symbols are to be displayed in the Symbol List.

Menus

File menu

Please refer to [File menu — page 15](#).

Edit menu

Please refer to [Edit menu — page 16](#).

Show menu

Please refer to [Show menu — page 18](#).

Status menu

You can use the **Status** menu to change the documentation status of a symbol's documentation. A symbol's documentation can be in one of three possible states: undocumented, partially documented, or (fully) documented.

Note that you alone are responsible for determining what the documentation status of a symbol is. SNIFF+ does not automatically change the status when you have made changes to a symbol's documentation.

Info menu

Please refer to [Info menu — page 20](#).

When you choose the **Documentation Synchronizer...** command in this menu, the Documentation Synchronizer appears. See also [Documentation Synchronizer — page 93](#).

Class menu

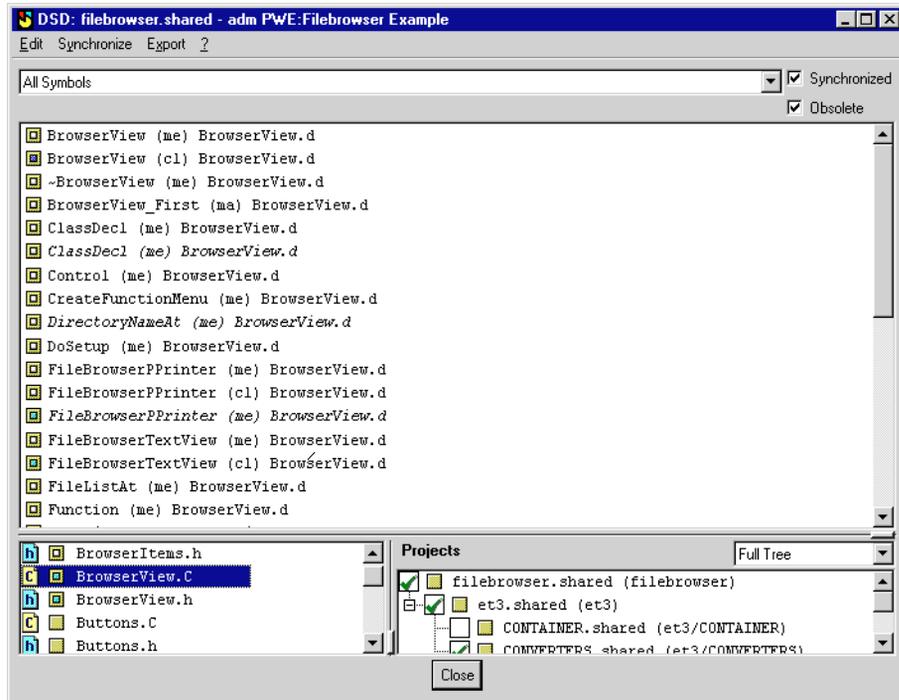
Please refer to [Class menu — page 22](#).

History menu

Please refer to [History menu — page 23](#).

Documentation Synchronizer

The Documentation Synchronizer appears when you choose the **Info > Documentation Synchronizer...**



Quick Reference — Synchronizer

Icons used in the Documentation Synchronizer

Icon	Symbol list	File List	Project Tree
	—	Empty file: This source file doesn't have a corresponding documentation file.	Empty project: None of the source files in the project has a corresponding documentation file.
	Undocumented symbol: The documentation status of the symbol is <i>undocumented</i> .	Undocumented file: All symbols in the corresponding documentation file are undocumented.	Undocumented project: Project contains at least one undocumented source file.

Icon	Symbol list	File List	Project Tree
	Partially documented symbol: The documentation status of the symbol is <i>partially documented</i> .	Partially documented file: At least one symbol in the corresponding documentation file is either partially documented or fully documented.	Partially documented project: Project contains at least one source file that is either partially documented or fully documented.
	Fully documented symbol: The documentation status of the symbol is <i>documented</i> .	Fully documented file: All symbols in the corresponding documentation file are fully documented.	Fully documented project: All source files in the project are fully documented.

Basic Components—Synchronizer

File List

The File List displays the header and implementation files that are part of the projects check-marked in the Project Tree. All header and implementation files are displayed—both those for which documentation files exist and those for which no documentation files exist. Note that there is one documentation file (with extension `.d`) for each header file and its corresponding implementation file.

Each entry in the File List contains the name of the source file and two icon boxes. The first icon box indicates the type of the source file—header or implementation. The second icon box tells you what the documentation status of the source file is. A source file can be in one of four possible documentation states—empty, undocumented, partially documented and fully documented. These states are described on [Icons used in the Documentation Synchronizer — page 93](#).

When you select a file—let's say `BrowserView.C` or `BrowserView.h`—in the File List, the symbols that have documentation frames in `BrowserView.d` are displayed in the Symbol List.

You can select multiple files in the File List by pressing `<SHIFT>` without letting go and then clicking on each file name. The symbols that are documented in the corresponding documentation files are then displayed in the Symbol List.

You can jump directly to a source file by double-clicking its name in the File List.

Synchronizing documentation and source files with each other

After time, your documentation and source files may no longer be in sync with each other. For example, you might have defined a new symbol in a source file, or renamed an existing one. To update documentation and source files with each other, select one or more source files in the File List and then choose the **Synchronize Documentation of Selected Files** command in the dialog's **Synchronize** menu. Note that you can also use this command to generate new documentation files.

Symbol List

The Symbol List displays the documented symbols in the source files that are selected in the File List. Documented symbols are those symbols for which documentation frames exist.

Each entry in the Symbol List contains the name of the symbol, followed by its type in quotes and then file in which it is documented. The icon box that precedes the symbol's name tells you what the documentation status of the source file is. A symbol can be in one of three possible documentation states—undocumented, partially documented and fully documented. See also [Status menu — page 92](#).

You can jump directly to the documentation frame of a symbol by double-clicking its name in the Symbol List.

Filters—Synchronizer

Project Tree

The Project Tree in the Documentation Synchronization dialog is very similar to those in the other SNIFF+ tools.

Each node of the Project Tree contains the name of the project and two icon boxes. The first icon box should already be familiar to you—it's the box that you have to click on to checkmark the corresponding project. When you checkmark a project, its source files—both implementation and header—are displayed in the File List.

The second icon box tells you what the documentation status of the project is. A project can be in one of four possible documentation states—empty, undocumented, partially documented and fully documented. These states are described on [Icons used in the Documentation Synchronizer — page 93](#).

Symbols drop-down

You can use the entries in this menu to filter the list of symbols that are displayed in the Symbol List according to documentation status.

Menus — Synchronizer

Edit menu

You can modify the documentation files that are listed in the File List directly by means of the entries in the **Edit** menu.

You can also issue these anywhere in the Documentation View with the right-click **Context menu**.

Note

If you issue **Edit** commands before saving modifications to the documentation file(s) of the selected symbol(s), SNIFF+ will first revert the file(s) to their unmodified state before executing the command.

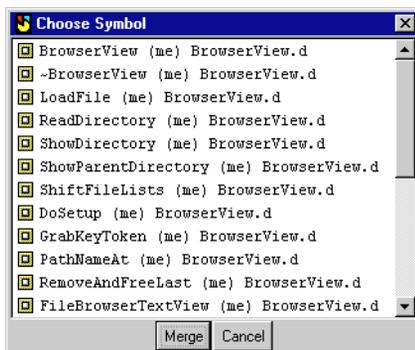
Synchronize menu

The following entries are available in the **Synchronize menu**:

Matching symbols for Obsolete symbol

This menu entry is highlighted when you select an obsolete symbol in the Documentation View. You can use this command to paste the documentation body of an obsolete symbol over the documentation body of another symbol.

When you choose this command, the Choose Symbol dialog appears. This dialog contains a list of undocumented symbols from the same documentation file in which the obsolete symbol is documented. Note that only symbols of the same type as the obsolete symbol are displayed.



To copy the documentation of the obsolete symbol (that is selected in the Documentation View), select one of the matching symbols in the match list and press the **Merge** button.

The documentation body of the obsolete symbol will then be pasted over the documentation body of the matching symbol.

Synchronize Documentation of Selected Files

You can use this command to update the documentation of the source files that are selected from the File List.

When you select one or more source files from the File List and then execute this command, SNIFF+

- checks to see if all the symbols in the source file(s) are documented in the corresponding documentation file(s). Documentation frames for symbols that aren't yet documented will be created. If no documentation file exists for a source file, SNIFF+ will create a new documentation file with empty (undocumented) documentation frames for all the symbols in the source file.
- SNIFF+ then checks to see whether the symbols documented in the documentation files also exist in the Symbol table. Those symbols that do not exist in the Symbol table will be marked as obsolete.

Note that there is one documentation file for both types of source files — header and implementation.

Synchronize Documentation of Checkmarked Projects

You can use this command to update the documentation of the projects that are selected from the Project Tree.

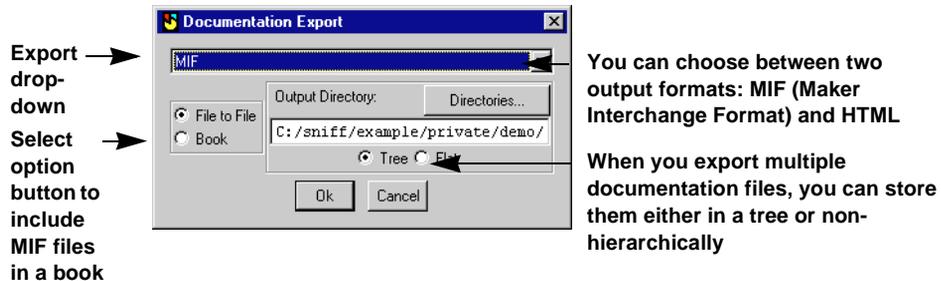
When you checkmark one or more projects in the Project Tree and then execute this command, SNIFF+

- checks to see if all the symbols in the source files of the checkmarked projects are documented in corresponding documentation files. Documentation frames for symbols that aren't yet documented will be created. If no documentation file exists for a source file, SNIFF+ will create a new documentation file with empty (undocumented) documentation frames for all the symbols in the source file.
- SNIFF+ then checks to see whether the symbols documented in the documentation files also exist in the Symbol table. Those symbols that do not exist in the Symbol table will be marked as obsolete.

Export menu

You can use the command in the **Export** menu to export the documentation of whole files or whole projects, in either file or book format.

When you choose either one of these two commands, the Documentation Export dialog appears.

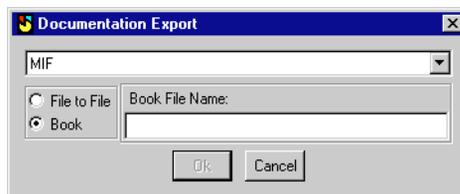


You can export your documentation files in either FrameMaker™ MIF (Maker Interchange Format) or HTML format. SNIFF+ uses the names of the files that you chose from the File List of the Documentation Synchronization dialog for the names of the corresponding export files (MIF files have the extension `.mif`, and HTML files have the extension `.htm`). By default, the export files are stored in the project directory.

Exporting your documentation files as MIF files

When you export your documentation files as MIF files, you can create a book file that contains the MIF files, or you can leave the MIF files as is.

- To create a book file, switch on the **Book** option button in the dialog. The contents of the Documentation Export dialog change.



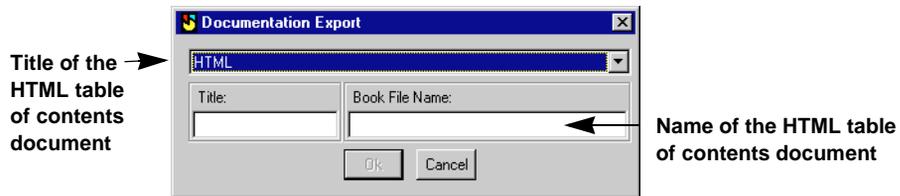
- Enter the name of the book file and the extension `.book` in the **Book File Name** field. By default, the book file is stored in the project directory.
- Press **OK** to create the book file.

SNIFF+ will then create the individual MIF files and the book file.

You can then browse and edit the MIF files with any version of FrameMaker™.

Exporting your documentation in HTML format

When you choose HTML from the **Export** drop-down, the contents of the Documentation Export dialog change:



- Enter the title of the HTML table of contents document in the **Title** field. You will then see this title at the top of the HTML table of contents document in your HTML browser.
- Enter the name of the HTML book file in the **Book File Name** field.

The table of contents document consists of the names of the symbols in the documentation files that you selected in the File List of the Documentation Synchronization dialog. You can jump to a symbol's documentation frame by clicking on its name in the table of contents document.

Note

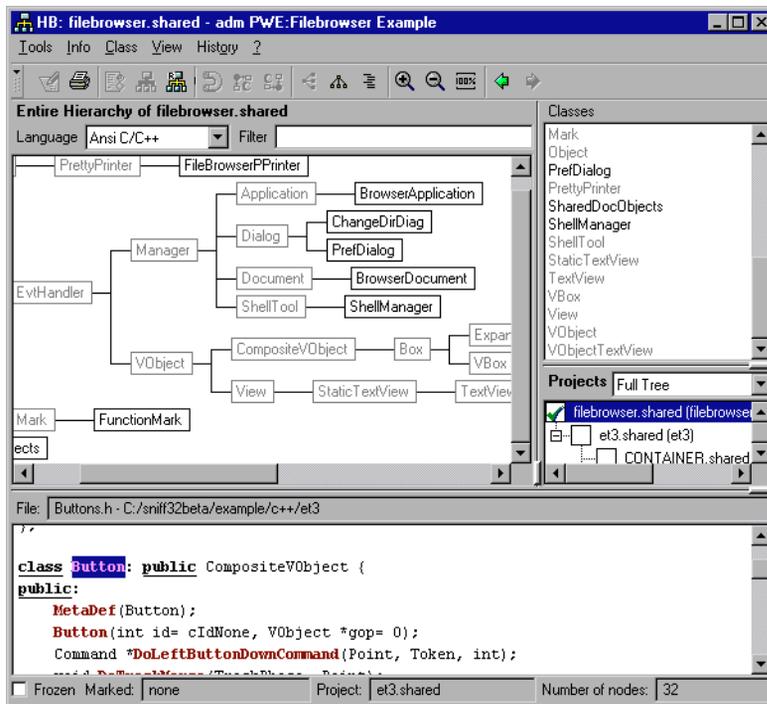
An HTML file is created for each of the files that you selected in the File List of the Documentation Synchronization dialog. The name of the HTML file is the same as the name of its corresponding documentation file, and its extension is `.htm`.

Hierarchy Browser

Introduction

The Hierarchy Browser shows the inheritance relationships of classes. It either displays the entire class tree or only the superclasses and subclasses of a class. Like other SNIFF+ browsers, the Hierarchy Browser allows you to filter the tree according to project boundaries and other filter settings.

You can invoke the Hierarchy Browser from the **Class** menu, or by choosing **Tools > Hierarchy Browser**.



Quick Reference

Typeface in Hierarchy view and Symbol List

The typeface indicates the following about a class:

Typeface and frame	Class is
Normal typeface and rectangular frame	normal
<u>Blue italics</u> and rectangular frame	abstract
Italics and rounded corners	an interface (Java)
Green Typeface	final (Java)
Bold Typeface	documented
Grayed	not in checkmarked projects, only the classes needed to draw a minimal tree are displayed

- Selected classes are connected to their immediate base and derived classes by red lines.

Mouse clicks in Hierarchy view and Symbol List

- Clicking on a class in the Symbol List selects the class in the Hierarchy view (and vice-versa).
- Deep-click (<CTRL>double-click) on a symbol in the Hierarchy view, jumps to a symbol's reference in the Source Editor.

Keyboard Navigation

As in lists, the current selection can be positioned to an element (in this case a class) by typing the name while the mouse cursor is over the Hierarchy view or the Symbol List.

Basic components

Hierarchy view

For a description of the abbreviations and typeface used in the Hierarchy view, please see [Quick Reference — page 102](#). The Hierarchy view shows inheritance relationships from left to right. Multiple inheritance is displayed by multiple connecting lines between class nodes. The view can be restricted to just relatives of a class. Classes of disabled projects are grayed-out if they are needed to draw a minimal tree (see “Project Tree” below). C++ templates are shown with their formal parameter list.

Code view

References to the symbol selected in the Hierarchy view are displayed in the Code view. The Code view is a read-only view.

Symbol List

Classes are displayed in the Symbol List in alphabetical order.

Filters

Project Tree

The Project Tree can be used as a filter to show only classes belonging to a certain project. Classes of projects not checkmarked in the Project Tree are only shown grayed-out if they are needed to form a minimal Tree for the classes of the checkmarked projects. The illustration above shows the Hierarchy Browser with only the root project checkmarked.

Language drop-down

In the **Language** drop-down, you can choose the language whose symbols you want to browse. Only the languages used in your project structure are listed.

Inheritance drop-down (Java)

This drop-down is only visible for Java projects.

Inheritance	Shows
complete	class and interface inheritance
class	classes only
interface	classes inheriting from interfaces

Filter field

Please refer to [Regular expression filters — page 11](#).

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Marked

When the Hierarchy Browser is started from another tool using the menu command **Class > Mark Definers of *method*** or **Class > Mark Relative Defining *method***, the classes implementing *method* are marked by being displayed in bold typeface. This information field tells you which method is being queried.

Project

Indicates the project to which the selected symbol belongs.

Number of Nodes

Indicates the number of nodes in the Hierarchy view.

Menus

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Class menu Command	Description
Edit	Loads the implementation of the corresponding method of the marked class into an Source Editor.
Reset Markings	Resets the current markings.

For a description of the other commands, please refer to [Class menu — page 22](#).

View menu

The first three commands in the View menu are described under [View menu — page 23](#). Other commands:

View menu command	Description
Layout >	<p>The first three commands in the Layout submenu allow you to select a general layout for the Hierarchy Graph.</p> <p>The three commands below the separator allow you to select the kind of connecting lines used for each general layout selected above the separator. This in turn affects the overall layout.</p>
Zoom commands	Allow you to zoom in and out in the Hierarchy view.

History menu

Please refer to [History menu — page 23](#).

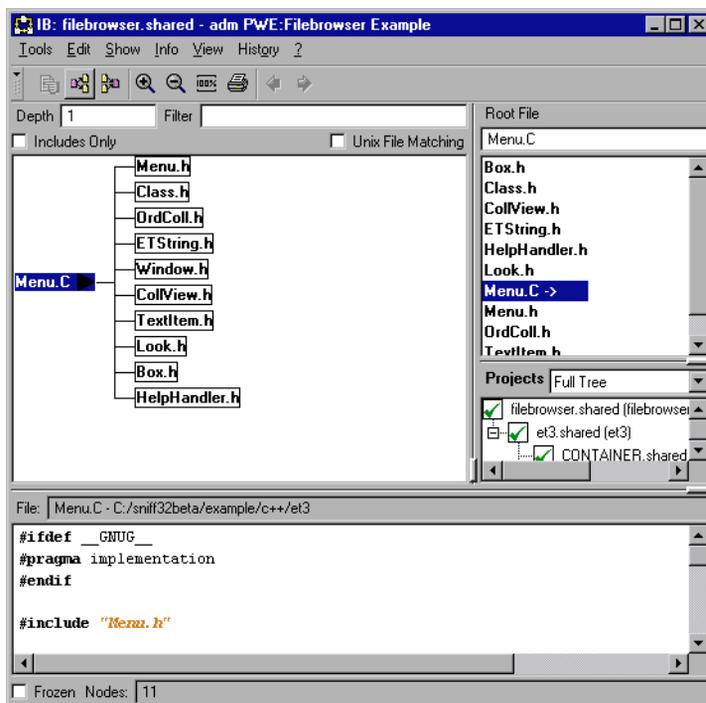
Include Browser

Introduction

The Include Browser graphically displays include references between files in your projects. It is very similar in both layout and functionality to the Cross Referencer tool.

You can use the Include Browser to see which files are included by a particular file and vice-versa, as well as to make sure that there are no redundant includes. As an example of a redundant include, suppose `A.h` includes `B.h` and `B.h` includes `C.h`. A third include, namely `A.h` includes `C.h`, would be redundant, and you will be able to notice such redundancies in the Graph View of the Include Browser.

Furthermore, you can also use the Include Browser to make sure that the includes in your project files are “clean”, meaning that the files are neither included by nor include files outside of your projects.



Quick Reference

Typeface in Graph view

The typeface indicates the following about a class:

Typeface	Description
bold	File is part of a SNIFF+ project.
italics	File is already in the parent tree of the graph, danger of infinite loop.
grayed-out	Symbol information of the file is no longer up-to-date (e.g., file has been modified), the next query updates the Graph view.
normal	File is not part of a SNIFF+ project.

Mouse clicks in the Graph View

- double-click** A double-click on a symbol node in the Graph View loads the file into a Source Editor.
- <SHIFT>double-click** A <SHIFT>double-click on a symbol node loads the location of the include statement of the symbol into a Source Editor (depending on the include type).

Basic components

Graph View

For a description of the abbreviations and typeface used in the Graph View, please see [Quick Reference — page 108](#). The Graph View shows the results of the include queries. It always has one root file and extends from left to right. The view shows either files that are included by the root file or files that include the root file (indicated by the direction of the arrows in the nodes). You can limit the amount of information that is shown in the graph by means of filter settings. Each node represents a file.

Note

For Java projects, only those files that are actually used are shown. Files in imported packages that are not used in the project are not shown.

File List

The File List is an alphabetical listing of the nodes in the graph. Multiple nodes in the graph are only shown once in the index. When a node is selected in the graph, the corresponding entry in the list is also selected and vice-versa. A right-arrow (>) after a file means that the included files of this file are displayed in the graph; and a left-arrow (<) after a file means that the files that include this file are displayed in the graph.

Filters

Root File field

You can use any file name as the root file in the graph by entering the file's name in the **Root File** field. By pressing <Return>, the existing graph is reset and the entered file name becomes the root file in the graph. You can then select a query in the **Reference** menu.

Project Tree

You can use the Project Tree to constrain the search in an include query. A check mark next to the name of a project means that the contents of the project will also be searched during a query. To update the graph so that it corresponds to the new settings in the Project Tree, select a query in the **Reference** menu.

Check boxes

Includes Only

Filters the Graph view to display only .h files.

Unix File Matching

Toggles case sensitivity of file names.

Status Line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Nodes

Indicates the number of nodes in the Graph view.

Menus

Edit menu

Edit menu command	Description
Replace Include Statements	Starts the Retriever in Replace Only mode where you can globally change the references.

Show menu

Show menu command	Description
Include Statement	Opens a Source Editor and is positioned at the include statement.
<i>FileName</i>	Opens a Source Editor and loads the currently selected file into it.

Info menu

Info menu command	Description
Includes	Starts a query to display the files that the selected file includes. The current filter setting applies.
Included-By	Starts a query to display the files that include the selected file. The current filter setting applies.

View menu

The first three commands in the View menu are described under [View menu — page 23](#). You can use the following **View** menu entries to specify the layout of the Graph View:

View menu Command	Description
Show/Hide Project Name	Toggles whether project names are displayed in the Graph View
Layout >	The first three commands in the Layout submenu allow you to select a general layout for the Hierarchy Graph. The three commands below the separator allow you to select the kind of connecting lines used for each general layout selected above the separator. This in turn affects the overall layout.
Show/Hide Project Name	Shows/hides project names.
Start from <i>FileName</i>	Makes the currently selected file the root file of the graph. All other nodes are deleted from the graph.
Hide Subnodes of <i>FileName</i>	Hides all subnodes of the selected file.
Show Restricted Tree of <i>FileName</i>	Shrinks the graph in such a way that only parent and subnodes are displayed. All other nodes are deleted from the graph.
Zoom commands	Allow you to zoom in and out.

History menu

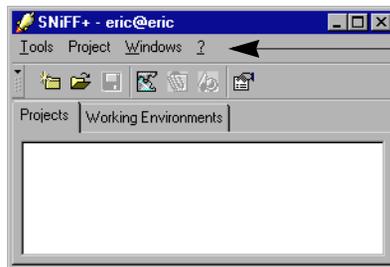
The **History** menu contains a list of previous root files, as well as the current root file. Selecting a file from the list makes it the new root file of the graph.

Launch Pad

Introduction

The Launch Pad is the main SNIFF+ application window and serves to manage projects and open tools on your desktop. You can also open projects in your Private Working Environments directly from the Working Environments tab.

The **Help (?)** menu has supplementary commands that are available only in the Launch Pad, see [Help \(?\) menu — page 117](#).



Use the **Help (?)** menu in the Launch Pad to open online documentation, and also for information relating to your SNIFF+ installation

Quick Reference

Icons in the Projects Tab

The icons in the **Projects** tab indicate the following about the status of open projects:

Icon	Project Status
	The project is writable; the Project Description File (PDF) is writable and the files of the project may be modified if they are writable
	The project is frozen; neither the Project Description File (PDF), nor the files of the project may be modified even if they are writable. Typically such projects are libraries.
	The Project Description File (PDF) is read-only, but files of the project may be modified if they are writable
	The project (attributes or structure) has been modified, but not yet saved to the Project Description File (PDF).

Typeface in the Projects tab

Typeface	Project is
Normal	visible
<i>Italics</i>	hidden

Mouse clicks in the Projects tab

- **Double-click** on a listed project to hide/show the project and all tools associated with the project.
- **<CTRL>click** on a listed project to show the selected project and hide all other open projects.

Mouse clicks in the Working Environments tab

- **Double-click** on an entry to invoke the Open Project dialog. The dialog lets you select from a list of projects that can be accessed in the selected Private Working Environment (PWE). See also [Open Project dialog — page 26](#).

Basic components

Projects tab

For a description of the icons and typeface used in the Projects tab, please see [Quick Reference — page 114](#).

Open projects are listed in the Projects tab together with the working environment they were opened in.

Double-click on a listed project to hide/show the project and all tools associated with the project.

<CTRL>-click on a listed project to show the selected project and hide all other open projects.

Working Environments tab

Your Private Working Environments (PWEs) and PWEs that belong to nobody (that is, the default user adm) are listed in the Working Environments tab. See also [Working Environments — page 239](#)

Double-click on an entry to invoke the Open Project dialog. The dialog lets you select from a list of projects that can be opened in the selected Private Working Environment (PWE). See also [Open Project dialog — page 26](#)

Menus

Project menu

The **Project** menu offers commands for handling projects, as well as a list of the five most recently opened projects.

Project menu command	Description
New Project... >	Opens a submenu with 3 options for creating new projects. For more information please refer to User's Guide — Project Setup Overview — page 53 .
> with Defaults...	Opens the Directory dialog. In the Directory dialog, you select the directory where the source files of the new project are located. Then, an Attributes of a New Project dialog appears. You set the new project's attributes in this dialog, see also New Project Options — page 165 . The defaults for new projects are set in the Preferences, see also New Project Setup view — page 146 .

Project menu command	Description
> with Template...	Allows you to set the attributes of a new project from a template. A dialog opens and you can select a Project Template File (extension <code>.ptmpl</code>) to use as a template. Select a file to open the Project Attributes dialog. You can edit the opened template, have the new project created, and save the edited template under a new name. See also User's Guide — Working with new project templates — page 57 .
> with Wizard...	Starts the Project Setup Wizard, which guides you through project setup. See also User's Guide — SNIFF+ Project Setup Wizard — page 53 .
Open Project...	Opens a dialog where you can navigate to the Project Description File (PDF) of the project that you want to open. After a PDF has been specified, the project is loaded into SNIFF+ and the environment (all window positions, sizes and contents) is restored to the same status that the project had when it was last closed.
Save Project project	Saves the Project Description File (PDF). If Make Backups in the Tools view of the Preferences dialog is enabled, a backup file with the name <i>project%</i> is created. This command is only enabled if the PDF has changed.
Save Project project As...	Opens a File dialog for saving the current project to a new Project Description File.
Reload Project >	Reloads the selected project from disk. The commands in the submenu can be chosen if the Project Description File (PDF) has changed while the project was open, or when you want to discard all unsaved modifications to a project. If the structure of the selected project has changed, the corresponding files/sub-projects are (un)loaded. The project attributes are also updated.
> In Current Working Environment	Reloads the open project in the working environment you are currently working in.
> In Other Working Environment...	Opens a dialog where you can select the working environment to reload the project in.

Project menu command	Description
Delete Project project	Opens an Alert dialog for the selected project and each of its subprojects. Select the Repeat check box on the dialog and confirm. SNIFF+ deletes the Project Description File of the project (thereby deleting the project) and all other files that were generated by SNIFF+. Note that all files not generated by SNIFF+ (e.g., source code files) remain untouched.
Close Project project	Closes the selected <i>project</i> and all associated tools. If the structure or attributes of the project have been modified, a dialog appears asking whether the project file should be saved. Projects are reopened with the same tools and window settings that they were closed with.
Hide/Show Project project	Hides/shows the selected project and all tools associated with it. When the project is hidden, the project name is displayed in italics.
List of Recent Projects	At the bottom of the Project menu there is a list of up to five recently opened projects together with the working environments they were opened in. Note that projects are only listed once. If you, e.g., open the same project in the same environment twice, once with symbols and once without, the latest version only is stored in the list.

Windows menu

A list of all open tool windows, grouped by project, is available in the **Windows** menu. Select an item to activate the tool window. Note that tool windows of hidden projects are not displayed in this list. Project-independent tools are also grouped.

Help (?) menu

Apart from the **Help** commands common to all SNIFF+ tools (**Tool Help**, **Context Help** and **Quick Ref**), the Launch Pad **Help** menu provides the following commands:

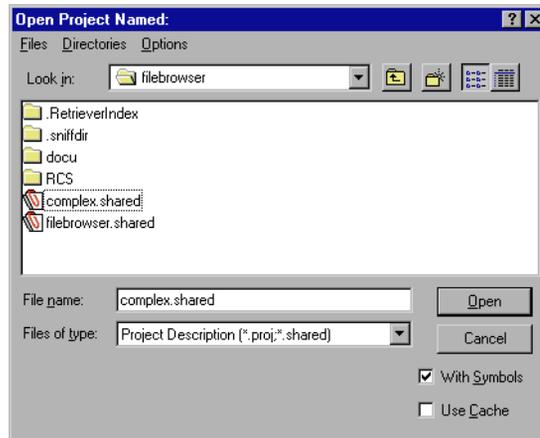
Help menu command	Description
Online Documentation	Opens User's Guide and Reference manuals.
Tutorials >	Offers a submenu with four tutorials to choose from. Each tutorial uses an example project in a different programming language.

Help menu command	Description
> CPP	This tutorial uses C++ code examples.
> C	This tutorial uses a C code example.
> Java	This tutorial uses a Java code example. A SNIFF+ for Java Technical Reference is also included.
> Fortran	This tutorial uses a Fortran code example.
Launch Pad Help	Provides help for the Launch Pad.
Context Help	Provides context help. Choose this command or, point to a tool element with the mouse and press <F1> for context-specific help. If no context help is available, <F1> opens the tool help.
Quick Ref	Opens documentation at a Quick Reference to icons, typeface and mouse-clicks used in the Launch Pad.
Welcome To SNIFF+...	Opens a dialog allowing you to quickly check on installation and license features. The dialog also guides you to the SNIFF+ language-specific tutorials.
Licenses...	Opens the Licenses dialog to display information about the current license status of your SNIFF+ installation (see Licenses dialog — page 45).
Feedback...	Opens a form for sending TakeFive's Support service feedback about your SNIFF+ installation.
About SNIFF+...	Opens the About SNIFF+ dialog to display version and copyright information.

Open Project dialogs

Two different Open Project dialogs can be invoked from the Launch Pad.

1. One dialog is invoked via a double-click on a Working Environment in the Working Environments tab. This dialog is described under [Open Project dialog — page 26](#).
2. The second dialog is invoked from the Launch Pad's **Project > Open Project...** menu, and is described below.



Menus

Menu	Description
Files	Lists a history of recently opened files.
Directories	Lists a history of recently opened directories.
Options	Allows you to configure the Files and Directories menus.

Opening Mode check boxes

Selected Opening Mode check boxes determine what data is used in opening a project.

With Symbols Opens projects with symbol information. For browsing and day-to-day development work, enable this check-box. If you do not need symbol information, e.g. to open large projects for making structural changes, clear this check box. Default: selected.

Use Cache When projects are closed, all necessary information for re-opening them is cached in a single file to speed up project opening. If **Use Cache** is selected, only this file is read. If not, all and the original Project Description Files (PDFs) are read, and all source files are checked.

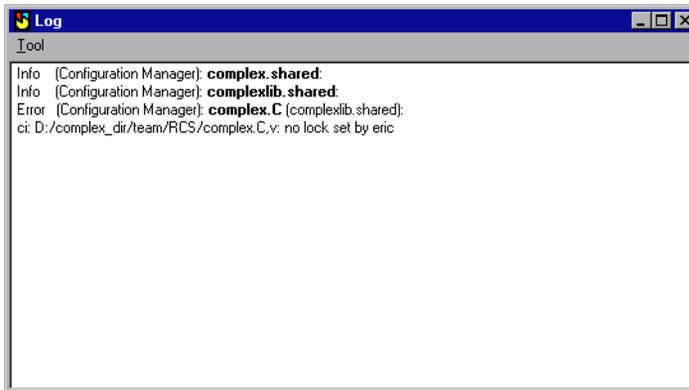
Caution: The cached information can be **incorrect** if (1.) changes are made to projects and files between SNIFF+ sessions, that is, outside of SNIFF+, or (2.) if the preceding SNIFF+ session was terminated unexpectedly. Default: not selected.

Introduction

The Log window displays SNIFF+ error and control messages. Messages are sent to the Log window and not to the terminal where you started `sniff`. The information displayed in the Log window can be appended to a log file, and the window can be set (not) to pop up automatically on output (defined in the Preferences, [Tools view — page 130](#)).

Log window

To open the Log window, choose the **Tools > Log** menu command. If set (see Preferences, [Tools view — page 130](#)), the window pops up automatically when output is written to it.



Tool menu

Tool menu command	Description
Close	Closes the Log window.
Clear	Clears the buffer of the Log window.
Find...	Opens the Find/Change dialog.
Find Again	Repeats the last find operation.

Tool menu command	Description
Show Error	Allows direct navigation to the source of parsing errors reported in the Log. This only works if the <code>-e</code> preprocessor directive is set.

Preferences

Introduction

This chapter focuses on the Preferences dialog. For advanced customizing options, please refer to [Advanced Customization — page 279](#).

SNiFF+ supports customization at three different levels:

- **Site level**—Each installation of SNiFF+ can have its own settings. These settings are used by all users that access this installation.
- **User level**—Each user can have private settings that override or merge with the site level settings.
- **Project**—Each project has its own project attributes that are stored in the project description file (PDF) and can be edited using the Project Attributes dialog. [Project Attributes — page 163](#).

Preferences

The preference attributes in your Preferences allow you to customize certain aspects and of SNiFF+ and its tools:

- appearance
- tool-specific settings
- default project attributes settings for new projects
- version and configuration management interface
- file types
- platform-specific settings
- other preferences

There are two levels of preferences in your Preferences:

- user preferences (stored in `$HOME/.sniffrc/UserPrefs.sniff` on Unix and in `%SNIFF_DIR%\Profiles\\UserPrefs.sniff` on Windows)
- site preferences (stored in `$SNIFF_DIR/SitePrefs.sniff` on Unix and in `%SNIFF_DIR%/SitePrefs.sniff` on Windows)

The site level preference file with default values is delivered with SNiFF+; an empty user preferences file is created in your home directory when you start SNiFF+ the first time.

Whenever SNIFF+ reads a preference attribute, it first searches at the user level, and if the value there is not specified, it takes the value from the site preferences. If you have write permissions to access the `SitePrefs.sniff` file, there are two tabbed pages in your Preferences, a **User** tab and a **Site** tab. To make site level changes, select the **Site** tab.

Caution

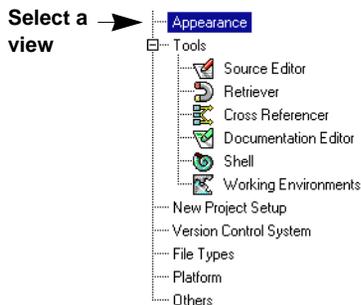
The preferences files are marked-up ASCII files that contain a data schema in front of the actual data. These files should be modified with the Preferences dialog. If you modify them manually, be very careful that you do not corrupt the structure of the files.

Note

If you are working in some directory other than the SNIFF+ installation directory, set the environment variable `SNiFF_RESOURCE_DIR` to this directory before starting SNIFF+. When set, the user specific preferences files will be added to this directory.

Preferences dialog

The Preferences dialog allows the modification of both site- and user-level preferences. The dialog consists of thirteen views or groups of preference attributes.



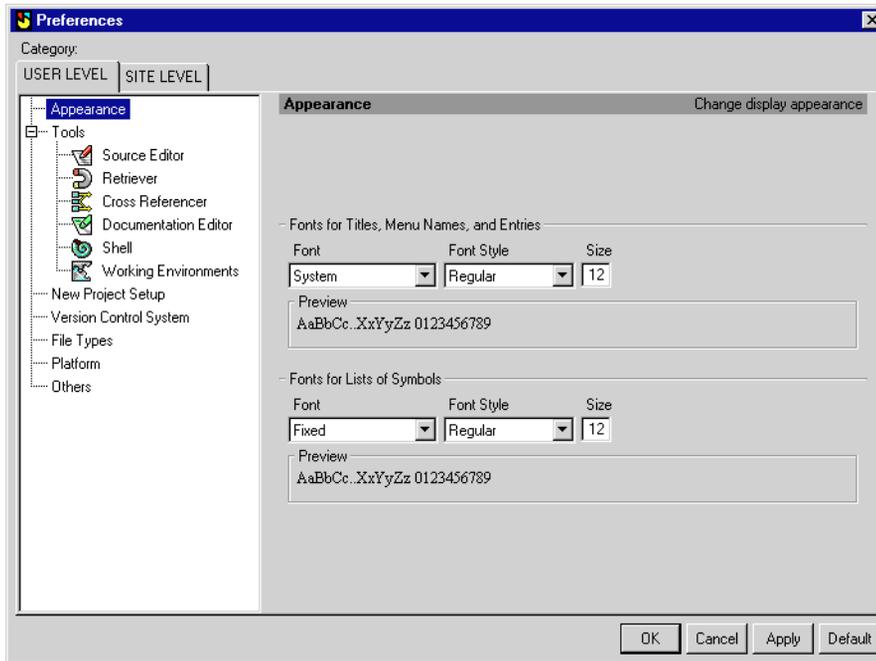
User-level preferences can be manipulated directly and saved. Site level preferences need to be locked before they can be modified to prevent concurrent modification by two users. However, you can break a lock of another user.

The dialog offers the following buttons that are present at the bottom of all views:



- OK** Saves the current modifications to the corresponding preferences file (dependent on the level), updates SNIFF+ with the changes and closes the Preferences tool.
- Cancel** Discards all changes and reverts to the last saved version of the active view.
- Apply** Saves the current modifications to the corresponding preferences file (dependent on the level) and updates SNIFF+ with the changes.
- Default** Resets all values in the current view to those in the next preferences level. For example, you may want to press this button at the user level to reset the complete view and to take all values from the site level.

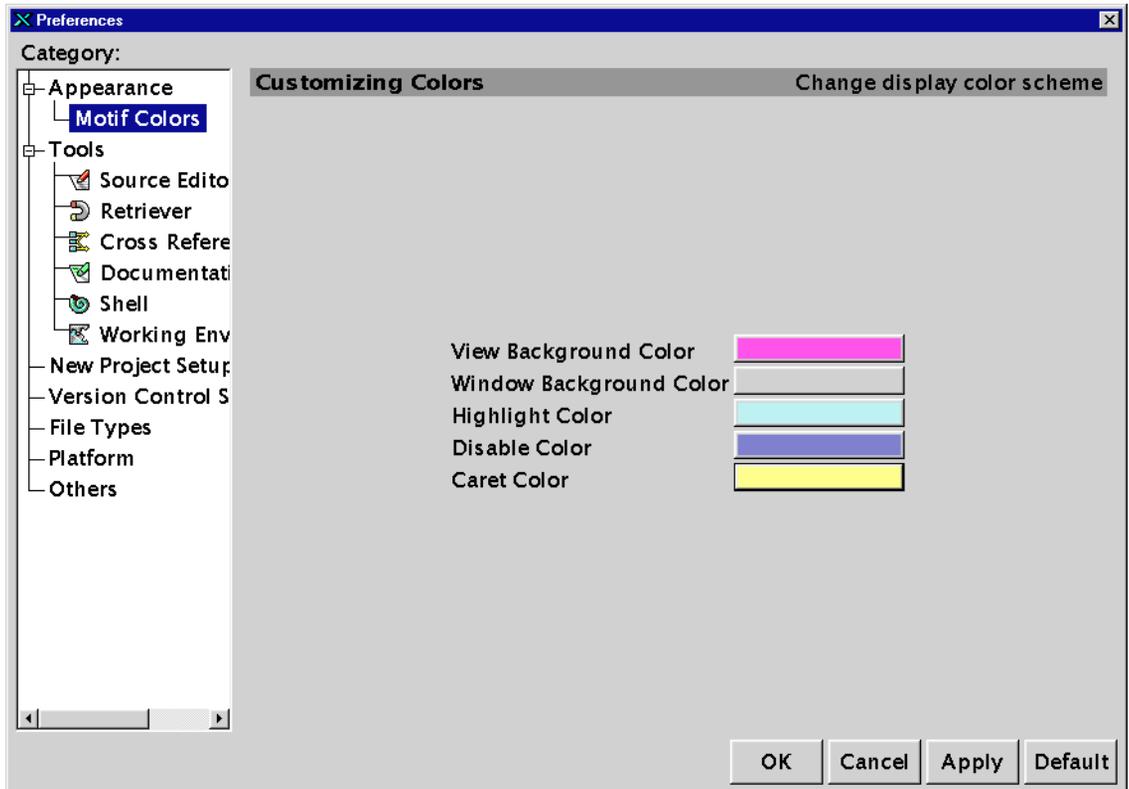
Appearance view



Look

- | | |
|--|---|
| Motif (Unix only) | Indicates whether the windows are drawn in Motif look. |
| Windows (Unix only) | Indicates whether the windows are drawn in Windows look. |
| Font for Titles, Menu names and Entries | Defines the general font that is used for all SNIFF+ texts that are drawn in proportional text. |
| Font for Lists of Symbols | Specifies the font that is used for all fixed-width texts. Note that Source Editor fonts are specified separately. See also Source Editor view — page 132 . |

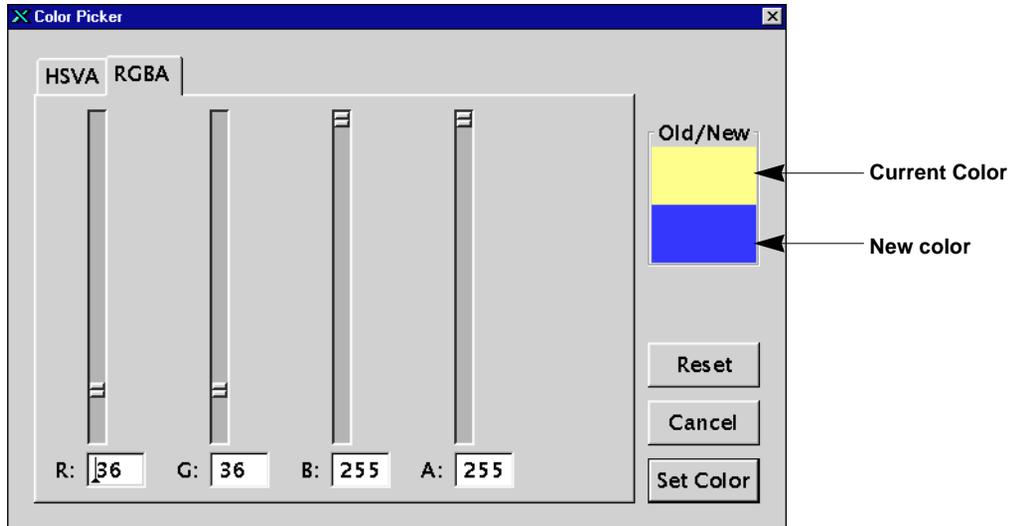
Motif Colors (Unix only)



- | | |
|--------------------------------|--|
| View Background Color | Specifies the background color of all views in SNIFF+. |
| Window Background Color | Defines the background color of all windows in SNIFF+. |
| Highlight Color | Defines the highlight color used for selected text or menus. |
| Disable Color | Specifies the color in which disabled items are drawn. |
| Caret Color | Defines the color of the text caret. |

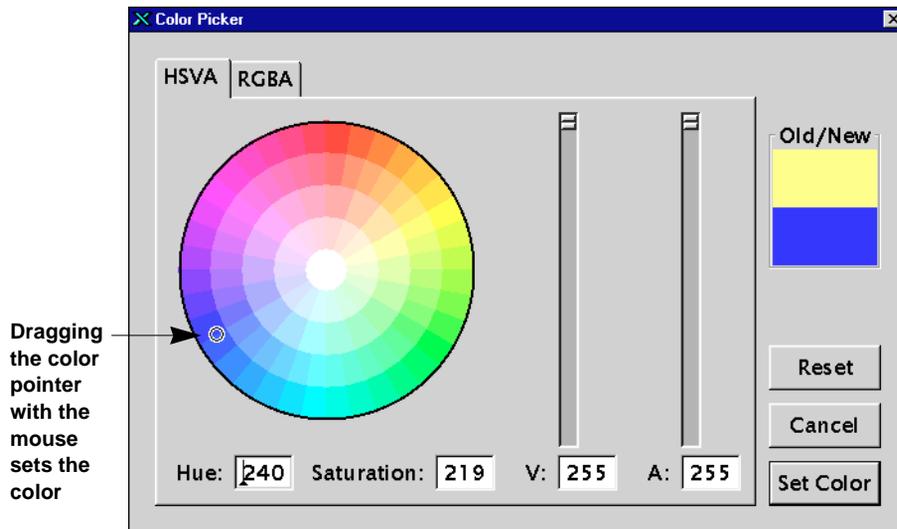
Color Picker dialog

Colors in SNIFF+ can be set in the Color Picker dialog. This dialog appears when you press the **Color** button in either the Source Editor or Documentation Editor view of the Preferences dialog.



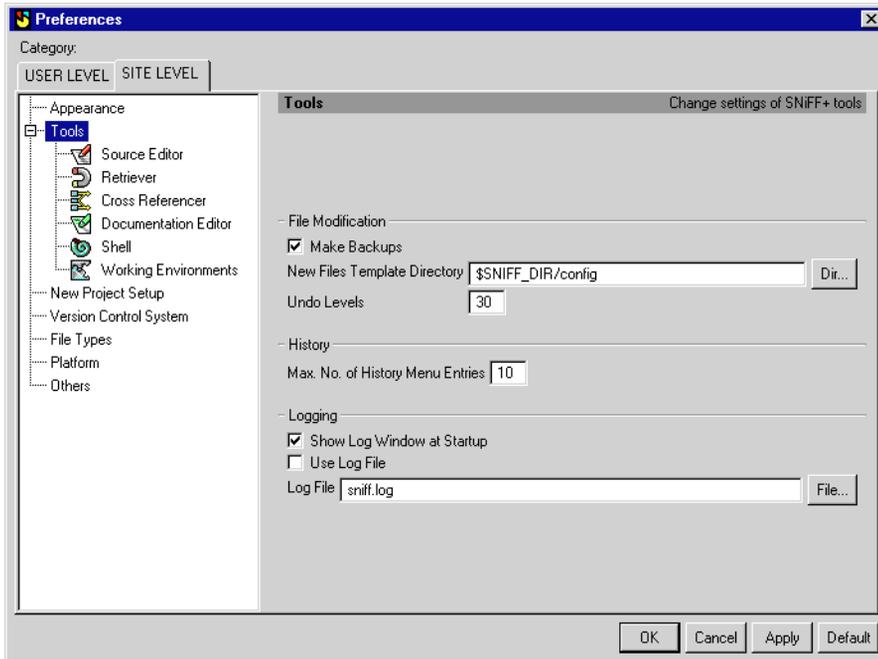
- | | |
|------------------|--|
| Gauges | Define the RGBA values of the color:
R: red
G: green
B: blue
A: alpha value (should always be 255) |
| HSVA tab | Switches to the HSVA view. For details, please refer to page 129 . |
| Reset | Resets the color to the value it had when the dialog was opened. |
| Cancel | Leaves the current color values unchanged, discards all modifications and closes the dialog. |
| Set Color | Sets the current color to the values of the Color Picker and closes the dialog. |

The following illustration shows the HSVA view of the Color Picker on a color wheel:



- Hue** Defines the hue of the color (the value is modified when the color pointer is dragged with the mouse).
- Saturation** Defines the saturation of the color (the value is modified when the color pointer is dragged with the mouse).
- Gauges** Define the values of the color wheel:
V: Sets the Value of the color wheel.
A: Sets the Alpha value of the color wheel.
- RGBA tab** Switches to the RGBA view. See also [Color Picker dialog — page 128](#).

Tools view



File Modification

Make Backups

Specifies whether SNIFF+ creates a backup file when a file is saved. The setting is used by the Source Editor, the Diff/Merge tool and the Documentation Editor, as well as by the Launch Pad and the Project Editor for project description files. The name of the backup file is *file-name%*.

Default: selected (make backup file)

New Files Template Directory

Indicates the directory in which template files are stored. Template files are used for new files that are created in SNIFF+. The template that is used is determined by the extension of the new file. Templates must be called *template.extension*, whereby *extension* is one of the allowed file type extensions.

Template files are stored in your `$SNIFF_DIR/config` directory.

Undo Levels

Defines how many commands can be undone in tools like the Source Editor, the Diff/Merge tool and the Documentation Editor.

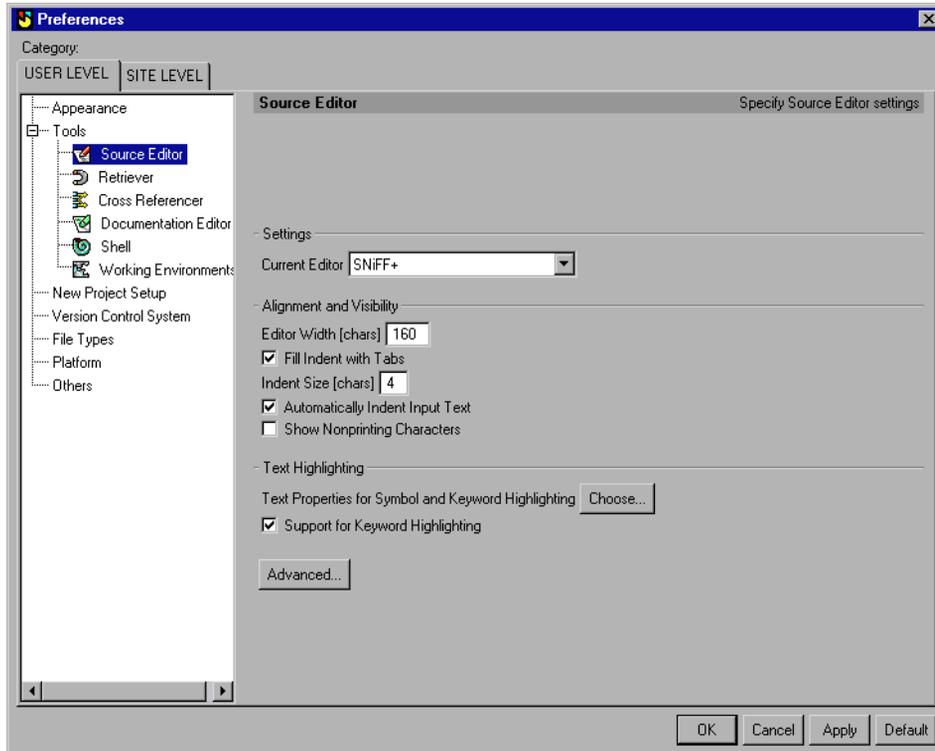
History

Max. No. of History Menu Entries	Defines the size of the History menu, i.e., how many entries are remembered by SNIFF+. Default: 10
---	---

Logging

Open Log Window on Output	Indicates whether the Log window should automatically appear on your screen when output is written to. Default: selected
Use Log File	Indicates whether a log file should be used to store logging information. Default: not selected
Log File	Specifies the location of the SNIFF+ log file. Default: On Unix: \$HOME/.sniffrc/sniff.log On Windows: %SNIFF_DIR%\Profiles\ <username>\sniff.log</username>

Source Editor view



Settings

Current Editor Specifies which editor is used, an external editor or the internal Source Editor.

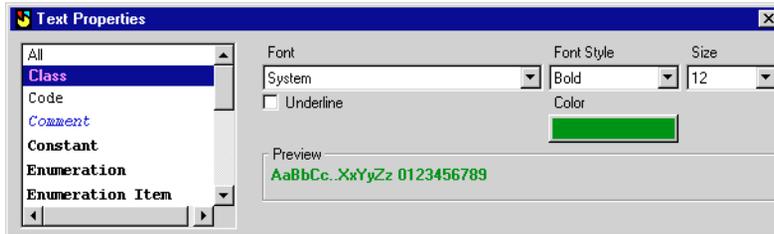
The following Reference settings apply for the SNIFF+ Source Editor only.

Alignment

Editor Width [chars]	Defines the line length (horizontal size) of the Source Editor view in characters. Lines that are longer than the Editor Width are wrapped.
Fill Indent with Tabs	Determines whether indentations are filled with tab characters. If selected, indentations are filled with the largest possible integer value of tab characters from the quotient Indent Size/Tab Width (at least 1).
Indent Size [chars]	Specifies the spacing (in blank characters) when the <Tab> key is pressed or when lines are nested in the Source Editor. If Indent Size is set to 0, the Tab Size attribute is used. If Fill Indent with Tabs is selected, the indentations are filled with tab characters. For details about the Tab Size attribute, please refer to General Advanced — page 167 .
Automatically Indent Input Text	Automatically indents the text that is entered. Please note that changes are only effective once the current project is closed then reopened.
Show Nonprinting Characters	Shows all whitespace characters.

Text Highlighting

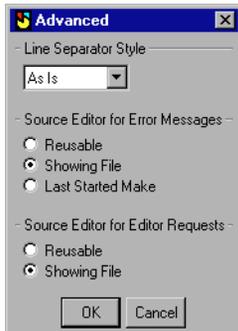
Text Properties for Symbol and Keyword Highlighting Defines the fonts and colors of the Text view in the Source Editor. Press **Choose...** to open the Text Properties dialog, in which you can choose the font and colors of various symbols and keywords as well as specify whether they should be underlined.



Support for Keyword Highlighting

Highlights language specific keywords in source code.

Advanced



Line Separator Style Specifies whether the line separator style should be Unix, Dos or Mac.

Source Editor for Error Messages During builds, the Shell displays line numbers in which your compiler finds errors during compilation. Here, you can specify which Source Editor should be use for positioning to location of the errors.

Reusable Loads the location of the error into a reusable Source Editor. If no reusable Source Editor is available, a new one will be opened.

Showing File Loads the location of the error into the Source Editor that already shows the file, regardless of whether the Source Editor is reusable. If no Source Editor shows the file and no reusable Source Editor is available, a new one is opened.

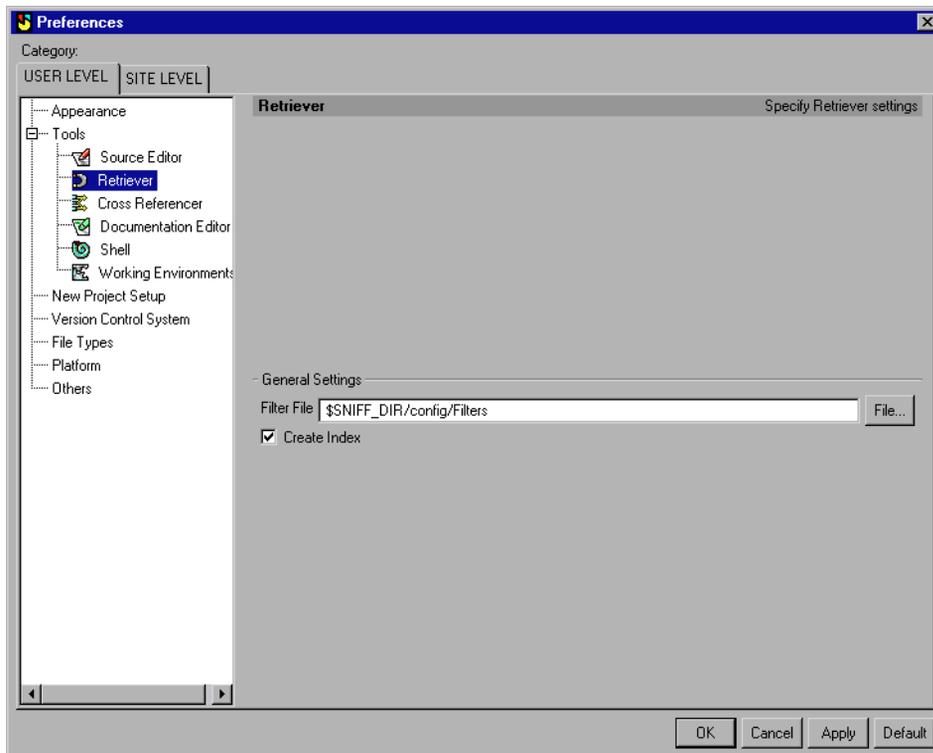
Last Started Make Loads the location of the error into the Source Editor from which Make was last invoked.

Default: Showing File

Source Editor for Editor Requests Specifies which Source Editor should be used for editing requests.

Reusable	Loads requested symbol/file into a reusable Source Editor. If no reusable Source Editor is available, a new one is opened.
Showing File	Loads the symbol/file into the Source Editor that already shows the symbol/file, regardless of whether the Source Editor is reusable or not. If no Source Editor has loaded the file and no reusable Source Editor is available, a new one is opened.
Default:	Showing File

Retriever view



General Settings

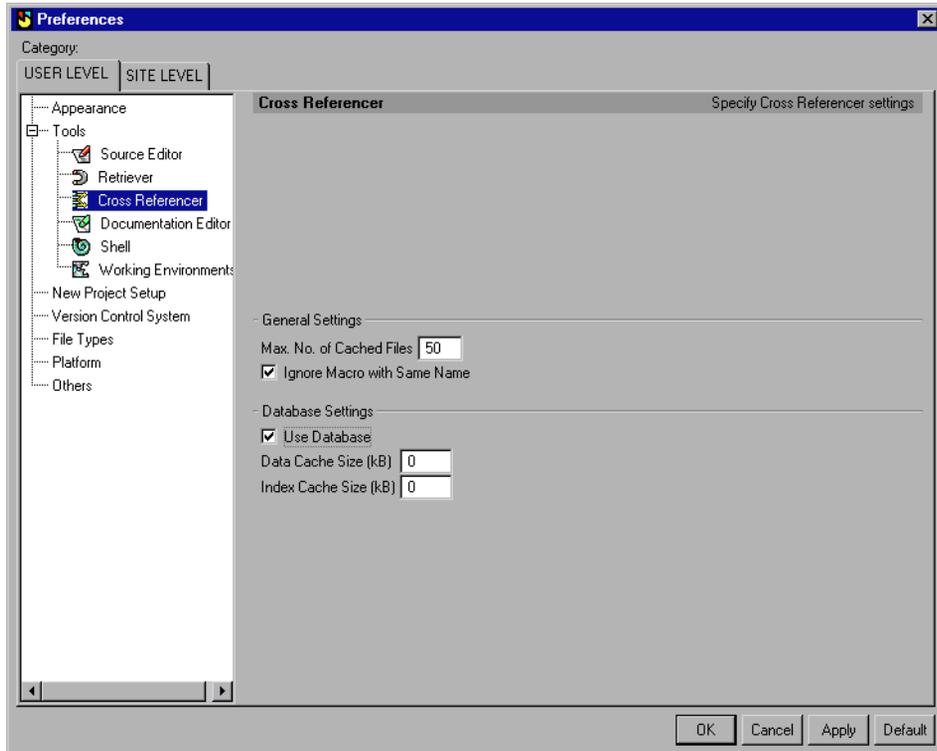
Filter File

Defines the location of the filter file for the Retriever. The file defines filters that are available in Find and Replace Filters dialog of the Retriever. In the file, each filter definition is contained in a struct data type. The name of the filter as it appears in the Retriever is specified by the “Name” element of a filter definition. The regular expression used for implementing the filter is specified by the “Search” element. In the regular expressions, %s expands to the current match for each matched source line currently displayed in the Retriever. %s can be used any number of times in the regular expressions.
Default: \$\$SNIFF_DIR/config/Filters

Create Index

Specifies whether an index should be created. To use this index, select the **Use Index** checkbox in the Retriever and each time a query is triggered the Retriever searches the index.

Cross Referencer view



General Settings

Max. No. of Cached Files

Specifies the maximum number of cross reference files that are kept in memory. A value of zero (0) means no limit (x-ref files are loaded on demand and memory is not limited). A value of 1 means that only one x-ref file is kept in memory. A larger value means that more x-ref files will be kept in memory, thereby reducing the query times of referred-by queries at the cost of increasing memory usage. Values in this field apply only if you do **not** use a cross reference database.

Ignore Macro with same name Determines how the Cross Referencer treats references to a symbol that has the same name as a macro. If disabled, the Cross Referencer treats the referenced symbol as the identically-named macro. If enabled, the Cross Referencer treats the referenced symbol as a symbol, thereby ignoring the fact that there is a macro with the same name.
Default: selected

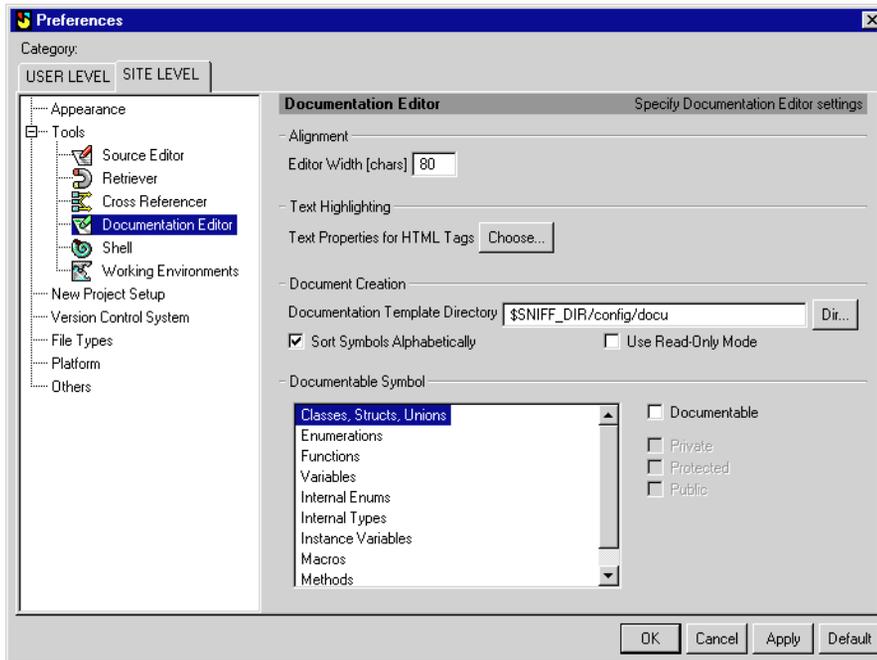
Database Settings

Use database If you select this option, a cross reference database is generated at the root of each Working Environment where Projects are opened. Once the database has been generated, cross reference queries directly access it. Consequently, subsequent "referred-by" queries are constantly faster and require negligible additional memory, regardless of Project size and query complexity. If this option is not selected, a RAM-based cross reference system is used. Please refer to the *User's Guide* for more information about the SNIFF+ cross reference subsystems.

Data Cache Size Limits the amount of memory for caching database index files. Default: 1000 kB. Smaller values are not recommended (performance).

Index Cache Size Limits the amount of memory for caching database data files. Default: 3000 kB. Smaller values are not recommended (performance).

Documentation Editor view



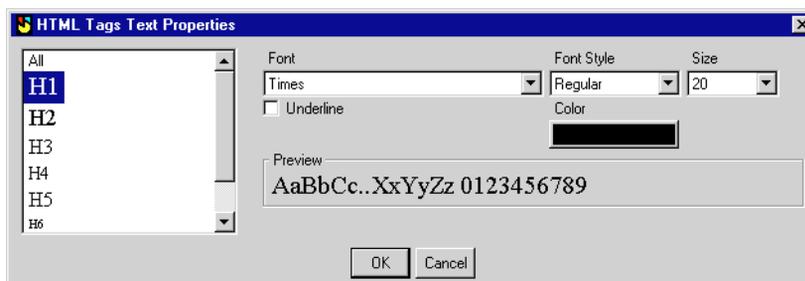
Alignment

Editor Width[chars]

Defines the maximum number of characters allowed on a line in the Documentation view. Lines that are longer than the **Editor Width** are wrapped.

Text Highlighting

Text Properties for HTML Tags Specifies the font and color properties of a number of the HTML tags that are supported by the Documentation Editor. Press **Choose...** to open the HTML Tags Text Properties dialog.



Documentation Creation

Documentation Template Directory Specifies the directory in which your documentation template files are stored. These template files are then used for creating documentation frames for symbols.

You can specify multiple directories in the **Template Directory** field. A colon (:) separates multiple directories from each other. When searching for a template file, SNIFF+ searches the directories in the order they are given and terminates the search as soon as the template file is found.

To learn how to customize template files, please refer to [User's Guide — Creating documentation templates files — page 300](#).

Default: \$SNIFF_DIR/config/docu (SNIFF+ default documentation template files are stored in this directory)

Caution

If you want to use the default documentation template files for documenting any or all of your symbols, make sure to include \$SNIFF_DIR/config/docu in the **Documentation Template Directory** field!

**Sort Symbols
Alphabetically**

Sorts the symbols in the documentation files alphabetically.

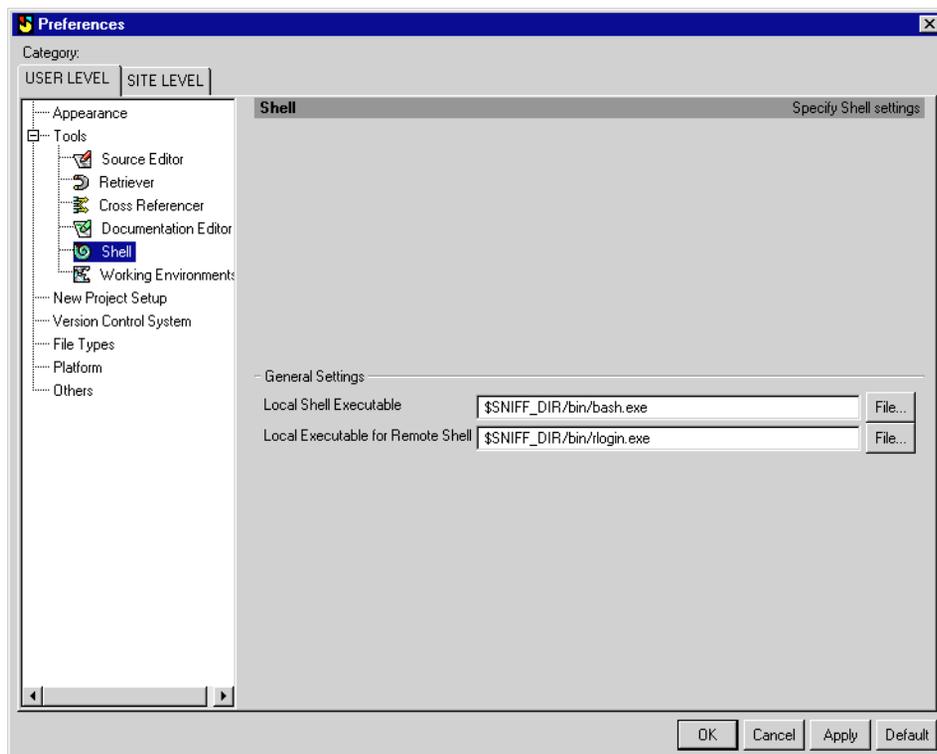
**Use Read-Only
Mode**

Documentation can only be browsed — it can neither be generated nor edited. Obsolete documentation is hidden completely.

Documentable Symbol

Specifies which kinds of symbols are documentable. You can restrict documentable symbols according to access specifiers (Private, Protected, Public).

Shell view



General Settings

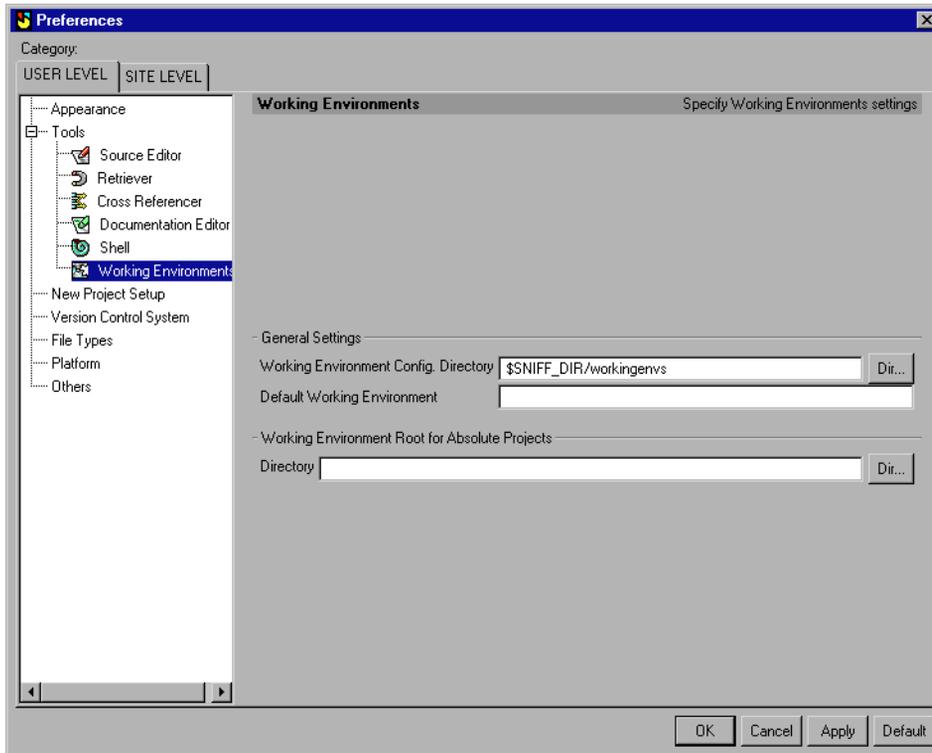
Local Shell Executable

Defines which shell runs in the Shell tool.

Local Executable for Remote Shell

Specifies the Shell Executable in which you can login to a remote host. For detailed information on remote compiling and debugging, please refer to [User's Guide — Remote Compile and Debug — page 217](#).

Working Environments view



General Settings

Working Environment Config. Directory

Defines the location of the directory where Working Environment information files are maintained. By default, this is `$$SNIFF_DIR/workingenvs`. There are two Working Environment files:
`WorkingEnvUser.sniff` - stores user and permissions information
`WorkingEnvData.sniff` - stores the location of Working Environment root directories.

Default Working Environment

Defines the Working Environment where Projects are opened by default, that is when no specific Working Environment is selected.

Default Working Environment for Absolute Projects

Information for Absolute (Browsing-Only) Projects (that do not have a Working Environment as such) is maintained in this directory. Any directory where you have write permissions can be entered. By default, that is, if you do not enter anything in this field, the directory used is:

■ On Windows

```
%SNIFF_DIR%\Profiles\Username
```

■ On Unix

```
$HOME/.sniffrc
```

The following information is maintained under this directory:

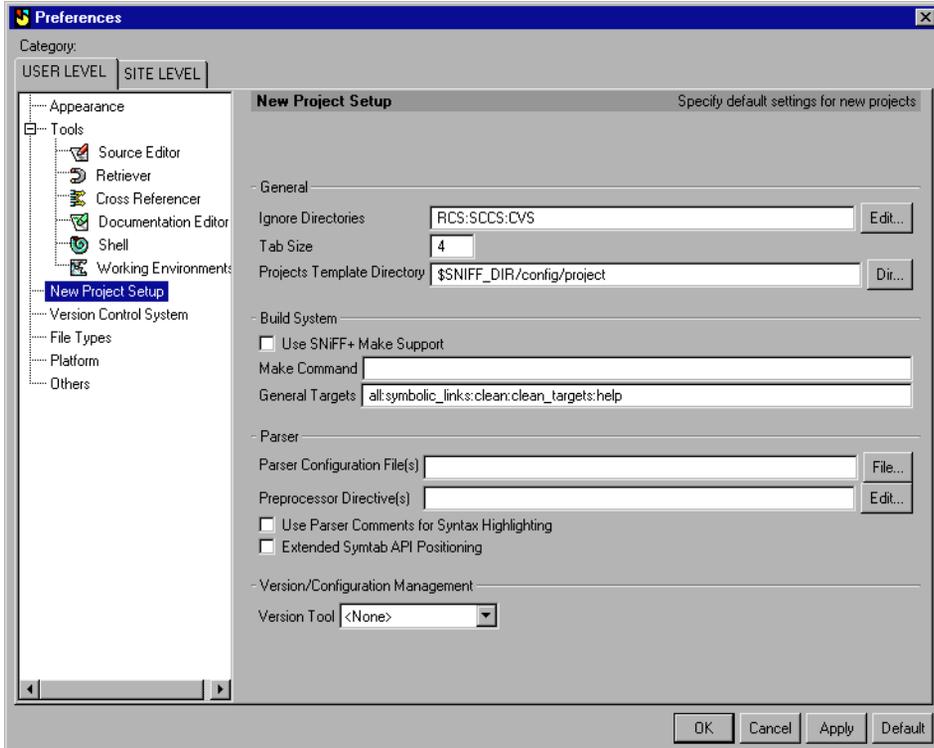
- Project cache files for fast opening of projects
- Retriever index files
- Cross Reference database files

Note that the above information is only written to this directory

- if Absolute Projects are opened outside of a “normal” Working Environment, or
- if this Working Environment (at the root of the Working Environments Tree in the Working Environments tool) is specifically selected for opening absolute projects.
- if the Retriever indexing option and/or the database-driven Cross Referencing option is used.

New Project Setup view

The values that you set here for the various check boxes and fields are used as default settings during the creation of new projects. (Please refer to the *User's Guide* for more information about setting up projects.)



General

Ignore Directories List of directories to ignore during the automatic creation of the project's subprojects. Colons (:) separate multiple directories from each other.

Example: It doesn't make any sense to create a SNIFF+ subproject for a shared data repository directory.

Default: RCS : SCCS : CVS

Tab Size	Spacing (in blanks) between two tab stops. Default: 4
Projects Template Directory	Specifies the location of the template files to be used for setting up new projects. For more information on setting up projects with templates, please refer to User's Guide — Project Setup Overview — page 53 .

Build System

Use SNIFF+ Make Support	Select if you want to use SNIFF+'s Make Support. For details about Make Support, please refer to User's Guide — Build and Make Support — page 83
Make Command	Command to be submitted to the Shell when a Make command is issued. For details about executing Make commands, please refer to User's Guide — Building a project's targets — page 195 . Default: On Unix: <code>gmake</code> On Windows: <code>sniffmake</code>
General Targets	Name(s) of a project's general targets (also known as "help targets"). The general targets are used to drive the Make command and the Debugger. Multiple targets are separated with a colon (:). For details about general targets, please refer to User's Guide — SNIFF+ help targets — page 200 . Default: <code>all:symbolic_links:clean:clean_targets:help</code>

Parser

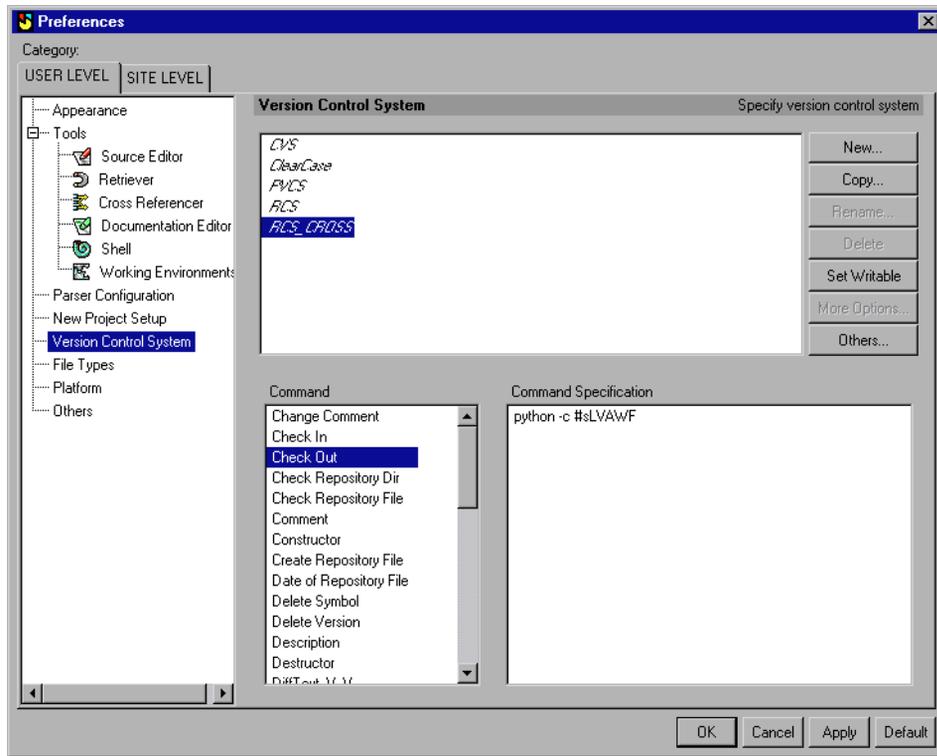
Parser Configuration File(s)	<p>Specifies the location of any parser configuration files that you use for selectively preprocessing your source code. The path specification of the parser configuration file(s) must be absolute. For details about preprocessing source code in SNIFF+, please refer to User's Guide — Preprocessing C/C++ Code in SNIFF+ — page 183.</p> <p>Default: no file (blank)</p>
Preprocessor Directive(s)	<p>Specifies the preprocessor directives you use for preprocessing your source code. The <i>cpp</i> command line syntax should be used (<i>-Dmacro-spec</i> and <i>-Umacro-spec</i> strings separated by blanks). For details about preprocessing source code in SNIFF+, please refer to User's Guide — Preprocessing C/C++ Code in SNIFF+ — page 183.</p> <p>Default: no directives (blank)</p>
Use Parser Comments for Syntax Highlighting	<p>When selected, SNIFF+ uses information from the Parser to recognize a comment. The advantage is that the Parser recognizes the comments and therefore SNIFF+ displays these accurately. The disadvantage is that SNIFF+ uses more memory. When not selected, SNIFF+ uses information from the Editor to recognize a comment. The advantage is that SNIFF+ uses less memory. The disadvantage is that it results in preprocessor related problems, e.g., no macro expansion.</p>
Extended Symtab API Positioning	<p>When selected, parameter names and start and end positions of the argument list and the constructor initialization list are added to the Symbol Table. To activate this option, execute the Force Reparse command in the Project Editor.</p>

Version/Configuration Management

Version Tool	<p>Default version control tool for a project. The list of available tools is determined by the adaptors that are defined for your SNIFF+ installation. SNIFF+ comes with a set of predefined adaptors for some of the most commonly-used version and configuration system tools. Please refer to the <i>Release Notes</i> for more information about the individual tools.</p> <p>Default: <None></p>
---------------------	---

Version Control System view

This view defines the adaptors used by SNIFF+ for integrating 3rd-party version control tools. The **Version Control System** view of the Project Attributes dialog allows you to choose one of the adaptors that are defined in your Preferences.



The following buttons allow you to modify the list of file types:

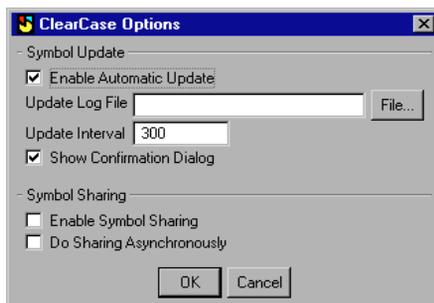
Note

There is no limit to the length of the CMVC commands that you enter in the **CMVC Command Specification** fields. Note, however, that your Preferences are written to a file. The longer the file, the longer the time necessary to read its contents. Therefore, rather than entering a long list of commands in the **CMVC Command Specification** fields, we suggest that you write your commands in scripts and execute these scripts in this view.

- New...** Allows you to create a new adaptor. Pressing the button will open a New Adaptor dialog prompting you for the name of the new adaptor. You will be warned if an adaptor with the same name already exists.
- Copy...** Copies the name of the currently selected adaptor and adds -1 at the end of the existing name.
- Rename...** Opens the Rename dialog in which you can rename the selected adaptor.
- Delete** Deletes the selected adaptor from your Preferences.
- Set Writable** In the **User Level** tab, press this button to copy the currently selected adaptor from the Site level. You can then modify the adaptor at the User level.
- More Options...** Enabled only if **ClearCase** is selected. Opens the [ClearCase Options dialog — page 150](#).

ClearCase Options dialog

The following applies for ClearCase only.



Symbol Update

Enable Automatic Update	Enable to start automatic periodic updating.
Update Log File	Specifies the name of the log file containing information about the updates.
Update Interval	Specifies the interval at which SNIFF+'s should check the Update Log File.
Show Confirmation Dialog	When selected, whenever SNIFF+ checks the Update Log File, a dialog will appear with a list of all the files checked-in since the last check of the log file.

For more information, please refer also to the [User's Guide — Notifying SNIFF+ of files checked-in with ClearCase — page 282](#)

Symbol Sharing

■ General

SNIFF+ symbol files can be shared among views, via winkin, as derived objects. To achieve this, a `.shared` directory holding the symbol files is checked in to the project root directories.

The whole mechanism is fully automated, and requires no user action apart from the steps outlined below.

Because this feature demands a certain overhead (creation of derived objects and winkin), it should only be applied where a number of similarly configured views are used.

■ Initial activation

For initial activation, open your root project(s) (**With Symbols enabled!**) in a view with a baseline configuration and no branches.

After setting the symbol sharing options, close the project(s) in the current view.

Once the projects have been closed after the initial setting, they can be opened in other views.

- **Dialog check boxes**

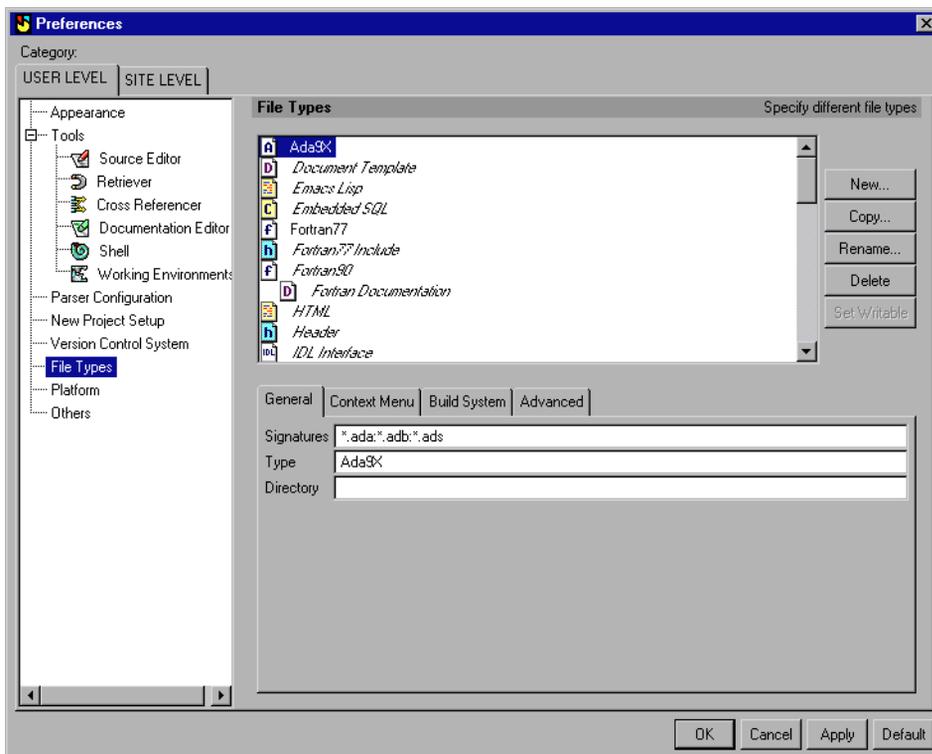
Enable Symbol Sharing Makes SNIFF+ symbol files available to other views, via winkin, as derived objects.

Do Sharing Asynchronously If symbol sharing is enabled, derived object files are created as needed when projects are closed. Depending on the project size and ClearCase server performance, this may take some time. Enable this option to run the sharing process in the background.

Caution: Do not attempt to immediately reload a project before the sharing process has run its course!

File Types view

Each time a new project is created, the file types are copied from your Preferences to the new project attributes. After project creation, each project manages its own file types. If you want to modify the file types of a project, use the Project Attributes dialog. The file types in your Preferences should only be modified if you want to modify the file types globally for future projects.



You can modify the list of file types with the following buttons:

New...	Allows you to create a new file type. When the button is pressed, a New File Type dialog opens, prompting you for the name of the new file type. You will be warned if a file type with the same name already exists.
Copy...	Copies the name and attributes of the currently selected File Type and adds -1 at the end of the existing name.
Rename...	Opens a dialog in which you can rename the currently selected file type.
Delete	Deletes writable copies of the selected file type.
Set Writable	Press this button to create a writable copy of the currently selected file type.

Tabs

General tab

Signatures	Specifies the pattern for the file type using shell regular expressions. SNIFF+ uses signatures to determine the file type of a given file. A file type can have more than one signature. Multiple signatures are separated from each other by a colon (:). SNIFF+ keeps an alphabetical list of file types. If a file matches the signature of more than one file type, SNIFF+ associates the file with the first file type it finds in the list.
Type	Specifies a generalized (pre-defined) name for the File Type (e.g. Implementation or Header).
Directory	Specifies the directory where files of this file type are stored. If the Directory specification is a relative path, it is relative to the Project Directory. An absolute path can also be specified.

Context Menu tab

- Context menu...** Opens the Context menu dialog. Here you can modify the commands that appear in the context menu in the Project Editor's File List.
- Command Label** Specifies the menu command labels that should appear in the Project Editor's File List context menu. For each **Command Label**, you must specify a tool name in the **Corresponding Command** list. Multiple names are separated with a colon (:).
- Corresponding Command** Specifies the possible tools that can be opened from the context menu. Multiple tools are separated with a colon (:). The tools can be called in the Project Editor's File List by selecting a file and then pressing the right mouse button. The first tool in the list is the default tool.
- The following predefined tools can be opened from the Project Editor's File List context menu:

Tool name	SNiFF+ tool	Can be used for
SniffEdit	Source Editor	All file types
DocBrowser	Documentation Editor	Documentation
SniffOpen	Launch Pad	Project Description Files

Commands are executed in the Shell tool and can contain the following variables that will be expanded before execution of the command:

%d	full path of Project Description File (PDF)
%f	full path of source file
%F	base name of source file
%D	source directory of project
%l	repository path without the tool-specific extension

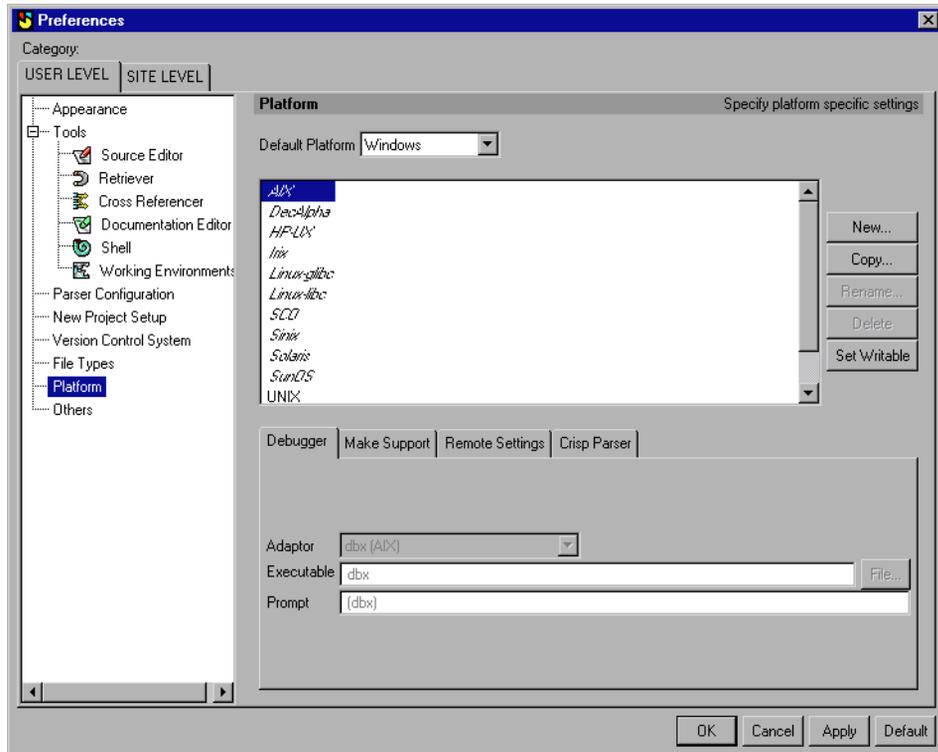
Build System tab

- General Makefile** Specifies the Language Makefile. For details, please see [User's Guide — Language Makefiles — page 91](#).
- Generated From** Defines from which other file type this type is generated. For example, object files are generated from source files.

Advanced tab

- Source Code Parser** Defines which SNIFF+ parser is used for extracting symbolic information from files of this file type. If this field is empty, the file is not parsed at all.
- Icon for File Type** Specifies the icon that should be associated with the file type. It is shown whenever the file is shown in SNIFF+. SNIFF+ comes with a set of icons for most of the predefined types. If the directory specification is a relative path, it is relative to the project directory. An absolute path can be specified instead. By clicking the **File...** button, you can browse and select an icon from the File dialog.
- Add/Remove Automatically to/from Project** Select if you want SNIFF+ to add/remove files of a particular file type automatically to/from a project. We suggest that you select this check box for “derived files” (e.g., object files, template files, c files generated by lex, ref, IDL, or other tools). On the other hand, you should clear this check box for files that you share with other team members (e.g., source files). Note that you cannot add/remove file of file types for which this check box is selected; SNIFF+ will add/remove these files for you. If SNIFF+ cannot find files during the opening or updating of a project, you will not see these files in your project. Furthermore, if you delete a file outside of SNIFF+ and then select it in the Project Editor, an Alert dialog appears and the file disappears from the File List.
- Generated in Object Directory** The files that are automatically added/removed are loaded from the object redirection directory of the current platform when a project is opened or reloaded.
- Is Default File Type** Defines whether the file type should be automatically loaded into a new project during the project's creation.

Platform view



Default Platform The default platform that you specify in this field will be used for all Working Environments where no platform is specified and for projects that aren't in a Working Environment, i.e., absolute projects.

You can modify the list of platforms with the following buttons:

- New...** Allows you to specify a new platform. When the button is pressed, a New Platform dialog opens, prompting you for the name of the new platform.
- Copy...** Copies the name and attributes of the currently selected platform and adds -1 at the end of the existing name.
- Rename...** Opens a dialog in which you can rename the currently selected platform.

Delete	Deletes writable copies of the selected platform.
Set Writable	In the User Level tab, select this button to copy the currently selected platform from the Site level. You can then modify the specifications of the platform at the User level.

Tabs

Debugger tab

Adaptor	List of supported debuggers (see the <i>Release Notes</i> for a list of currently supported debugger back ends). Choosing an item from the list instructs SNIFF+ to use the correct adaptor for the back-end process and also fills the other fields with default values.
Executable	Name of the back-end executable. If only the name of the debugger executable is specified without a path, SNIFF+ searches for the executable in the default command path list. To call a specific executable, specify the complete path.
Prompt	Must match the prompt of the back-end debugger. This attribute should be changed only if the prompt of the back-end debugger does not match the suggested default value. If Prompt does not match the prompt of the back-end debugger, the SNIFF+ Debugger will not work correctly.

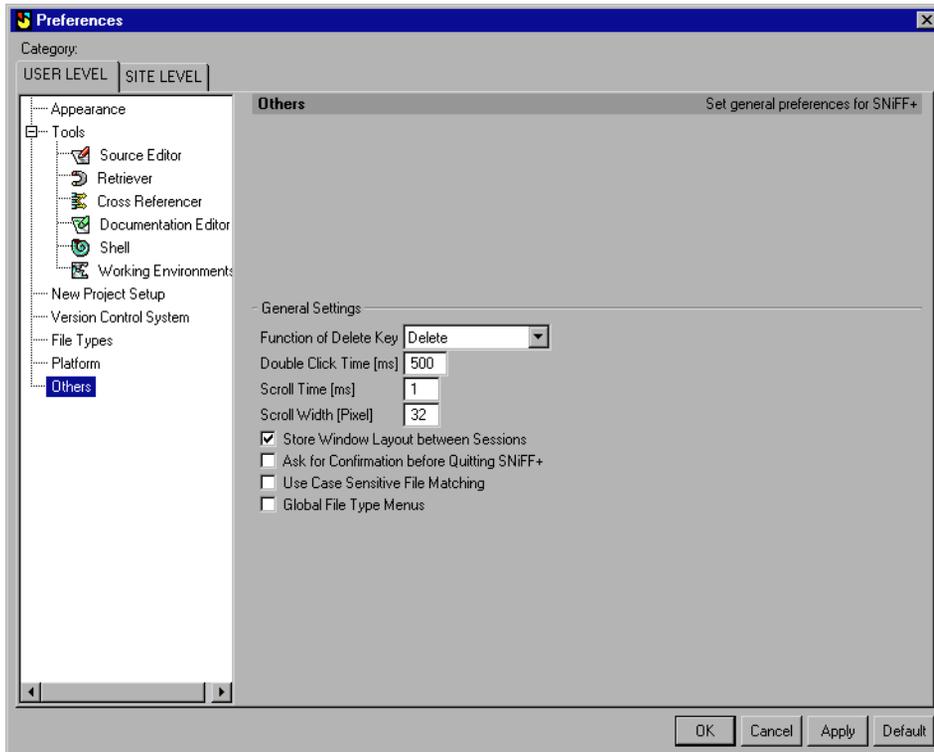
Make Support tab

Make Command	Specifies the Make command for each platform, e.g. <code>sniffmake</code> on Windows and <code>gmake</code> on Unix.
Platform Makefile	Sets the Platform Makefile used during builds. For details, please refer to User's Guide — Platform Makefile — page 92 .
Object Directory	Directory where the object files will be generated.
Target Directory	Directory where the target files will be generated.

Remote Settings

Host	Defines the build and debug host of the specified platform and checks to see whether a local or remote shell should be run in the Shell tool and the Debugger.
Wait_Seconds for Shell to Become Ready	Specifies the number of seconds reserved for remote shell script execution. If the scripts need more time to execute, increase this value.
User Name	Specifies the name of the User working on the remote host.
Remote Host SNIFF_DIR	Specifies the path to the SNIFF+ installation directory on your remote host. For detailed information on remote compiling and debugging, please refer to User's Guide — Remote Compile and Debug — page 217 .

Others view



General Settings

- Function of Delete Key** Defines the function of the <Delete> key to be either **Delete** (delete the character right of the cursor) or **Backspace** (delete the character left of the cursor). Set the value according to your keyboard.
- Double Click Time [ms]** Specifies the time in milliseconds (ms) in which SNIFF+ recognizes a second mouse click as a double-click.
- Scroll Time [ms]** By default, Scroll Time is set to 1 which is the maximum scrolling speed. To make it slower, increase the value in the range 1 to 500.
- Scroll Width [Pixel]** By default, Scroll Width is set to 32. You can change the Scroll Width by selecting a value between 1 and 500.

Store Window Layout between Sessions	Defines whether the window layout is stored between sessions. Default: selected (window layout stored between sessions)
Ask for confirmation before Quitting SNIFF+	Defines whether SNIFF+ asks for confirmation when Quit SNIFF+ is chosen from the Tools menu. Default: not selected (do not ask before quitting)
Use Case Sensitive File Matching	Determines how include directives are generated and external requests (Sniffaccess and external editors) are handled. Default: selected (case sensitive file matching used)
Global File Type Menus	When selected , adds the commands specified in the Preferences — File Types view — page 153 to the context menu in the Project Editor's File List. When not selected , adds the commands specified in the Project Attributes — File Types view — page 182 to the context menu in the Project Editor's File List.

Project Attributes

Introduction

When you select a project or subproject and then choose **Project > Attributes of Project *Projectname...***, the Project Attributes dialog appears. This dialog also appears when you double-click on a project in the Project Tree. Default values for some of these attributes are defined in your [Preferences — page 123](#).

Changes to the project attributes take immediate effect if not otherwise specified in the text below. Attributes of frozen projects or of projects with a read-only project description file (PDF) cannot be modified.

SNiFF+'s Project Attributes

Project attributes are organized in five categories in the Project Attributes dialog:

- General
- Build Options
- Parser
- Version Control System
- File Types

All views contain the following two buttons:

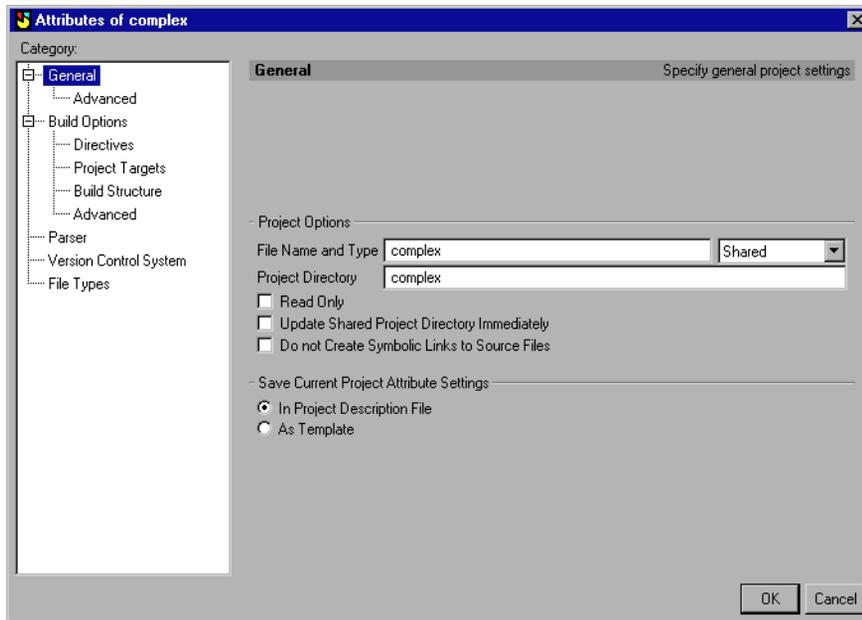


OK Editing the project attributes and pressing **OK** only modifies the project in memory and sets the state of the project to modified (indicated by a modified icon in the Project Editor and the Launch Pad). To save the Project Description File to disk, choose **Save *project*** from the **Project** menu.

Cancel Discards all modifications, leaving the project attributes unchanged.

General view

The following illustration shows the **General** view of the Project Attributes dialog:



Project Options

- File Name and Type** Specifies the name of the Project Description File and automatically shows the type of the project.
- Project Directory** Specifies the main Project Directory. In the standard case, this path is set when the project is created and is never changed. If the path specification (which can contain shell environment variables) evaluates to an absolute path, this is an absolute project (see “File Name and Type” above). If the path specification of this attribute evaluates to a relative path, this is a shared project and the complete Project Directory path is formed by adding this relative path to the directory path specified by the working environment root directories.
- Read Only** Indicates that the project is a library and doesn't use Shared Source Working Environments.

- Update Shared Project Directory immediately** Should only be selected if you want to immediately update the Shared Source Working Environment when you check in a file. If this button is selected and you check in a file, the file is automatically checked out to the Shared Source Working Environment (SSWE) that the current working environment accesses. This means that you must have write permissions in this SSWE. We recommend that you do not select this attribute and only update working environments at regularly scheduled times.
- Do not create Symbolic Links to Source Files** Symbolic links (local copies on Windows) between sources in Shared and Private Working Environments need only be created if you use SNIFF+ Make Support. Also, if you only want to browse (not compile) sources in a Shared Working Environment there is no need for symbolic links (local copies) to be created.

New Project Options

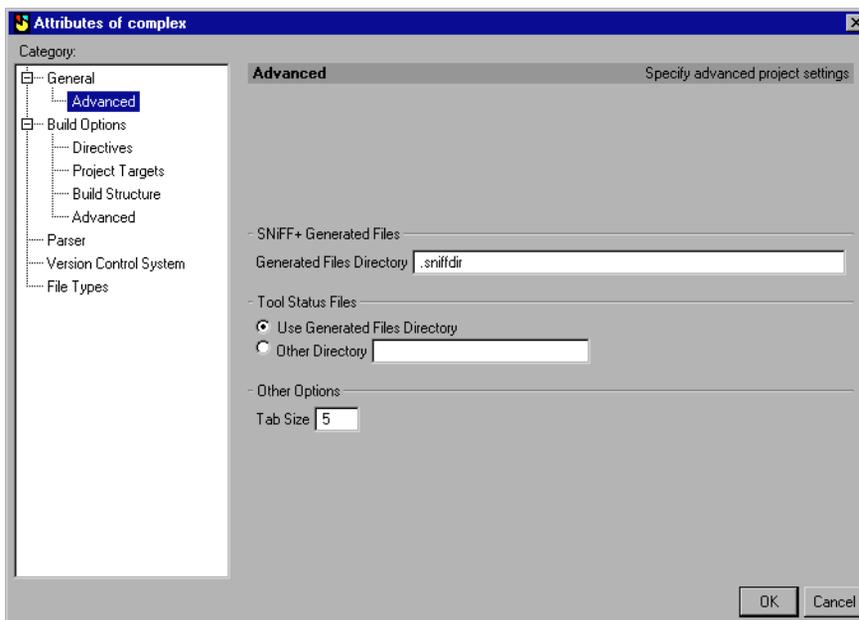
- The following are **only** visible when you generate a new project using the Launch Pad's **Project > New Project > with Defaults** or **Project > New Project > with Template** menu commands.

- Ignore Directories** List of directories for which subprojects are not to be created during the automatic creation of the project's subprojects. Colons (:) separate multiple directories from each other.
Example: It doesn't make any sense to create a SNIFF+ subproject for a repository directory. **Default:** RCS : SCCS : CVS
- Destination Directory** You can specify the directory where SNIFF+ should store the newly created project description files.
- Ignore Directories without Project Specific Files** Ignores directories which do not contain files that match the file types specified for the project.
- Generate Subproject Tree** Subprojects are automatically generated for all the subdirectories in the project root directory.
- Generate Unique Project Names** When selected, specifies that each project should have a unique name.
- Follow Symbolic Links (Unix only)** Specifies whether symbolic links to other directories should be followed and whether new projects should be generated for those directories.

Save Current Project Attribute Settings

- In Project Description File** Saves the current project attribute settings in the project description file.
- As Template** Saves the project description file as a template in the **Projects Template Directory** specified in the Preferences. For more information on setting up projects with templates, please refer to [User's Guide — Project Setup Overview — page 53](#).

General Advanced



SNIFF+ Generated Files

Generated Files Directory

Indicates the directory where SNIFF+ stores the generated files for this project. A relative path is always relative to the Project Directory. By default, the generated files are stored in `.sniffdir` (created in the Project Directory). You may want to change the Generated Files Directory if you do not have write permission in the Project Directory. SNIFF+ displays a warning message in the Log window if permissions prevent writing to the Generated Files Directory. See also [SNIFF+ - Generated Files — page 319](#).

Tool Status Files

Use Generated Files Directory Points to the directory where SNIFF+ stores the tool status files. You may want to change the directory if you do not have write permission at the default location. If SNIFF+ cannot write the tool status file of a project, the window positions, contents and tool histories are not restored the next time the project is opened.

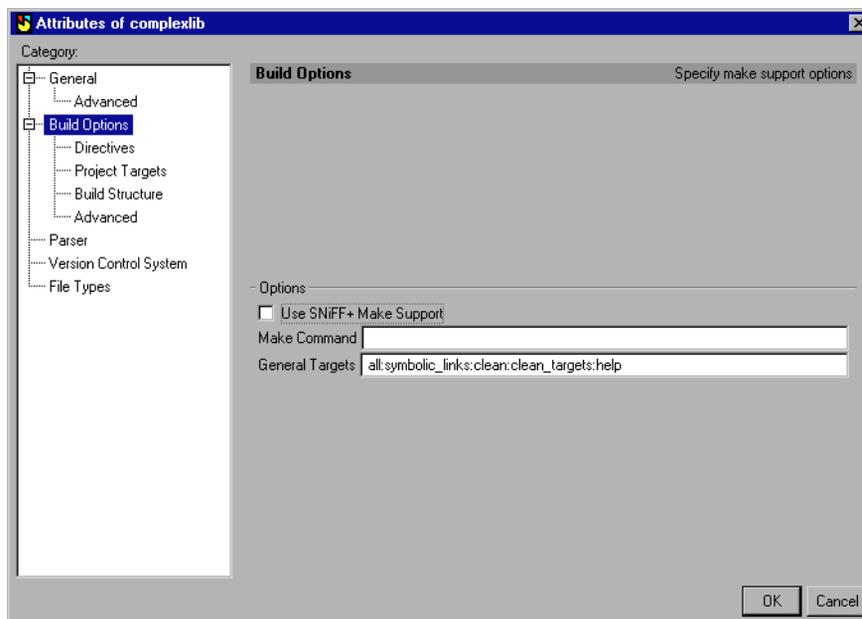
Other directory You can specify a directory in which SNIFF+ should store state files.

Other Options

Tab Size Specifies the number of spaces to use for tab size.

Build Options view

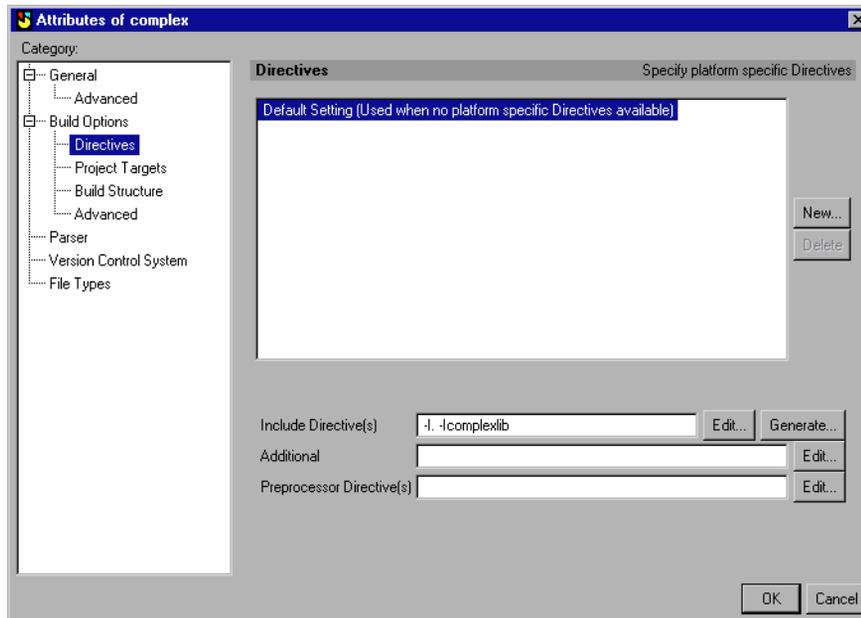
The project attributes that are relevant for Make Support are grouped together in the **Build Options** view. The default values for the various fields and check boxes in the view are taken from your [Preferences — page 123](#).



Options

- Use SNIFF+ Make Support** Select to use SNIFF+'s Make Support.
- Make Command** Command submitted to the Shell when a Make command is issued. You can attach Make arguments to the command. Furthermore, you can also use your own shell scripts.
Default: On Unix: gmake. On Windows: sniffmake
- General Target(s)** Name(s) of a project's general target(s). The general targets are used to drive the Make command and the Debugger. Multiple targets are separated with a colon (:). (Please refer to the *User's Guide* for a description of the general targets.)
Default:
all: symbolic_links: clean: clean_targets: help

Directives

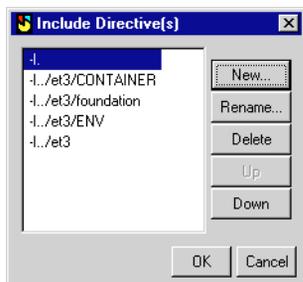


You can add a new platform or delete an existing platform from the Platform list using the following buttons:

- New...** Allows you to add a new platform to the Platform list. When the button is pressed, a New Platform Setting dialog opens, prompting you for the name of the platform. You can scroll down the list and select a platform.
- Delete** Deletes the platform from the Platform List.

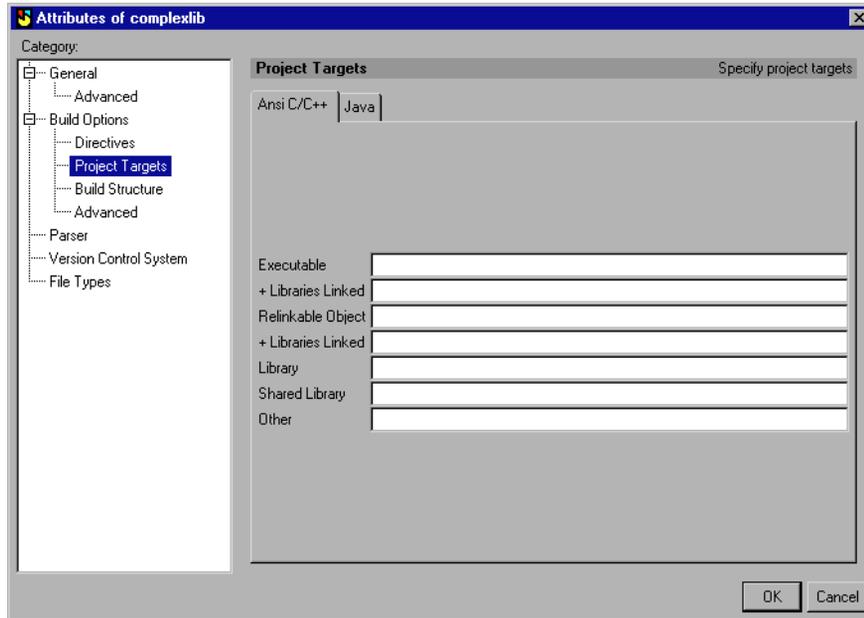
Include Directive(s)	List of include directive(s) that are used by the preprocessor. This field is initially empty. Press the Generate button, to generate include directive(s) for the project. As a rule, you should use the Generate button whenever you've changed a project source file to include a file that isn't listed. Directives cannot be generated for included files that are external to the SNIFF+ project structure. Enter these in the Additional field (see below).
Additional	List of additional include directive(s) used by the preprocessor. Preprocessor command line syntax is used in this field (<i>-Ipath</i> strings separated by blanks). Additional include directives point to included files that are external to the project structure, and that cannot therefore be generated (see above).
Preprocessor Directive(s)	Specifies the directives for the preprocessor (excluding the include directives, which are defined by the Include Directives attribute). Preprocessor command line syntax is used in this field (<i>-Dmacro-spec</i> and <i>-Umacro-spec</i> strings separated by blanks).
Edit...	Opens the Include Directive(s) , Additional and Preprocessor Directive(s) dialog.

Edit Directives dialog



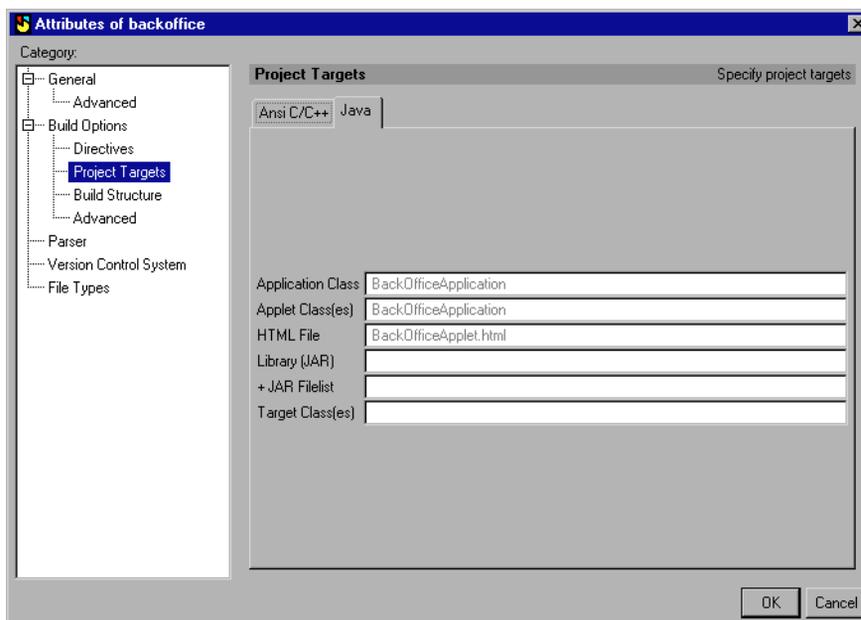
New...	Opens a dialog where you can enter a new directive name.
Rename...	Opens a dialog where you can rename the currently selected directive.
Delete	Deletes the currently selected directive.
Up	Moves the currently selected directive one position up in the list.
Down	Moves the currently selected directive one position down in the list.

Project targets - C/C++



Executable	The name of the executable to be built.
+ Libraries Linked	The libraries that are linked to the executable.
Relinkable Object	The name of the relinkable object to build.
+ Libraries Linked	The libraries that are linked to the relinkable object.
Library	The name of the static library to be built.
Shared Library	The shared library to be built.
Other	User-defined targets. Note that SNIFF+ doesn't provide Make rules for building user-defined targets.

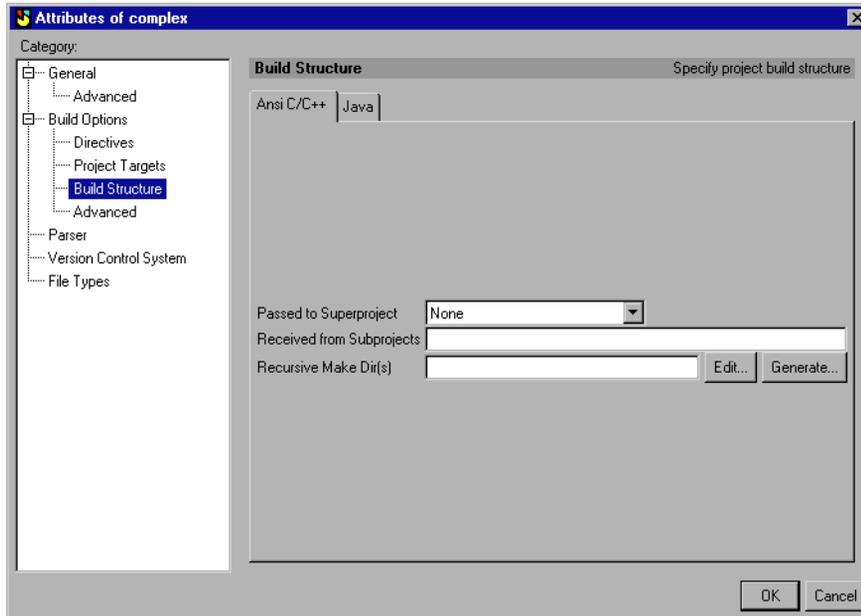
Project targets - Java



Note: Please refer to the Java Technical Reference in the *Java Tutorial* for more information.

- | | |
|--------------------------|--|
| Application Class | Specifies the application target class. |
| Applet Class(es) | Specifies the applet target classes. |
| HTML File | Specifies the HTML file which embeds the applet. If no file is specified SNIFF+ generates a default file. Only works for JDK 1.1.x, because of changes in behaviour of JDK 1.2 appletviewer. |
| Library (JAR) | Specifies the name of the library target to be built. |
| + JAR Filelist | List of files used by your project but not compiled as part of the project. These files must, however, be relative to the Project directory. |
| Target Class(es) | Specifies other target classes (e.g., for Java beans). |

Build Structure - C/C++



Passed to Superproject

Select the type of object that is to be exported to the superproject. For details, please refer to [User's Guide — Build and Make Support — page 83](#).

Received from Subprojects

The objects that are exported from the subprojects of the project are listed in this box. These objects are prerequisites that are used to build the project's targets.

Note that you cannot alter the contents of this box.

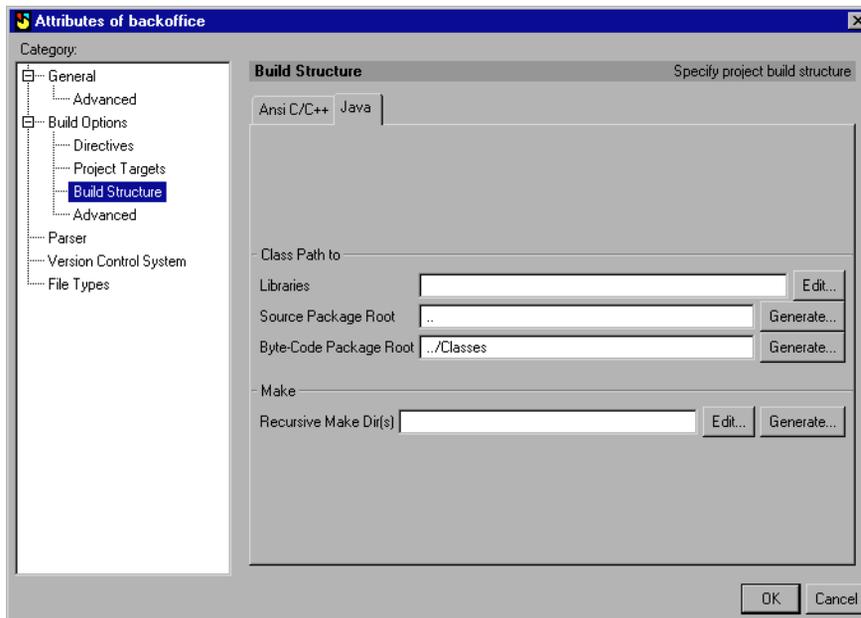
Recursive Make Dir(s)

This field lists the subprojects in which Make is to run recursively. For details, please refer to [User's Guide — Building targets recursively — page 98](#).

Edit...

Opens the Recursive Make Dir(s) dialog. This dialog is very similar to the Include Directive(s) dialog. See also [Edit Directives dialog — page 171](#).

Build Structure - Java



Class Path to

Libraries

This setting is used to provide a list of absolute paths to library byte-code, zip or jar files used in you project, if there is no source code available for them.

However, if the source code **is** available for the libraries, as is the case for the JDK library, it is best to create SNIFF+ Projects for them and then add these as subprojects. There is then no need to enter anything in the Classpath field for these projects.

The advantage of creating and adding subprojects is that you can then also correctly browse inheritance relationships and cross-references etc.

You can also enter the class path to external source code packages that you have **not** added to your SNIFF+ project as subprojects. This will allow correct compilation, but you will not see any symbol information.

This applies also to libraries where no sources are available; SNIFF+ will recognize the data types in such files, but not the symbol names.

If you use **RMI**, you have to enter the path to the JDK class files in this field.

Source Package Root This setting is the class path as you would enter it after `-classpath`. You can specify this relative to the current project or, in the case of an absolute project, as an absolute path. The class path to the Source Package Root ends where the package begins. That is, it is always one directory level **higher** than the highest-level directory containing code in named packages. Bear this in mind especially also during Project Setup. Once the project has been set up, this can be set relative to the project, as seen from the package (e.g. `..` or `../..` etc.)

If you specify the class path to the Source Package Root for a root project, it is generated correctly for all sub-projects. For subsequent modifications, it is easiest to checkmark all projects in the Project Editor and to choose **Project > Attributes of Checkmarked Projects**. Then press the **Generate** button to the right of the field.

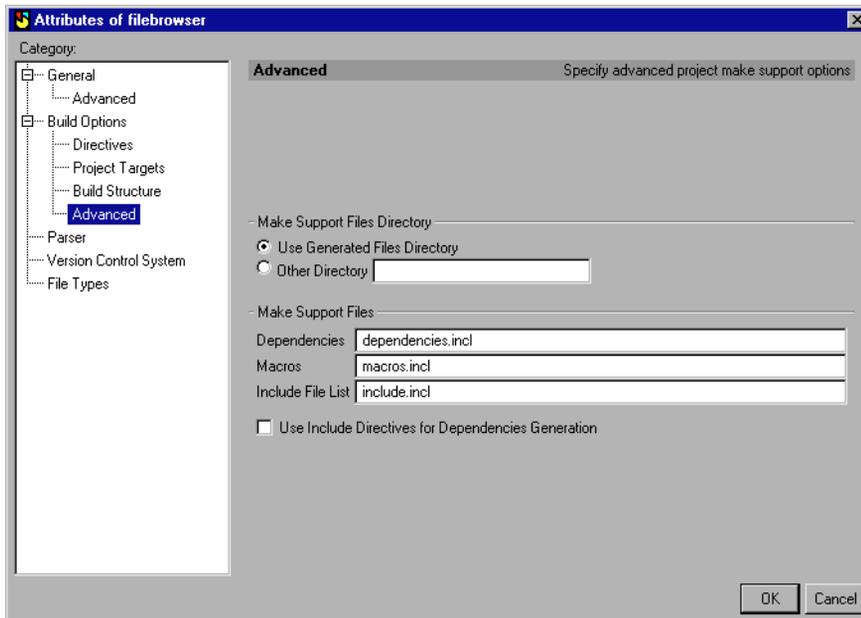
Byte-Code Package Root By default, byte-code is generated to the same directories as the source code. If you prefer to keep your source and byte-code separately, enter a root directory (relative or absolute) where you would like your byte-code packages to start, then press the **Generate** button. SNIFF+ will create the specified directory, and recreate the package structure. Byte-code will be generated to this directory when you build your project.

If you simply enter a name for a directory, it will be created in the current project directory.

Make

Recursive Make Dir(s) Lists the subprojects where Make is to run recursively.

Build Options Advanced



Make Support Files Directory

Use Generated Files Directory

Location of the generated Make Support files. By default, the generated Make Support files are located in the directory that is specified in the **Generated Files Directory** field of the **General > Advanced** view. If you do not use the default value, you will have to modify the project Makefile to reflect the location of the Make Support files.

Note

You can set the `SNIFF_MAKEDIR` macro in the project's macros file to the directory in which the generated Makefiles are located (default: `.sniffdir`). As a result, Makefiles need not be modified when the location of the generated Make Support files changes.

Other Directory You can specify a directory in which SNIFF+ should store generated Make Support files.

Please note: The path specified in this field must be relative since absolute paths are not currently supported.

Once you've specified another directory for Make Support files, you must manually enter the path to this directory in your project Makefiles.

Make Support Files

These fields are enabled if you use SNIFF+ Make Support; you can set this in the [Build Options view — page 169](#).

Dependencies Name of the dependencies file that is generated for the project.

Macros Name of the macros file that is generated for the project.

Note that SNIFF+ automatically updates the Macros Make Support file of a project when you do one of the following:

- Choose the **Add/Remove Files to/from *project*...** command, or the **Add Subproject to *project*** command, or the **Remove Subproject *subproject*** command from the **Project** menu in the Project Editor.
- Modify any Make attribute of the project.

Make Support files are automatically added to and removed from projects, since they are associated with the SNIFF+ **Make Support** file type. (The **Add/Remove Automatically** attribute of this file type is selected.) When you create a new project, you have to explicitly

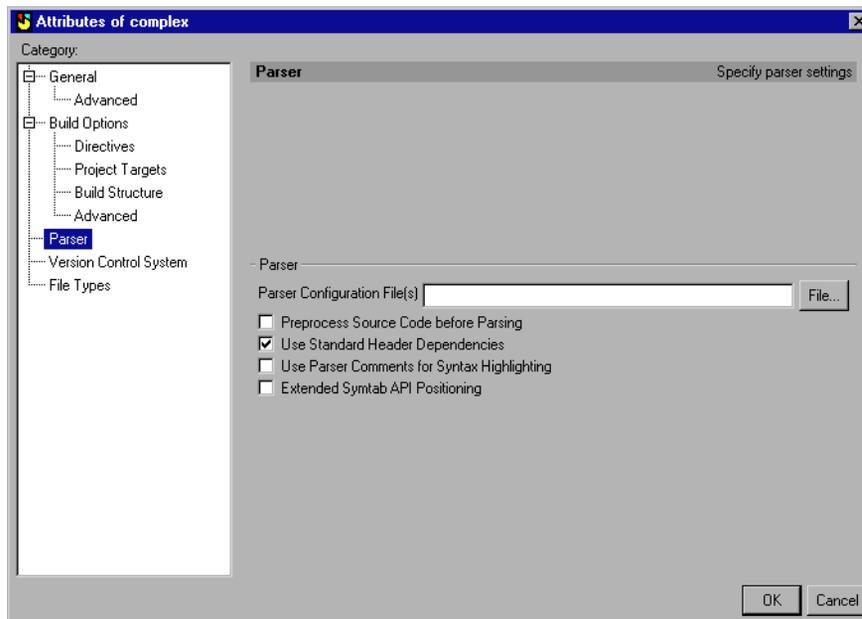
tell SNIFF+ to add the **Make Support** file type to the new project. You can do this by selecting the file type in the **File Types** view of the Attributes of a New Project dialog and then pressing the **Copy from Preferences** button (in the same view).

Include File List Name of the file that contains the include file list for the project.

Use Include Directives for Dependencies Generation Determines whether the include directives (from the **Include Directive(s)** field, see [Directives — page 170](#)) should be taken into account when generating the project's dependencies file.

Parser view

You can look at or modify the parser settings of a project in the **Parser** view. The default values of the various fields and check boxes in this view are taken directly from your Preferences. See also [Preferences — page 123](#)

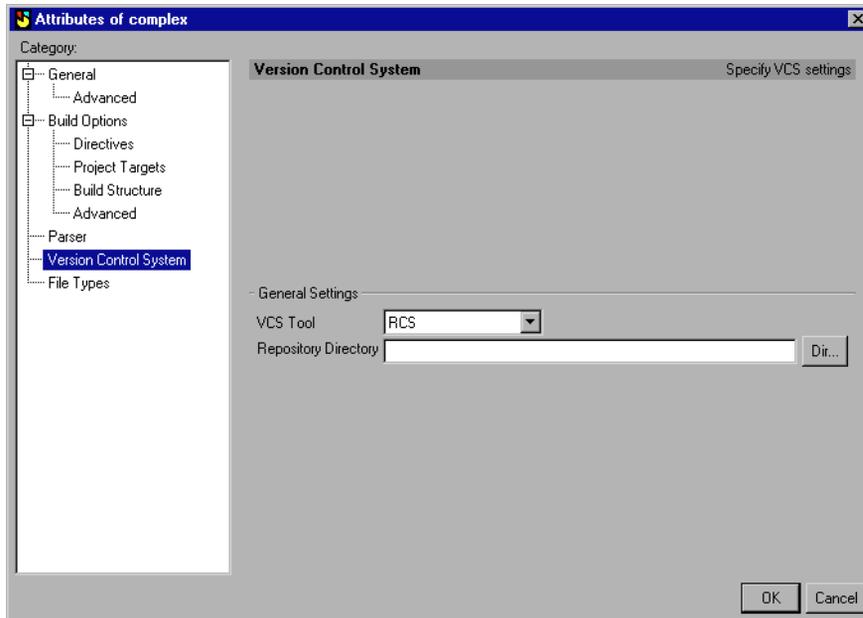


Parser

Parser Configuration File(s)	Specifies the Parser Configuration File(s). Multiple files are separated with a semi colon (;). The files must be specified with an absolute path (but can contain environment variables which evaluate to an absolute path). If the list of files or their contents change, you should reparse the project by choosing Reparse from the Project menu. Default: no files (also none inherited from Preferences).
Preprocess Source Code before Parsing	Specifies whether the source code of this project should be preprocessed before parsing. If you change the setting, you should reparse the project by choosing Force Reparse from the Project menu.
Use Standard Header Dependencies	SNiFF+ also considers standard includes, for example <code>#include<file.h></code> for dependencies generation.
Use Parser Comments for Syntax Highlighting	When selected, SNiFF+ uses Parser information to recognize comments, and is therefore more accurate. However, more memory is required.
Extended Syntab API Positioning	When selected, parameter names and start and end positions of the argument list and the constructor initialization list are added to the Symbol Table. To activate this option, execute the Force Reparse command in the Project Editor.

Version Control System view

The **Version Control System (VCS)** view groups attributes that are related to the version controlling of projects. The default VCS tool can be specified in the Preferences — [Version Control System view — page 149](#).



General Settings

- VCS Tool** Version control tool used for this project. The drop-down lists tools for which pre-defined adaptors are provided. Please refer to the *Release Notes* for more information on specific tools. If no adaptor is available for your tool, you can define a new one. **Default:** RCS.
- Repository Directory** Note that SNIFF+ specifies a default value for the Repository Directory, so you can generally leave this field blank. If you do, SNIFF+ assumes that each Repository Directory is a subdirectory of its corresponding Project Directory. If you want to specify another location for your repository, enter the path to the repository in this field. Note that multiple SNIFF+ projects can share a single Repository Directory.

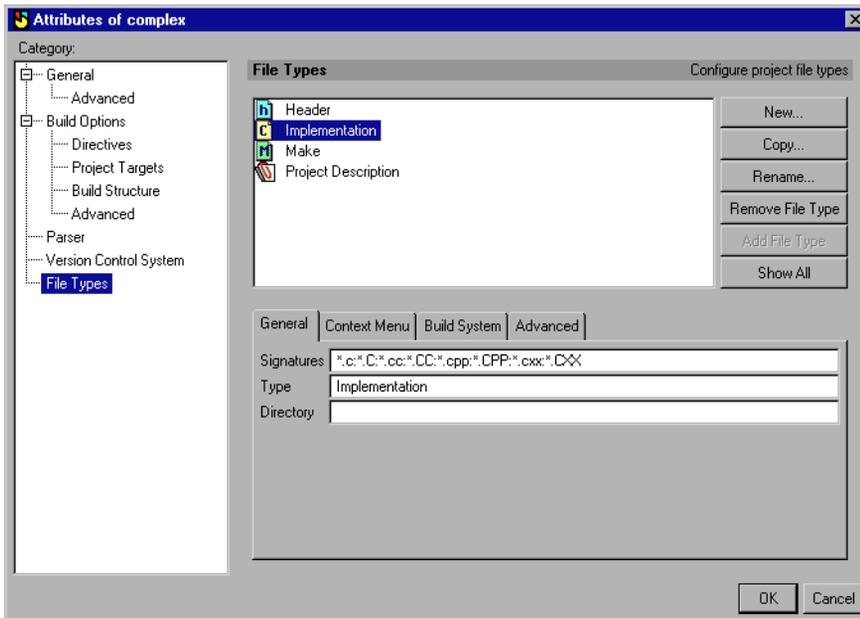
File Types view

SNiFF+ handles different kinds of files by associating them with file types. SNiFF+ comes with a set of predefined commonly used file types.

During the creation of a project, the file types are copied from your Preferences to the project attributes. Each project has its own set of file types, so modifying the file types in the Preferences does not affect already existing projects.

New file types can be created by pressing **New...** For details, please see [To set File Types attributes — page 143](#).

The following illustration shows the **File Types** view:



Typography of File Types List

- Bold** File type is part of the project. To remove this file type from the project, select it and press the **Remove** button.
- Italics** File type is not part of the project. To add this file type to the project, select it and press the **Add File Type** button.

You can modify the list of file types with the following buttons:

New...	Allows you to create a new file type. When the button is pressed, a New File Type dialog opens, prompting you for the name of the new File Type. You will be warned if the name already exists.
Copy...	Copies the currently selected file type together with its attributes and adds -1 at the end of the existing name.
Rename...	Opens a dialog in which you can rename the currently selected File Type.
Remove File Type	Removes the selected File Type from the Project.
Add File Type	Adds a pre-configured File Type to the Project (to see all pre-configured File Types, press the Show All button).
Show All / Hide Unused	File Types which are not part of the Project (<i>italics</i>) will be shown/hidden.

File Types — Tabs

General tab

Signatures	Specifies the pattern for the file type using shell regular expressions. SNIFF+ uses signatures to determine the file type of a given file. A file type can have more than one signature. Multiple signatures are separated from each other by a colon (:). SNIFF+ keeps an alphabetical list of file types. If a file matches the signature of more than one file type, SNIFF+ associates the file with the first file type it finds in the list.
Type	Specifies a generalized (pre-defined) name for the File Type. Some File Types (<i>Implementation</i> , <i>Object</i> and <i>IDL</i>) have a special significance in SNIFF+. These are described under Special File Types below).
Directory	Specifies the directory where files of this file type are stored. If the directory specification is a relative path, it is relative to the Project Directory. An absolute path can also be specified.

Special File Types

SNiFF+ uses generalized File Types, some of which have a special significance in the SNiFF+ Make Support system.

■ Implementation File Type

If you add a new file type (extension `*.myC`) to a SNiFF+ project, and you define `*.myC` files to be of the *Implementation* type, the SNiFF+ Make Support system will know that

- the `*.myC` files are compilable, and that, by default, `*.o` files are to be generated
- dependencies have to be checked.

■ Object File Type

If you add an *Object* file type and derive it from a given implementation file type (e.g. from the `*.myC` files used in the above example), you can specify a custom extension for generated object files.

■ IDL

If you specify files to be of the *IDL* type, the targets for these files will always be generated prior to all other (standard) targets.

Context Menu tab

This tab allows you to configure the right-click context menu of the Project Editor's File List according to file types.

Command Label Specifies the menu command labels that should appear in the Project Editor's File List context menu. For each **Command Label**, you must specify a tool name in the **Corresponding Command** list. Multiple names are separated with a colon (:).

Corresponding Command Specifies the possible tools that can be opened from the context menu. Multiple tools are separated with a colon (:). The tools can be called in the Project Editor's File List by selecting a file and then pressing the right mouse button. The first tool in the list is the default tool. The following predefined tools can be opened from the Project Editor's File List context menu:

Command	SNiFF+ tool	Can be used for
SniffEdit	Source Editor	All text file types
DocBrowser	Documentation Editor	Documentation
SniffOpen	Launch Pad	Project Description Files

Commands are executed in the Shell tool and can contain the following variables that will be expanded before execution of the command:

%d	full path of Project Description File (PDF)
%f	full path of source file
%F	base name of source file
%D	source directory of project
%l	repository path without the tool-specific extension

Build System tab

General Makefile Specifies the Language Makefile. For details, please refer to [User's Guide — Build and Make Support — page 83](#).

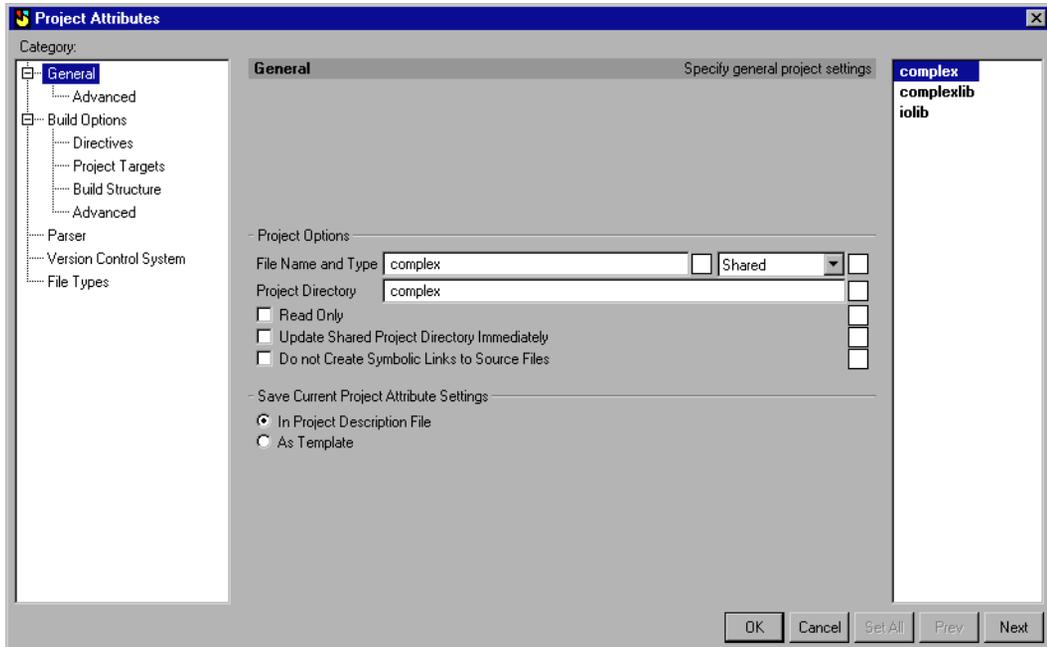
Generated From Defines whether this file type is generated from another file type. For example, object files are generated from source files.

Advanced tab

Source Code Parser	Defines which SNIFF+ parser is used for extracting symbol information from files of the File Type selected in the L . If this field is empty, the file is not parsed at all. SNIFF+ accommodates many predefined parsers. Please contact SNIFF+ Support for information on predefined parsers and incorporating your own parser.
Icon for File Type	Specifies the icon that should be associated with the file type. It is shown whenever the file is shown in SNIFF+. SNIFF+ comes with a set of icons for most of the predefined types. If the directory specification is a relative path, it is relative to the project directory. An absolute path can also be specified.
Add/Remove Automatically to/from Project	<p>If you want SNIFF+ to add/remove files of a particular file type automatically to/from a project, checkmark Add/Remove Automatically to/from Project.</p> <p>We suggest that you checkmark Add/Remove Automatically to/from Project for “derived files” (e.g., Make Support files). On the other hand, you should clear the Add/Remove Automatically to/from Project checkbox for files that you share with other team members (e.g., source files). This attribute simply allows SNIFF+ to add/remove files to/from a project. It does not modify files outside the SNIFF+ environment in any way.</p> <p>Note that you cannot add/remove files in SNIFF+ when Add/Remove Automatically to/from Project is checkmarked. By checkmarking it, you have delegated the task of adding/removing these files to SNIFF+. If SNIFF+ cannot find files (when Add/Remove Automatically to/from Project is checkmarked) during the opening or updating of a project, you will not see these files in your project. Furthermore, if you delete a file outside of SNIFF+ and then select it in the Project Editor, an Alert dialog appears and the file is no longer listed in the Project Editor.</p>
Generated in Object Directory	Only applicable for IDL. The files that are automatically added/removed are loaded from the Object Directory of the current platform (see also Platform view — page 157) when a project is opened or reloaded.

Group Project Attributes

You can modify the attributes of multiple projects in the Project Tree at the same time by checkmarking them and then choosing **Project > Attributes of Checkmarked Projects....** The Group Project Attributes dialog appears. The views in the Group Project Attributes dialog are the same as those in the Project Attributes dialog, except for a few additional features. These appear in all views and are described below using the General view.



Setting attributes for multiple projects

To set attributes for multiple projects:

1. Select a project in the Project List.

The attributes of the highlighted project are now shown in the various views.

Note

Attributes of frozen projects or of projects with a read-only project description file (PDF) cannot be modified.

2. Set the individual attributes as you would when setting attributes for single projects.
3. Select the check box to the right of the individual attribute to make the attribute globally effective for all projects in the Project List.

4. Press the **Set for All** button and then **OK**.

The changes made to the project attributes will immediately take effect.

Navigating in the Project List

- You can navigate in the Project List by using the **Prev** and **Next** buttons. These buttons display the attributes of the previous/next project in the Project List.

Further features in the File Types view

Merge Options: Replace Add Delete

Checkbox	Description
Replace	Applies current settings to the same file type in all projects in the Project List.
Add/Delete	Adds/Deletes current file type to/from all projects (including current one) in the Project List.

For information about setting file types attributes for multiple projects, please refer to [User's Guide — To set File Types attributes — page 143](#)

Project Editor

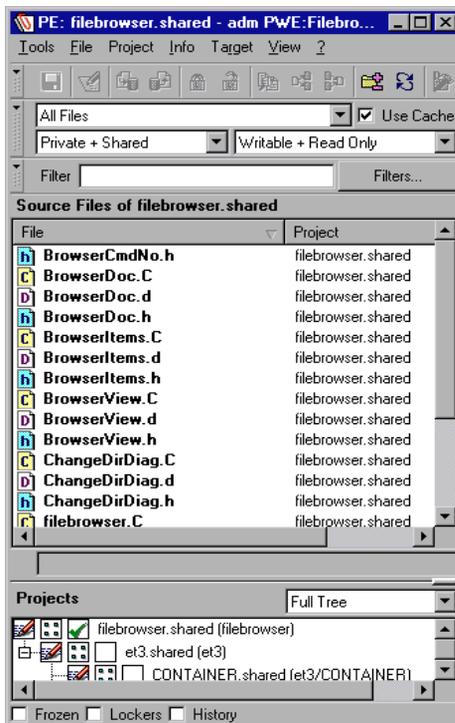
Introduction

The Project Editor is used to edit and browse project-specific information, including

- project attributes
- subprojects
- files
- version control and locking information

When you create new projects, a Project Editor is automatically opened.

You can open a Project Editor by choosing **Project Editor** from the Tools menu of any tool.



Quick Reference

Icons in the File List

Icons are used to represent file types. Note that the following is only a selection of some default file types, you can add your own file types and/or associate files with different icons.

Icon	File Type
	C/C++
	Header
	Java
	Fortran
	IDL
	Project Description (PDF)
	Makefile
	Other
...	etc.

Typeface in the File List

Entry typeface	File is
Bold	writable in current working environment
Non-bold	read-only in current working environment
<i>Italics</i>	located in a shared working environment

Mouse clicks in the File List

- **Double-click** on a file to open it in the associated editor.
- **<Shift>-click** selects all items from a previously selected item to the currently selected file.
- **<Ctrl>-click** adds/removes current selection to/from an existing selection.

Icons in the Project Tree

The icons in the Project Tree indicate the following about the projects:

Icon	Status of the project
	The project is writable; the Project Description File (PDF) is writable and the files of the project may be modified if they are writable.
	The project is frozen; neither the Project Description File (PDF) nor the files of the project may be modified (even if they are writable). Typically, such projects are libraries.
	The Project Description File (PDF) is read-only, but files of the project may be modified if they are writable.
	The project (attributes or structure) has been modified, but its Project Description File (PDF) hasn't been saved yet.
	The object files of this project will be linked directly to the targets of its super-project.
	Do not link the object files of this project directly to the targets of its superproject.

Typeface in the Project Tree

The typeface in the Project Tree indicates the following about the project

Entry typeface	Description
Normal	Symbol information is available for the project.
<i>Italics</i>	No symbol information is loaded for the project.

Mouse clicks in the Project Tree

- **Double-click** on a project to open the Project attributes dialog for the selected project.
- **<Ctrl>click** on a project (not on the checkbox) to display only files from that project and hides all others

Icons in the History window

The History window opens when you select the History check box at the bottom of the tool. Icons are described under [History window — page 204](#)

Basic components

File List

Icons, typeface and mouse shortcuts are described under [Quick Reference — page 190](#).

The File List shows the files belonging to the projects that are checkmarked in the Project Tree. The list can be further constrained by various filters (all located above the list).

You can filter the File List according to:

- status — modified? locked? own? (this information can be cached).
- location — private/shared working environments?
- permissions — read/write?
- file types
- regular expressions

See also [Filters — page 193](#)

Right-click Context menu in the File List

Apart from including frequently used commands, the right-click **Context menu** is configurable. To find out how to configure the commands, please refer to the Preferences on [page 153](#). Note that the available commands in the menu are the same for all files of a particular file type.

File Info Line

The File Info Line, between the File List and the Project Tree, uses an icon to show read/write/modified status of a selected file, followed by the name of the project the file belongs to. The icons are the same as those used in the Project Tree and are described under [Quick Reference — page 190](#). Note that the “modified” status can only be shown if the **History** check box at the bottom of the tool is selected.

Project Tree

Icons, typeface and mouse shortcuts are described under [Quick Reference — page 190](#).

The Project Tree shows the hierarchical project structure.

The nodes in the Project Tree can be clicked to expand or collapse them. An expanded node is indicated by a “-” sign, a collapsed node by a “+” sign.

The icons in the Project Tree show whether a project is writable and whether its objects should be linked to its superproject’s targets.

The attributes displayed in the Project Tree can be edited in the Project Attributes dialog, for more information please refer to [Project Attributes — page 163](#).

A checkmark next to a project’s name means that the files of the project are shown in the File List. In SNIFF+, a project with a checkmark next to it is referred to as a *checkmarked project*.

Note that the same project can be a subproject of more than one project.

The Project Tree drop-down

This drop-down, above the Project Tree, is used to organize the view in Project Tree:

Entry	Description
Full Tree	The full hierarchical structure of the Tree is shown.
No Duplicates	Duplicates are hidden (the same project can be a sub-project of more than one project).
Sorted	The Projects are sorted by path names (case sensitive).

Filters

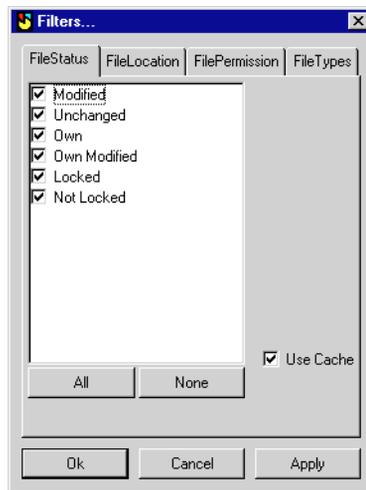
A number of filtering controls are provided at the top of the tool. Individual selections are possible in the various drop-downs, multiple combinations of filters can be selected using the Filters dialog.

Filters... Button

The **Filters...** button opens the Filters dialog, where you can select multiple combinations of filters.

The Filters Dialog

- The **Apply** button applies the selected filters and leaves the dialog open.
- The **Ok** button applies the selected filters and closes the dialog.



If multiple combinations are selected, the corresponding drop-downs on the tool show the entry, **Filtered...** Selecting **Filtered...** in a drop-down opens the Filters dialog. The Project Editor's Filters dialog has four tabs.

File Status tab

The entries correspond to those described under [File Status drop-down — page 194](#).

File Location tab

The entries correspond to those described under [File Location drop-down — page 195](#).

File Permission tab

The entries correspond to those described under [File Permission drop-down — page 195](#).

File Types tab

The entries depend on the file types included in the project. Use this tab to display any multiple combination of file types included in the project.

File Status drop-down

The **File Status** drop-down filters the File List according to locking and version information provided by your version control system. This information can be cached using the **Use Cache** check box (see [Use Cache check box — page 195](#)).

Entry	Description
All Files	Displays all files in the File List regardless of the file status.
Modified	Opens a Version dialog prompting for a configuration name; only those files that differ from the entered configuration are then displayed.
Unchanged	Opens a Version dialog prompting for a configuration name; only those files that are unchanged compared to the entered configuration are then displayed.
Own	Displays only those files that are locked by you.
Own Modified	Opens a Version dialog prompting for a configuration name; only those files that are locked by you and different compared to the entered configuration are then displayed.
Locked	Displays only locked files in the File List.
Not Locked	Displays only unlocked files in the File List.

Entry	Description
Filtered...	Multiple entries are selected in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.

Use Cache check box

Caution: Use this check box with care!

If the check box is selected, file status information provided by your version control system is cached and therefore more quickly accessed. This improves performance when you change filters in the [File Status drop-down — page 194](#). Versioning commands issued within SNIFF+ by yourself will update the cached information.

However, if **other** team members e.g. check out/in files, this will **not** be reflected if the Use Cache check box is selected. Repository changes made **outside** of SNIFF+ are also **not** registered. Information may therefore be inconsistent.

If you need reliable, up-to-date Repository information, clear this check box.

File Location drop-down

Use this drop-down to filter for files located in

- the Private Working Environment you are working in or
- Shared Working Environment(s) or
- Private and Shared Working Environments.

File Permission drop-down

Use this drop-down to filter for

- writable files or
- read-only files or
- both read-only files and writable files

Filter field

Enter a regular expression here and hit <Return> to filter accordingly.

For more information, please refer to [Regular expression filters — page 11](#).

Status line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Lockers check box

Selecting the Lockers check box adds a new column to the File List. The following locking information of currently displayed files is shown:

- the version control tool used
- the owner of the lock and
- the version number of the locked files

History check box

Selecting the **History** check box opens the History window. Please refer to [History window — page 204](#) for details.

Menus

File menu

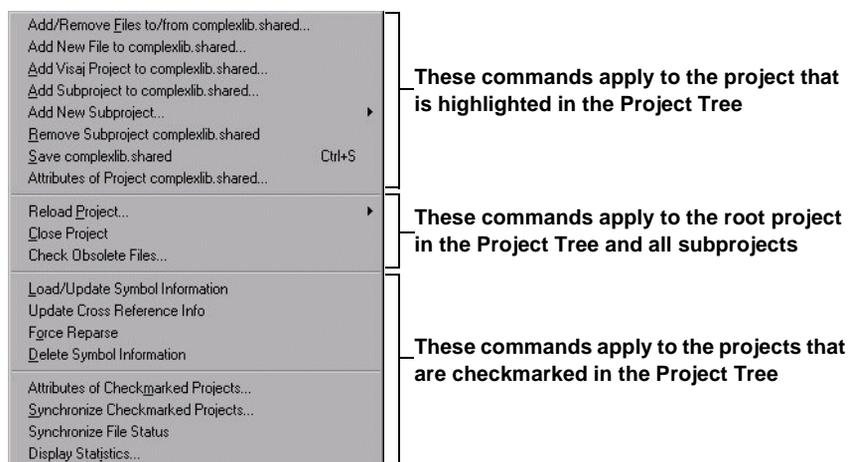
The Project Editor's **File** menu serves to issue commands for the currently selected file(s) of the File List.

File menu command	Description
Check Out...	Opens a Check Out dialog for checking out selected file(s).
Check In...	Opens a Check In dialog for checking in selected file(s).
Lock...	Opens a Lock dialog for locking selected file(s).
Unlock...	Opens an Unlock dialog for unlocking selected file(s).
Delete Version...	This command is enabled when the History window is open. Opens a Delete Version dialog for deleting a version of the selected file(s). If a version is selected in the Version Tree, this version is filled in as the default value of the dialog's Version field. If no version is selected in the Version Tree, SNIFF+ uses the Default Configuration of the working environment in which you opened the project. For more information, please refer to User's Guide — Default Configuration — page 150 .
Replace Description	This command is enabled when the History window is open. Replaces the description of the selected file(s) in the underlying version control tool.
Replace Comment...	This command is enabled when the History window is open. Opens a Replace Comment dialog for changing the comment of a specific version of the selected file.
Select All	Selects all entries in the File List.

File menu command	Description
Synchronize File...	<p>Synchronizes the currently selected files with the repository status. A Files Compared To dialog appears, where you can enter a configuration name. If Ok is pressed in the dialog, all selected working files in the File List are compared with the entered configuration.</p> <p>In shared working environments, out-of-date read-only files are replaced with up-to-date versions from the repository.</p> <p>In private working environments, out-of-date read-only files are replaced with a view to files in the corresponding shared environment.</p> <p>The Synchronize Checkmarked Projects... command of the Project menu does exactly the same at project level. This command uses the update file command of the version and configuration adaptor.</p>
Show Differences...	<p>Opens a dialog where you can enter two versions of a file (or multiple files) selected in the File List. After pressing Ok in the dialog the Diff/Merge tool opens.</p> <p>For a discussion of how to manage versions and configurations with SNIFF+, please refer to User's Guide — Version Control.</p>
Show File <filename>	<p>Opens the file in your preferred editor.</p>

Project menu

You can use the Project menu to issue commands that browse or modify the attributes and structure of the projects in the Project Tree.



Project menu command	Description
Add/Remove Files to/from <i>project</i>...	Only enabled if the Project Description File (PDF) is writable. Opens the Add/Remove Files dialog. See also Add/Remove Files dialog — page 206 .
Add New File to <i>project</i>...	Only enabled if the Project Description File (PDF) is writable. The new file is added to the PDF which is highlighted in the Project Tree.
Add Visaj Project to <i>project</i>...	Opens a dialog where you can enter a name - without any extension - for the Visaj project (the Visaj Project File Type must be included in <i>project's</i> Attributes). A file with the extension <code>.vc1</code> appears in the file list. Double-click on the <code>.vc1</code> file to open the Visaj Class Editor.

Project menu command	Description
Add Subproject to <i>project...</i>	<p>Opens a Subproject File dialog where you can select the Project Description File of the subproject to be added. The project that you selected before choosing the Add Subproject... command becomes the superproject of the selected subproject.</p> <p>Note: For absolute projects: To enhance the transportability of absolute projects, the specification of the subproject file can contain environment variables. Selecting files using the File dialog always writes the absolute file path into the Project Description File. Manually entering the complete subproject file specification in the text field retains environment variables and other shell metacharacters like '~'.</p>
Add Subproject... >	<p>This command allows you to create a new Project using either the New Project defaults or a Project template. The newly created Project is then immediately added to the highlighted project as a subproject.</p>
Remove Subproject <i>project...</i>	<p>Removes the selected subproject from the PDF of its superproject. Note that you cannot delete the Project Description File or any of the files of the subproject by issuing this command.</p>
Save <i>project</i>	<p>Saves the project that is selected in the Project Tree. This menu entry is only enabled if the selected project has been modified. If Make Backup in the Tools node of the Preferences dialog is set, a backup file with the name <i>projectname%</i> is created.</p>
Attributes of <i>project...</i>	<p>Opens the Project Attributes dialog. Please refer to Project Attributes — page 163 for more information.</p>
Reload Project... >	<p>Reloads the selected project from disk. The commands in the submenu are active if the Project Description File (PDF) has changed while the project was open, or when you want to discard all unsaved modifications to a project. If the structure of the selected project has changed, the corresponding files/subprojects are (un)loaded. The project attributes are also updated.</p>

Project menu command	Description
> In Current Working Environment	Reloads the open project in the working environment you are currently working in.
> In Other Working Environment...	Opens a dialog where you can select the working environment to reload the project in.
Close Project	Closes all projects.
Check Obsolete Files	Opens a dialog listing all the files which are in project directories, but which are not used in the projects, that is, they are not recorded in any of the PDFs in the Project structure. You can selectively filter and delete obsolete files.
Update Cross Reference Info	Applies only to C/C++ projects. This allows updating of Cross-Reference information without reparsing symbol information. Subsequent Referred-By queries are then faster. To update Cross Reference information for Java and other languages , use the Force Reparse command.
Load/Update Symbol table	Checks to see whether the symbol information of the files in the File List is current or not. If it is, no action is taken. If it isn't, those files where symbol information is no longer current will be reparsed. You should execute this command only when you have modified project files with tools other than SNIFF+. You can also use this command to incrementally load the Symbol table for projects that have been opened without symbolic information.
Force Reparse	Reparses the files of the projects that are checkmarked in the Project Tree. Reparsing is necessary, for example, when the parser configuration file has been modified. Note that only files of projects whose symbol information is currently loaded into SNIFF+ are reparsed. In large project structures, reparsing can be time consuming. For more information, please refer to User's Guide — Parser configuration file — page 188 .

Project menu command	Description
Delete Symbol Information	Deletes all generated symbol information stored in the “Generate Directories” of the projects that are checkmarked in the Project Tree. Symbol file management is normally fully transparent to the user. This command is necessary only if the symbol files have a wrong modification date (due to a copy or some other reason) or are corrupt; then, when a project is closed, new symbol files will automatically be created.
Attributes of Checkmarked Projects...	Opens the Project Attributes dialog for multiple projects, that is, for the projects checkmarked in the Project Tree. See also Group Project Attributes — page 187
Synchronize Checkmarked Projects	Synchronizes all files in checkmarked projects with the repository files. Note that the checkmarked projects are reloaded in the current working environment after the completion of the command. See also Synchronize File... — page 198 .
Synchronize File Status	Updates the File List display of the file read/write permissions of all files in the projects that are checkmarked in the Project Tree. If the files have changed on disk, they are reparsed and symbol information is brought up to date.
Display Statistics	Opens a Display Statistics dialog. In this dialog, you can select the type of statistics that you want to see - either File Type, Symbol table, or both. Statistics are displayed for each checkmarked project in the Project Tree. When you press the Ok button, a Statistics dialog appears. For more information, see Statistics dialog — page 208 .

Target menu

Please refer to [Target menu — page 19](#).

Info menu

The Info menu in the Project Editor is a simplified version of the Info menu that is available in other tools. For a description of the commands below, please refer to [Info menu — page 20](#).

Retrieve BrowserView.C From A ll Projects	
Retrieve BrowserView.C (using Retriever settings)	Ctrl+2
Find Symbols M atching BrowserView	Ctrl+1
Find Symbols C ontaining BrowserView	
BrowserView.C Includes	
BrowserView.C Is Included-By	

View menu

Please refer to [View menu — page 23](#).

Help (?) menu

Please refer to [Help \(?\) menu — page 23](#).

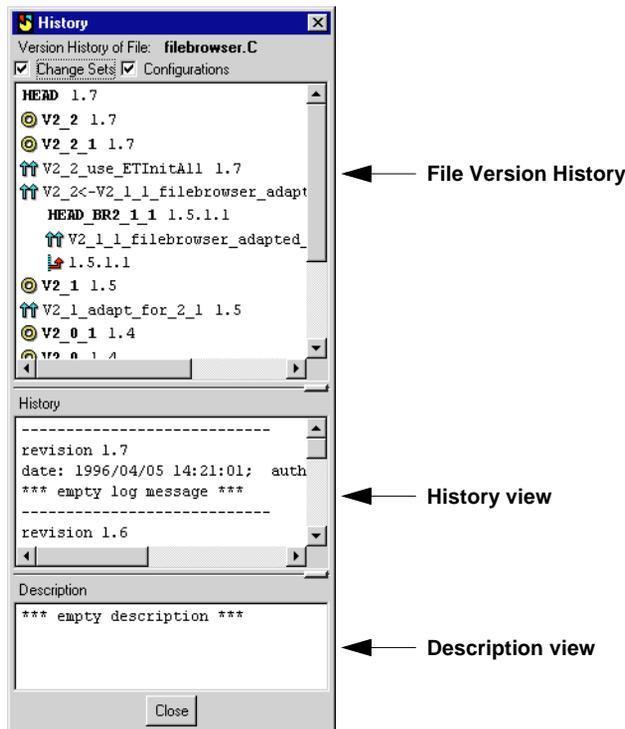
History window

Icons in the History window

The History window opens when you select the History check box at the bottom of the Project Editor.

Symbol	Description
	a single unnamed file version
	a new file version without a change
	file version is the first version on a new branch
	file version is part of a change set
	file version is part of a configuration

You can use the History window to check on file version and configuration information as provided by your version control system.



Filter check boxes

The two check boxes at the top of the window allow you to filter for Change Sets or/and Configurations.

Version Tree

The symbols and text in the Version Tree provide versioning information about the file selected in the Project Editor's File List.

History

Displays the history records (stored in the version control tool) of the selected file. If no entry is selected in the Version Tree, the complete history of the file is displayed. Otherwise, only the history record of the selection in the Version Tree is displayed. The exact contents of the History view depends on the underlying version control tool. The information contains at least the version number, the name of the person who created the version, and when it was created. Also, a comment string is stored with the history record. The comment string of a specific version can be changed by choosing the Project Editor's **File > Replace Comment...** menu command.

Description

Displays the description (stored in the version control tool) of the selected file. The description can be changed by choosing the Project Editor's **File > Replace Description** menu command.

Add/Remove Files dialog

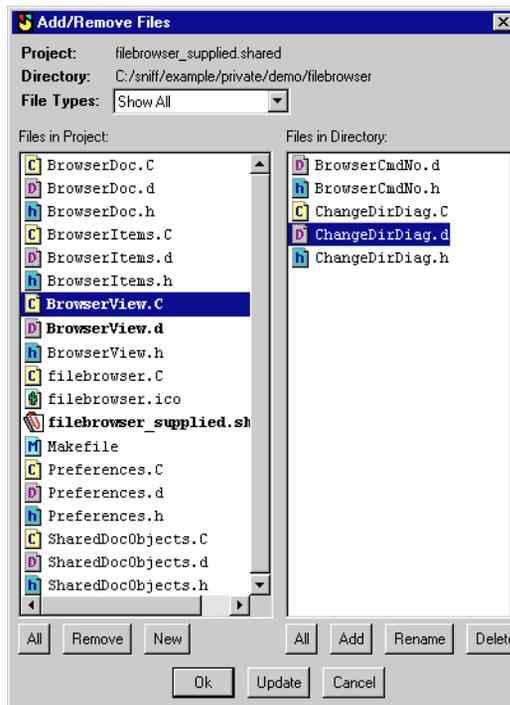
You can add/remove files to/from a project with the Add/Remove Files dialog.

In the Project Editor

To add and remove files from a given project:

1. Make sure the Project Description File you want to add/remove files to/from is writable.
2. In the Project Tree, highlight the relevant project by clicking on it's name.
3. Choose **Project > Add/Remove Files...**

The Add/Remove Files dialog appears. At the top of the dialog, the currently selected project and path are shown.



The File Types Filter

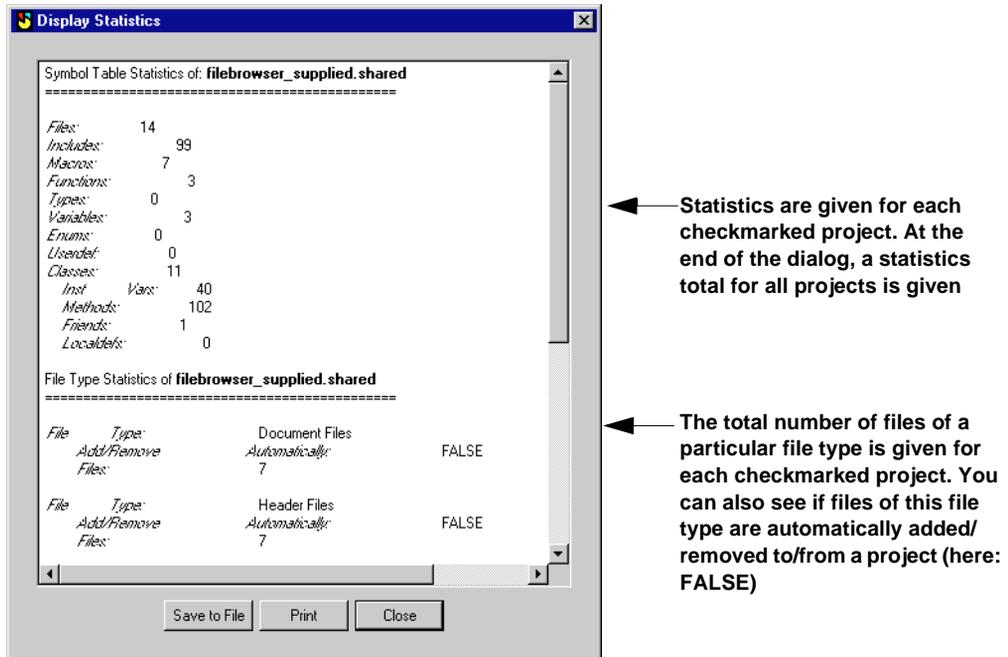
The File Types filter allows you to filter the view for individual, or all, file types included in the project.

Buttons in the Add/Remove Files dialog

Button	Description
All	Selects all elements in the respective File List.
Remove	Removes the selected source file(s) from the current project, but does not delete the file. All symbols of an unloaded file are removed from the project. A file can also be removed by double-clicking.
New	Opens a dialog which prompts for the name of a new file to be created and added to the project. The new filename must match one of the legal file types of this project (it must match one of the signature patterns. See also File Types view — page 182).
Add	Adds the selected source file(s) to the project. To add a file to a project means to parse the file and load the symbolic information. A file can also be added by double-clicking it.
Rename	Pops up a dialog box which prompts for the new name of the file. Only files in the directory can be renamed. To rename the file belonging to a project, unload the file, rename it and load it again. The button is only enabled if a file is selected. Note: This button should not be used to rename archived files, neither for Clearcase nor for any other Version Control tool.
Delete	Deletes the selected file(s) if the file permissions allow it. SNIFF+ asks for confirmation before actually deleting the file(s).
Update	Updates the file lists. Necessary when, for example, a new file is created in the shell.
Ok	Closes the Add/Remove Files dialog and applies the changes.
Cancel	Closes the Add/Remove Files dialog without applying any changes.

Statistics dialog

When you choose Project > Display Statistics..., a dialog appears asking what type of statistics (Symbol table and/or File Type) you would like to see. When you have selected the type of project statistics, the Display Statistics dialog appears and shows the appropriate statistics for the projects checkmarked in the Project Tree.



Statistics are given for each checkmarked project in the Project Editor. In the Display Statistics dialog, you first see statistics for the different types of the Symbol table information (number of files in the project, number of includes, etc.).

The Symbol table statistics are followed by the File Type statistics for each checkmarked project. You can also see whether SNIFF+ automatically adds/removes files of a particular file type in the Add/Remove Automatically field. A Symbol table Statistics total and a File Type Statistics total for all checkmarked projects are then given at the bottom of the Display Statistics dialog. Please refer to Preferences, [File Types view — page 182](#) for a description of the Add/Remove Automatically attribute of a file type.

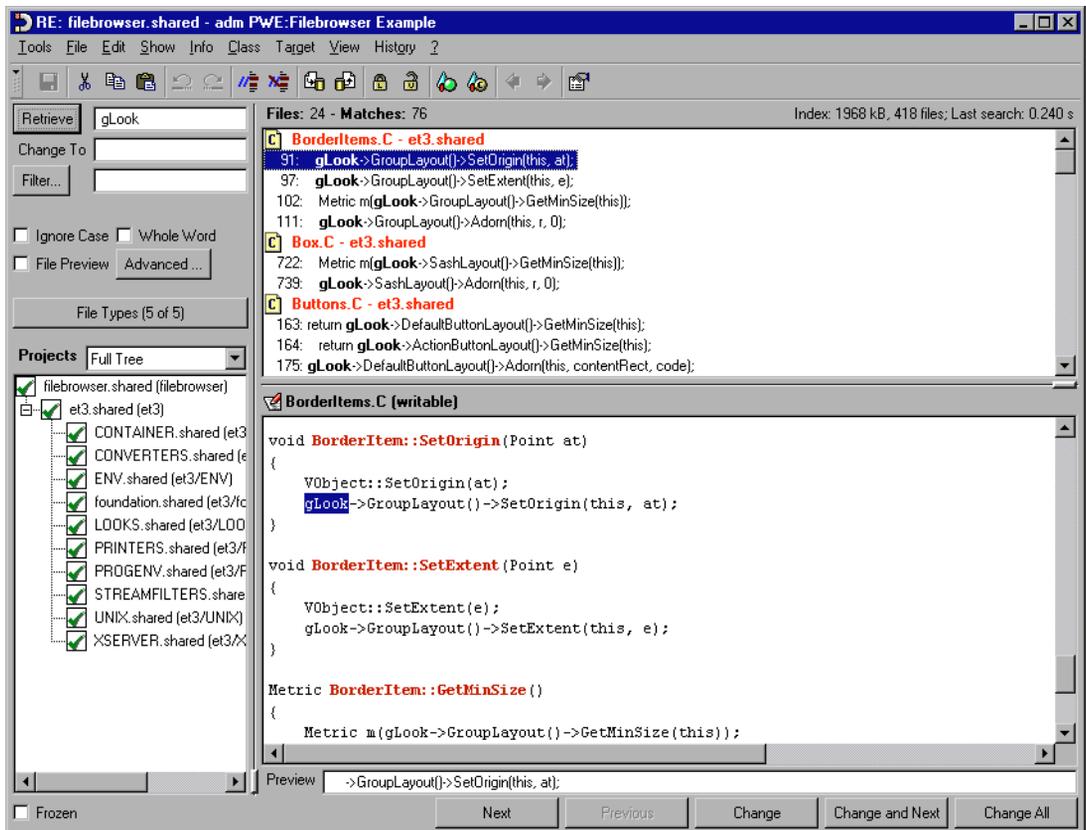
Retriever

Introduction

The Retriever is opened with the appropriate **Info > Retrieve** menu commands available in most tools, or by choosing **Tools > Retriever** from the menu of any tool.

The Retriever is a global textual retrieve-and-replace tool, whereby regular expression filters can be used for complex retrievals and modifications. For an introduction to regular expression syntax, please refer to [Regular Expressions in SNIFF+ — page 307](#).

Queries can be constrained in terms of projects and file types, and filtered using regular expressions. Inter-file navigation is seamless, and modifications can be globally applied.



Quick Reference

Regular expressions

Please refer to [Regular Expressions in SNIFF+ — page 307](#).

Icons in the Files — Matches List

These icons are used to represent the following default file types (note that you can add your own file types and/or associate file types with different icons):

Icon	File Type
	C/C++
	Header
	Java
	Fortran
	IDL
	Project Description (PDF)
	Makefile
	Other

Typeface in the Files — Matches List

Typeface	File is
Bold	writable in current working environment
Non-bold	read-only in current working environment
<i>Italics</i>	located in a shared working environment

Mouse clicks in the Files — Matches List

- **Click** on a text match to position the integrated Source Editor to the match.
- **Click** on a file name to position the integrated Source Editor to line one of the file.
- **Double-click** on a matched line to open the corresponding file in a new Source Editor positioned at the match.
- **Double-click** on a file name to open the file in a new Source Editor positioned at line one.

Indexing and caching

To speed up retrievals in large project structures, the Retriever allows a combination of word indexing and memory caching.

Indexing

Files and words (case-insensitive) are indexed, and therefore simple string queries are a lot faster. Index creation (true by default) is set in the Preferences [Retriever view — page 137](#).

Whether the index and associated features are used for individual queries can be toggled in the [Advanced Retriever Options dialog — page 217](#).

See also [Retriever index files — page 321](#) for more information about the generated index files.

Caching

Files can also be cached in memory after the first retrieval. This will further accelerate subsequent queries, especially also if you query a succession of complex regular expressions that cannot be easily mapped to indexed words. Or also something as simple as “i”, which is mapped as a substring of a whole lot of words.

Caching can be toggled in the [Advanced Retriever Options dialog — page 217](#).

Basic components

Retrieve button

The **Retrieve** button starts a query based on the entries in the **Retrieve** field and the **Filter** field. Note that you have to press this button to trigger a re-query after you change project selections in the Project Tree, or after changing selected File Types.

Queries can also be triggered either from the **Info** menu, or by pressing <Return> in the **Retrieve** field. Filtered queries can also be triggered from the [Find and Replace Filters dialog — page 218](#).

Note

Because the Retriever is independent of symbol information, it is the only SNIFF+ tool that is not updated after changes to the source code. To retrieve from modified files, you have requery.

Retrieve field

A string or a regular expression can be entered directly into the **Retrieve** field. The query starts when <Return> or the **Retrieve** button is pressed. See also [Regular Expressions in SNIFF+ — page 307](#)

Change To field

This field is used for entering modifications, that is, a replacement string which is subject to the modifications imposed by a combined regular expression filter/modifier in the **Filter** field. A preview of the line as modified according to the regular expression evaluation is supplied in the **Preview** field. Changes only take effect after one of the modification control buttons are pressed. Note that if the **Change To** field is empty, and one of the modification control buttons is pressed, the matched expression is replaced by an empty string, i.e. deleted.

For details about modification control buttons, please refer to [Modification control buttons — page 215](#).

Filter... button

The **Filter...** button resets the current filter and opens the Find and Replace Filters dialog. This dialog is used for creating, selecting, and maintaining regular expression filters, as well as for triggering filtered queries. See also [Find and Replace Filters dialog — page 218](#)

Filter field

You can enter regular expressions directly in this field. The string in the Retrieve field can be referenced as “%s”. For more complex, or pre-defined, regular expressions that you might also want to save for later use, the **Filter** field can be filled using the Find and Replace Filters dialog. To use the dialog, press the **Filter...** button.

Check boxes

If you change the settings in these check-boxes, you have to press **Retrieve** to requery using the new settings.

Ignore Case	Toggles case-sensitivity in the query.
Whole Word	Toggles whether only whole words are matched. Note that if you use an alternation, the alternatives must be grouped using <code>\(... \)</code> , e.g. <code>\(FirstWord\ SecondWord\)</code>
Preview Files	Lists indexed files in the checkmarked projects that contain the queried string (also as a substring). The displayed file list is, however, derived from a case-insensitive, word-based index list, and all filters are ignored. The number of files where matches are actually found will thus generally be a subset of the Preview. This check box is only enabled if the Create Index check box is selected (default) in the Preferences Retriever view — page 137 .

Advanced... button

This button opens the [Advanced Retriever Options dialog — page 217](#). The button is only enabled if the **Create Index** check box is selected (default) in the Preferences [Retriever view — page 137](#).

File Types Button

Press this button to open the File Types dialog, where you can select the file types you want to query and/or make changes in. Note that you have to press the **Retrieve** button to requery again after changing the file types selection.

Not all file types are selected in the dialog by default. It doesn't make sense, for example, to include image files in text-based queries. The fewer file types you select, the quicker the retrieval.

By default, the following file types are **not** selected: Image, Object, Makefiles, Generated, Project Description (PDF) and Documentation.

Project Tree

Changing the Project Tree settings affects only subsequent (re-)queries.

The settings in the Project Tree determine the scope of the search (exception: [Search All Indexed Files — page 217](#)). Only those files that are part of checkmarked projects are queried. The check boxes can be manipulated directly with the mouse or by using the right-click context menu.

Files — Matches List

The number of files and matches found is shown at the top of the list, as well as memory cache size (if used), the size and number of indexed files, and the time taken for the last search.

The list of files and matches shows the result of the latest query (subject to the various applied filters and constraints). The list shows the names of files and the projects they belong to in red, each followed by a list of text lines where matches were found within the file. Each match is preceded by the line number within the file.

Integrated Source Editor

The Retriever's integrated Source Editor is fully functional. For information on menus and handling, please refer to [Source Editor — page 225](#). Menus specific to the Retriever are described under [Menus — page 216](#).

Preview field

A preview of the line as it would appear after modification according to the contents of the **Change To** field is supplied in the **Preview** field. Changes only take effect after one of the modification control buttons are pressed. See also [Modification control buttons — page 215](#)

Navigation buttons

Navigation is across the full list of files and follows the order in the Files — Matches List. Individual matches can be directly displayed by clicking on a match in the Files — Matches List.

Button	Description
Previous	Shows the previous match in the Files — Matches List in the integrated Source Editor.
Next	Shows the next match in the Files — Matches List in the integrated Source Editor.

Modification control buttons

Button	Description
Change	Modifies the selected item according to the entries in the Change To field and the Filter field.
Change and Next	After modifying the selected item according to the entries in the Change To field and the Filter field, positions to the next match in the Files — Match List.
Change All	Changes all the matches in the Files — Matches List. Before global changes are made you are warned by a dialog, then the file locking information is checked. If not all the files are writable, a dialog opens to display the locking status of the affected files. See also Locking Status dialog — page 221

Undoing changes

Individual changes made using the **Change** and the **Change and Next** buttons can be undone using the right-click **Context Menu** in the integrated Source Editor.

You can undo global changes (**Change All** button) by choosing **Edit > Undo Change All** immediately after a **Change All**.

- Note that **Edit > Undo Change All** restores all files to the status before the last **Change All** command. Any changes already saved to disk, that is, also changes you may have made manually are also discarded. See also [Edit Menu — page 216](#)

Menus

Most of the menus and commands in the Retriever are common to many SNIFF+ tools. Only the menu commands that are unique to the Retriever are described here. For more information, please refer to [Common Menus — page 13](#).

File menu

File menu command	Description
Load File...	Corresponds to the menu command of the same name in the Source Editor (see Common Menus — page 13), except that files loaded via this menu are not stored in the Retriever's History menu.
Select All Files	Selects all files in the Files — Matches List. This is useful for checking the locking status or for multiple check outs.
Check Locking Status	Checks the locking status of files selected in the Files — Matches List and displays the results in the Locking Status dialog. See also Locking Status dialog — page 221 .

Edit Menu

Edit menu command	Description
Undo Change All	Appears after a Change All (button) command has been executed. Restores all affected files to the status immediately prior to the Change All command. All other changes that may have been made in the meantime are also discarded, even if these were saved to disk.
Redo Change All	Appears after an Edit > Undo Change All command has been executed. Redoes the undone global changes.

Show menu

The commands above the separator in the Show menu are not executed in the integrated Source Editor, but re-directed so that a new Source Editor is opened to show the requested information.

The **Next/Previous Match** commands correspond to the buttons described under [Navigation buttons — page 214](#).

The Retriever in “replace only” mode

The retriever is opened in “replace only” mode from the Cross Referencer with the menu command:

Edit > Replace Referencers of *Symbol*

and from the Include Browser with

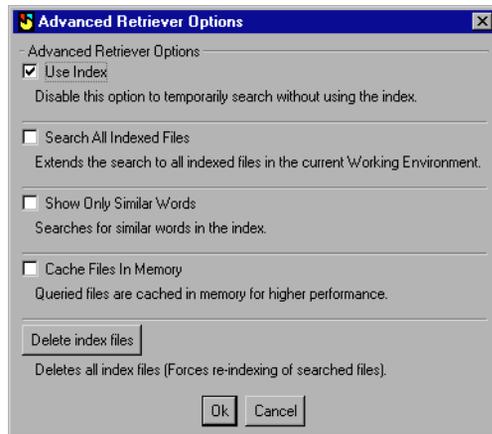
Edit > Replace Include Statements

Here, the query information is already supplied by the requesting tools, and the Retriever provides the functionality for globally changing the references.

For full retrieval functionality, press the **New Query** button.

Advanced Retriever Options dialog

This dialog opens when you press the Retriever’s **Advanced...** button. The button is only enabled if the **Create Index** check box is selected (default) in the Preferences [Retriever view — page 137](#).



Checkboxes

Use Index

Toggles whether the Retriever index is used for queries. See also [Indexing and caching — page 211](#).

Search All Indexed Files

Searches **all** files indexed in the current Working Environment, regardless of which (sub)projects are currently loaded. Note that this includes also files in subprojects that are physically outside the current Working Environment (e.g. libraries) that have been added as subprojects to projects within the Working Environment. See also [Retriever index files — page 321](#) for more information.

Show Only Similar Words

Lists all indexed words containing the queried string (case-insensitive). Regular expressions can be used to restrict the list to substrings that appear at the beginning (e.g. `\bmystring`) or end (e.g. `mystring\b`) of a word.

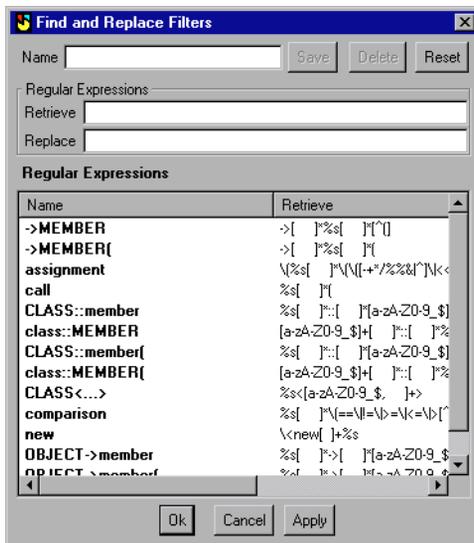
Cache Files In Memory

Toggles whether queried files are cached in memory. See also [Indexing and caching — page 211](#).

Show Only Similar Words

Lists all indexed words containing the queried string (case-insensitive). Regular expressions can be used to restrict the list to substrings that appear at the beginning (e.g. `\bmystring`) or end (e.g. `mystring\b`) of a word.

Find and Replace Filters dialog



The Find and Replace Filters dialog, opened by pressing the **Filter...** button, is used for creating, selecting and maintaining regular expression filters. You can also use the dialog for applying a once-only filter.

For more information on regular expressions in SNIFF+, please refer to [Regular Expressions in SNIFF+ — page 307](#).

Modification buttons

Button	Description
Save	If the name of the filter is unique, a new regular expression filter is added to the Regular Expressions List. If an existing filter has been changed, the changes are saved.
Delete	Removes the selected regular expression filter from the list.
Reset	Resets the current filter to show all matches from the latest retrieval.

Each defined regular expression filter that you save must have at least:

- a unique identification name.

Saving the name on its own is not usually very useful, so it makes sense to define at least

- a retrieve filter, which is applied in the retrieve process.

You may want to use the output of the retrieve filter to define

- a replace filter, which is applied in the replace process.

Name field

To make it easier to quickly find and identify the regular expression, each regular expression is associated with a unique name. Entering the name of an existing expression automatically selects the expression in the list.

Retrieve field

If an existing filter is selected, the “retrieve” part is copied into the **Retrieve** field. The “replace” part (if any) is copied into the **Replace** field. Retrieve filters can also be created and edited in this field. The filter here is applied in a re-query of the current set of matches when you press **Apply** or **Ok**.

Replace field

If an existing filter is selected, the “replace” part is copied into the **Replace** field. Modifiers can also be created and edited in this field. The filter here is applied in replacing matched lines when you press **Apply** or **Ok**.

The Regular Expressions List

Defined filters are listed in the Regular Expressions List. The name of the filter is followed by the corresponding regular expression. If you create a combined retrieve-and-replace filter, the name is followed by the retrieve part of the filter in the next column, and then the replace part of the filter in third column. Items are added to the Regular Expressions List using the **Save** button for new names in the **Name** field, or by pressing **Ok**.

Pre-defined filters

Name	Filters for
call	method, function or procedure calls
assignment	value assignments
comparison	parts of a comparison
new	dynamic allocations
>MEMBER	a dynamic member of any object
>MEMBER(a dynamic member function of any objects
Object>member	an object; all dynamic members are listed
OBJECT>member(an object; only dynamic member functions are listed
class::MEMBER	any static member of any class
class::MEMBER(a static member function of any class
CLASS::member	a class; all static members are listed
CLASS::member(a class; only static member functions are listed
CLASS<...>	a template class
whole word	a whole word

Control buttons

Button	Description
Ok	If the entry in the Name field is unique, a new filter is added to the Regular Expressions List and saved. If an existing filter has been changed, the changes are saved. If anything is entered in the Regular Expressions fields, the filter is applied. The dialog is closed.
Cancel	Discards any changes made in the dialog and closes it.
Apply	Adds a new regular expression filter to the List and applies it in the Retriever. The filter is, however, not saved. The dialog remains open.

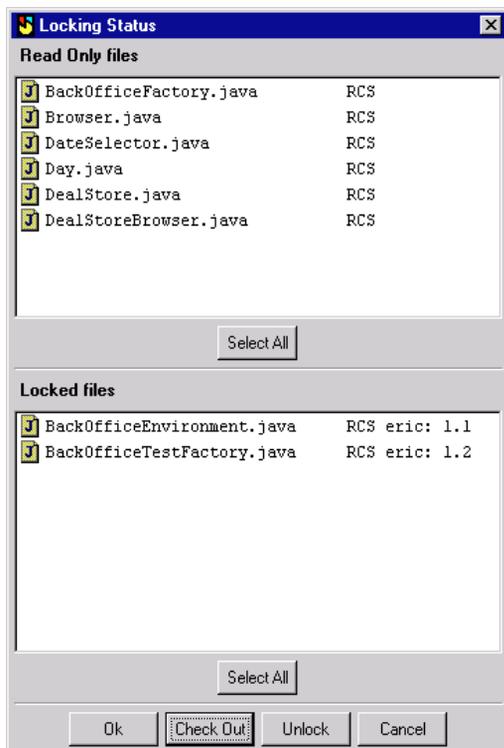
Locking Status dialog

The Locking Status dialog can be opened by choosing **File > Check Locking Status....** The dialog also opens before (batch) modifications are executed if not all selected files are writable.

If files are read-only (e.g., in a Shared Working Environment), or locked by other users, the Locking Status dialog shows these files in two lists. The file name and the versioning tool are displayed in both lists. In the Locked Files List, the name of the user who has checked out the file is also shown, followed by the file version number. Read-only files in shared working environments can be checked out, and files locked by other users can be unlocked or concurrently locked.

Pressing **Ok** closes the dialog and, if it was automatically started after a **Change All** command, the change process continues. If not all files are writable, you are warned by a dialog showing the non-writable files, and you can either continue or cancel the process.

Pressing the **Cancel** button aborts automatic check out and batch modification. No files are affected in any way.



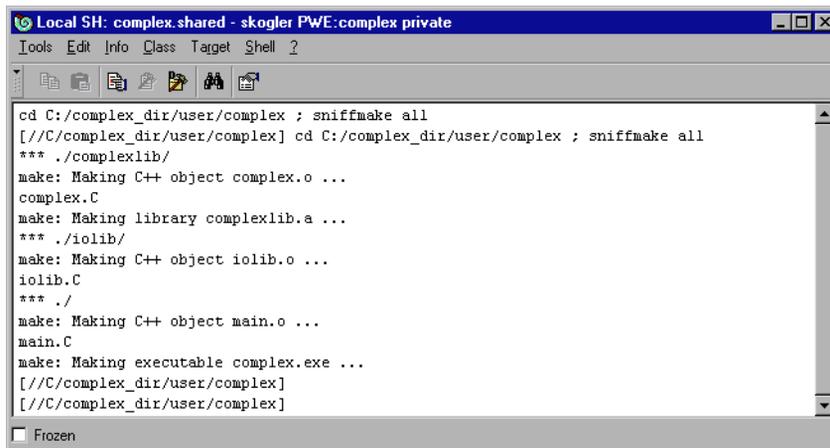
Note

Use the **Check Out** feature carefully when the Locking Status dialog is automatically started after a **Change All** command.

If a checked out file is different from the one originally searched by the Retriever, the result may be wrong. In particular, **do not** use this feature when Keyword Substitution for RCS and SCCS is used in your source files. Keyword Substitution always changes a file while checking it out and therefore will lead to wrong replace results.

Introduction

The Shell is a front-end to the regular Unix command line interface, the DOS command set is also supported. It can be used for system-level manipulations, and it is used by SNIFF+ to issue *make* commands. Furthermore, it serves to select an error message and to trigger the marking of the corresponding source code with the **Edit > Show Error** command.



```

Local SH: complex.shared - skogler PWE: complex private
Tools Edit Info Class Target Shell ?
cd C:/complex_dir/user/complex ; sniffmake all
[//C/complex_dir/user/complex] cd C:/complex_dir/user/complex ; sniffmake all
*** ./complexlib/
make: Making C++ object complex.o ...
complex.C
make: Making library complexlib.a ...
*** ./iolib/
make: Making C++ object iolib.o ...
iolib.C
*** ./
make: Making C++ object main.o ...
main.C
make: Making executable complex.exe ...
[//C/complex_dir/user/complex]
[//C/complex_dir/user/complex]
 Frozen
  
```

Menus

Edit menu

The **Edit** menu contains the usual **Cut/Copy/Paste/Find** commands, plus a **Clear** and a **Show Error** command.

Edit menu command	Description
Clear	Clears the complete Shell buffer.
Show Error	Filters the line containing the cursor. If it understands the error message format, it opens a Source Editor and displays the corresponding source code. The section Error formats — page 288 explains how to extend the list of understood message formats.

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Please refer to [Class menu — page 22](#).

Target menu

Please refer to [Target menu — page 19](#).

Shell menu

The following commands are available in the **Shell** menu:

Shell menu command	Description
Reconnect	Closes the connection to the current shell and connects to a new shell.
Auto Reveal On/Off	Turns the auto-reveal feature on and off. If auto-reveal is on and input is typed or sent from a process, the Shell automatically scrolls to reveal the new text (this is the default).

Introduction

SNiFF+ offers several possibilities for editing source code:

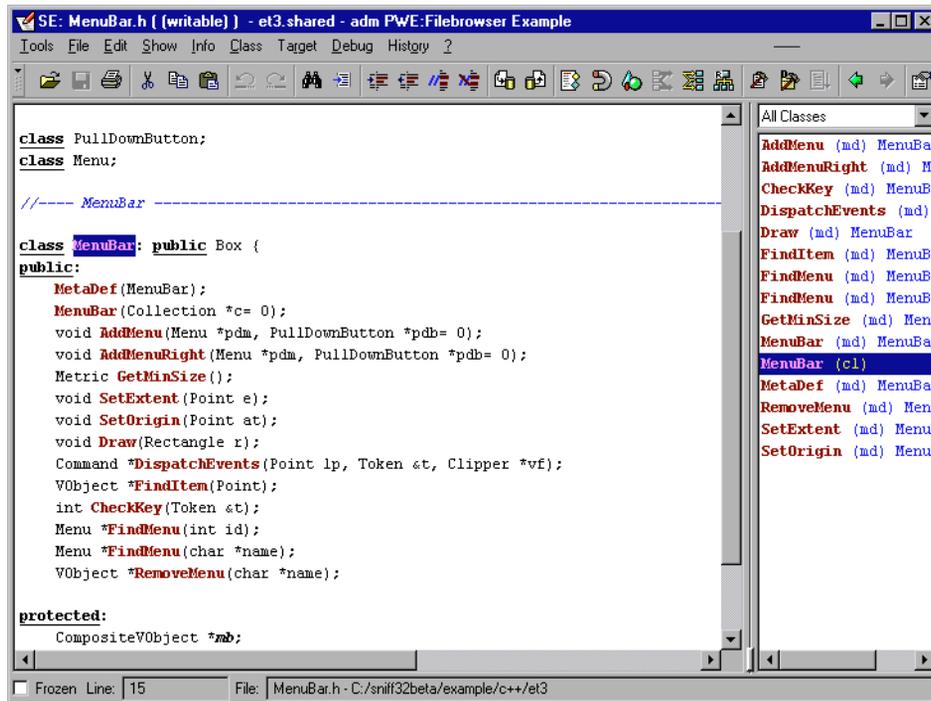
- SNiFF+'s own integrated Source Editor
- an interface to several third party editors (Emacs, vim, Codewright, MS Developer Studio). Please refer to the *User's Guide* for details.

This section describes how to work with the integrated Source Editor.

SNiFF+'s Source Editor consists of a multi-style text Source Editor and a list of classes, methods, and functions defined in the loaded file. The Source Editor understands C/C++, Java, Fortran, CORBA IDL, Ada and Python, and supports customizable syntax highlighting. In addition to the normal editing functionality, the Source Editor offers the following features:

- **Quick symbol navigation**—Many symbol navigation facilities are provided, for example, quick navigation to the symbols in a file via a Symbol List, switching between declaration and implementation of a symbol, navigating in the inheritance hierarchy, triggering all kinds of other browsing tools, and jumping directly to cross-references and retrieved strings.
- **Multiple undo/redo levels beyond file save**—This means you can go back to the original state of a file even if you have saved the file several times in between.
- **Wide range of keyboard shortcuts**—The Source Editor offers many keyboard shortcuts for fast cursor navigation and text manipulation. Most of the commands are accessible via Emacs-like shortcuts.
- **Word completion**—Words that already occur in the file can be automatically completed.
- **History**—The Source Editor remembers a history of files and locations you have visited during editing and browsing.
- **Customizability**—Colors, fonts and other attributes of the Source Editor can be customized in your Preferences. See also [Preferences — page 123](#)

The following illustration shows the SNIFF+ integrated Source Editor:



Quick Reference

- [File status indicators — page 226](#)
- [Selecting text — page 227](#)
- [Word completion — page 227](#)
- [Matching parentheses and quotes — page 227](#)
- [Copying/moving text — page 228](#)
- [Keyboard shortcuts — page 228](#)
- [Keystroke macros — page 229](#)

File status indicators

- On **Windows**, the file status (writable / read only / modified) is indicated in the title bar.

- On **Unix**, the icon of the **Tools** menu indicates the File Status:

Icon	File status
	writable
	read-only
	modified and not yet saved

When a file is saved, the symbol information of the file is extracted anew, the text is reformatted, and the Symbol List is updated.

Shortcuts

On **Windows**, keyboard shortcuts are generally native Windows like.

Selecting text

There are three ways to select text:

- to select a sequence of characters, **single-click** a character and drag the mouse
- to select a sequence of words, **double-click** on a word and drag the mouse
- to select a sequence of lines, **triple-click** on a line and drag the mouse

Word completion

<**Shift+tab**> completes a word if the word already occurs in the loaded file.

Matching parentheses and quotes

Double-clicking close to any of the following language elements causes the Source Editor to mark the code between this item and its matching one:

single quotes -- ' --

double quotes -- " --

parentheses -- (--

brackets -- [--

braces -- { --

Copying/moving text

Drag-and-drop copying/moving

You can drag text from one position to another by selecting it with the mouse and then dragging it. If you want to copy text rather than move it, hold down the <CTRL> key while dragging the text.

- **On Unix**, a **fast copy** command is available. To use it, press the <SHIFT> and <CTRL> keys at the same time and select text. The selected text is then inserted at the current cursor position.

Keyboard shortcuts

The following table shows the Source Editor's cursor navigation keyboard shortcuts

Cursor navigation	Keyboard shortcut	
	Normal	Emacs-like
Word forward	<SHIFT> cursor-right	<ESC> f
Word backward	<SHIFT> cursor-left	<ESC> b
Beginning of line	<CTRL> cursor-left	<CTRL> a
End of line	<CTRL> cursor-right	<CTRL> e
Page down	<PageDown> or <SHIFT> cursor-down	<CTRL> v
Page up	<PageUp> or <SHIFT> cursor-up	<ESC> v
Beginning of file	<Home> or <CTRL> cursor-up	
Bottom of file	<End> or <CTRL> cursor-down	
Start/End cursor text selection	<CTRL> <Space>	

The following table shows the Source Editor's text modification keyboard shortcuts

Text modification	Keyboard shortcut	
	Normal	Emacs-like
Delete left of cursor	<Backspace>	
Delete right of cursor	<Delete>	<CTRL> d
Delete from cursor to end of line		<CTRL> k
Delete word right of cursor	<CTRL><Backspace>	<ESC> d
Delete word left of cursor	<SHIFT><Backspace>	<ESC> <Backspace>
Zap (delete from cursor to right until character)		<ESC> z <i>character</i>
Cut	<Meta> x	<CTRL> x <CTRL> x
Copy	<Meta> c	<CTRL> x <CTRL> c
Paste	<Meta> v	<CTRL> x <CTRL> v
Undo	<Meta> z	<CTRL> /
Redo		<CTRL> ?

Keystroke macros

The SNIFF+ Source Editor can remember keystrokes in a macro. The following table shows how to define and execute a macro:

Macro	Keyboard shortcut
Start macro	<CTRL+x> (
End macro	<CTRL+x>)
Execute macro	<CTRL+x> e

Basic components

Text View

The Text View shows the source text by using different styles to highlight symbols and comments. Many attributes of the view, including fonts and colors for the various symbol types and keywords, or whether or not nonprinting characters are displayed can be customized in your Preferences (see [Source Editor view — page 132](#)).

Available shortcuts are described under [Shortcuts — page 227](#).

Symbol List

The Symbol List is constrained by the **Class** pop-up and shows the list of:

- method declarations (`md`) and implementations (`mi`)
- class declarations (`cl`)
- functions (`f`)
- structures (`st`)

The Source Editor is positioned at the symbol by clicking on an entry in the Symbol List. A deep click (<CTRL>click) on a declaration entry will position the Editor at the implementation and vice versa.

Class drop-down

The **Class** drop-down scopes the Symbol List to either show only symbols of one class or to show all symbols of this file. This feature eases navigation when there is more than one class defined in a file.

Menus

File menu

Please refer to [File menu — page 15](#).

Edit menu

Please refer to [Edit menu — page 16](#).

Show menu

Please refer to [Show menu — page 18](#).

Target menu

Please refer to [Target menu — page 19](#).

Info menu

Please refer to [Info menu — page 20](#)

Class menu

Please refer to [Class menu — page 22](#).

Debug menu

The commands in the **Debug** menu are enabled when the Editor is in debugging mode.

Please refer to [Execution menu — page 78](#), [Print menu — page 79](#) and [Display menu — page 79](#) for a description of the commands in the **Debug** menu.

History menu

Please refer to [History menu — page 23](#).

Debugging mode — extra buttons added to the Source Editor

After the **Debug target** command is issued from the **Target** menu, the Debugger is started and the Source Editor is in debugging mode. In debugging mode the file is read-only and a row of new buttons is added to the Source Editor window.



Button	Description
Run	Runs the application being debugged from scratch.
Cont	Continues interrupted execution.
Step	Single-steps into the next function/method.
Next	Single-steps over the next function/method.
Break In	Sets a break point at the current selection, whereby selection must be a valid function/method name.
Break At	Sets a breakpoint at the current cursor position (linewise).
Clear	Clears the breakpoint in the current line. The cursor must be positioned to a line with a breakpoint.
Print *	Prints the value pointed to by the current selection. The selection must evaluate to a valid pointer.
Print	Prints the value of the current selection. The selection must evaluate to a valid variable.
this	Prints the value of the current object.
Stack	Opens a Stack window and displays the current call stack. See also Callstack tab — page 80 .
Up	Goes one stack frame up in the call hierarchy. A reusable Source Editor is automatically positioned at the source location of the new stack frame.
Down	Goes one stack frame down in the call hierarchy. A reusable Source Editor is automatically positioned at the source location of the new stack frame.

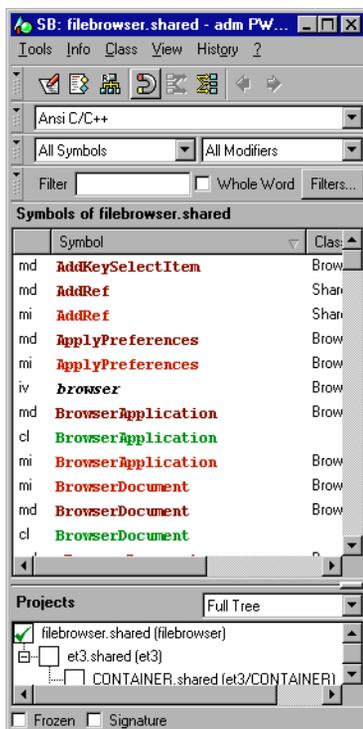
Symbol Browser

Introduction

The Symbol Browser shows the symbols used in your projects. You can filter according to language, symbol types, modifiers and regular expressions.

The content of the list is determined by the **Filter** field and the various filter drop-downs at the top of the tool, as well as the Project Tree at the bottom of the tool. To browse symbols:

- click on a string in an open tool and choose **Info > Find Symbols Matching selection** or **Info > Find Symbols Containing selection**, or
- choose **Tools > Symbol Browser** in any open tool.



Quick Reference

Typeface in the Symbol List

- Typeface in the Symbol List corresponds to text highlighting in the Source Editor (you can set this in your Preferences).

Mouse clicks

- In the Symbol List, **double-click** on a symbol to show it in the Source Editor.
- In the Project Tree, **<CTRL>click** on the name of a project lists members of that project only; members of all other projects are hidden.

Basic components

Symbol List

The Symbol List displays the symbols of the projects checkmarked in the Project Tree, subject to the various filter settings.

Symbols of the same type with the same name are qualified by the name of the file they belong to.

C++ templates are listed as classes. However, the names of templates are identified as templates and are shown with their formal parameters.

Project Tree

The Project Tree shows the project structure including subprojects. Only symbols used in checkmarked projects are shown in the Symbol List.

Filters

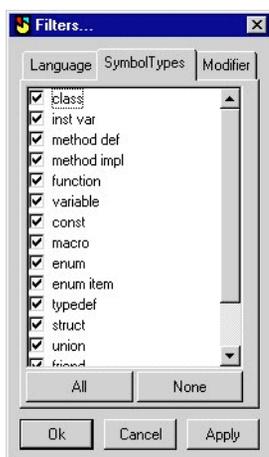
A number of filtering controls are provided at the top of the tool. Individual selections are possible in the various drop-downs, multiple combinations of filters can be selected using the Filters dialog.

Filters Button

The **Filters...** button opens the Filters dialog, where you can select multiple combinations of filters.

The Filters Dialog

- The **Apply** button applies the selected filters and leaves the dialog open.
- The **Ok** button applies the selected filters and closes the dialog.



The Symbol Browser's Filters dialog has three tabs, the elements correspond to the individual entries in the drop-downs on the tool.

If multiple combinations are selected, the corresponding drop-downs on the tool show the entry, **Filtered...**

Selecting **Filtered...** in a drop-down opens the Filters dialog.

Language drop-down

In the **Language** drop-down, you can choose the language whose symbols you want to browse. Only those languages used in your projects are listed

Symbols drop-down

The **Symbols** drop-down specifies the type of symbols shown in the Symbol List.

All Symbols	Shows all symbols.
Filtered...	Means that multiple selections were made in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.
Language-specific symbols	Shows the selected language-specific symbol types.

Modifier drop-down

The **Modifier** drop-down filters the symbol list according to modifiers. Note that this includes also implicit modifiers, e.g., overriding or overloaded methods.

All Modifiers	Shows all symbols, irrespective of modifier.
no modifier	Shows only methods without modifiers.
Filtered...	Means that multiple selections were made in the Filters dialog. Selecting the Filtered... entry itself opens the Filters dialog.
Language-specific modifiers	Shows only symbols modified by the selected modifier.

Filter field

Enter a regular expression here and hit <Return> to filter accordingly.
Please refer to [Regular expression filters — page 11](#) for more information.

Status line

Frozen check box

The **Frozen** check box is described under [Status line — page 12](#).

Signature check box

When the **Signature** check box is enabled, the full signature of each symbol is displayed.

Menus

Info menu

Please refer to [Info menu — page 20](#).

Class menu

Please refer to [Class menu — page 22](#).

View menu

Please refer to [View menu — page 23](#).

History menu

Please refer to [History menu — page 23](#).

Help (?) menu

Please refer to [Help \(?\) menu — page 23](#).

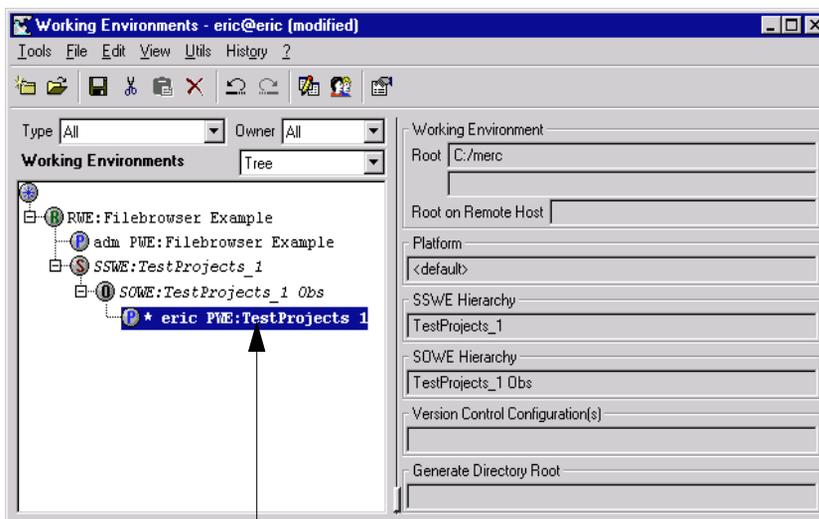
Working Environments

Introduction

To open the Working Environments tool, choose **Tools > Working Environments** from the menu of any tool.

You use the Working Environments tool to create and maintain working environments. You can also use the Working Environments tool to open projects.

If you are not familiar with SNIFF+ Working Environments, please refer to [User's Guide — Working Environments](#) for more information.



The selected Private Working Environment (PWE) is owned by the user eric. The * means that this is the working environment where projects are opened by default.

These fields show information relating to the selected working environment.

Quick Reference

Abbreviations

The following abbreviations are used in the Working Environments tool and in the documentation:

Abbreviation	Working Environment
RWE	Repository
SSWE	Shared Source
<i>SOWE</i>	Shared Object
PWE	Private

Typeface in the Working Environments Tree

The typeface indicates the following about a working environment:

Typeface	Working Environment is
Bold	PWE owned by you
Non-bold	RWE, and PWEs not owned by you
<i>Italics</i>	SOWE or SSWE
Asterisk (*)	default working environment

Mouse clicks in the Working Environments Tree

- **Click** on a working environment to make it the *current working environment*, i.e., the one in which projects will be opened.
- **Click** on a node of the Working Environments Tree to collapse/expand the node.
- **Double-click** on a working environment to invoke the [Open Project dialog — page 26](#).

Basic components

Type drop-down

List entry	Description
All	Shows all Working Environments in the Working EnvironmentsTree.
Repository	Shows only the Repositories in the Working EnvironmentsTree.
Shared Source	Shows all Shared Source Working Environments and all Working Environments that are accessed by an SSWE.
Shared Object	Shows all Shared Object Working Environment and all Working Environments that are accessed by an SOWE.
Private	Shows all Private Working Environments and all Working Environments that are accessed by a PWE.

Owner drop-down

List entry	Description
All	Shows the Working Environments of all team members in the Working EnvironmentsTree.
Own	Shows your own Private Working Environments and all accessed Working Environments.

View drop-down

List entry	Description
Tree	Shows the Working Environments in a hierarchical tree.
Sorted	Shows the Working Environments in alphabetical order.

Working Environments Tree

For a description of abbreviations and typeface, please see [Quick Reference — page 240](#).

The Working Environments Tree shows your development team's working environments. These can be displayed in a hierarchical tree or in alphabetical order, and filtered according to type and ownership.

Each node of the tree refers to one working environment. For example, the node

```
RWE: Filebrowser Example
```

refers to an RWE (Repository). The *name* of this working environment is `Filebrowser Example`.

The working environment currently selected in the Working EnvironmentsTree is the *current working environment*. All shared projects are opened in the current working environment.

If no working environment is selected, projects are opened in the *default working environment*, which is indicated by an asterisk (*) to the left of its name. The menu command **Edit > Set Default** sets the default working environment in the Preferences for you.

Each PWE has an *owner*. By default, all newly set up PWEs are assigned an owner with the log-in name of whoever set them up. This can be changed as described under [Users dialog — page 250](#). If no owner is assigned (i.e. deleted in the Users dialog), PWEs are assigned to a default user named `adm`.

For example, the node: `* harry PWE:MyPlace`

would refer to the PWE (Private Working Environment) named `MyPlace`, which is owned by the user `harry`. You can specify the owner of a PWE with the **Edit > Modify...** menu command. Being the owner of a PWE allows you to filter for your own PWE(s) and the shared environments accessed by your PWE(s). Once you have identified yourself in the Users Dialog, only you and users with administration permissions (i.e. permissions to create/modify hierarchically higher, shared working environments) can modify a PWE owned by you. See also [Users dialog — page 250](#). PWEs that you own, or that are owned by `adm` (nobody), are also shown in the **Working Environments** tab of the Launch Pad.

Modifying Working Environments

You can modify the structure of the Working Environments Tree using drag-and-drop, the appropriate toolbar buttons, or the right-click **Context menu** and the **Edit** menu.

Using drag-and-drop, you get visual feedback (highlighting) if the drop target is permissible.

The **Edit > Modify...** menu command opens the Modify/New Working Environment dialog, where you can modify attributes of individual working environments. See also [Modify/New Working Environment dialog — page 249](#)

If you modify working environments, all the working environments in the tree are locked and remain locked until the modifications have been saved. This prevents other members of your development team from trying to modify the same working environment at the same time.

Note that deleting a working environment deletes the selected working environment and all working environments hierarchically below the selected working environment.

- **On Windows**, you are warned in the title bar that there are unsaved modifications.
- **On Unix**, the **Tools** icon additionally indicates whether you have made any unsaved modifications to the working environments by changing to 

If you try to modify a working environment that has been locked by someone else, a warning message appears.

Note that any changes made to a working environment where a project is already open are only “seen” by the project after it has been closed and re-opened.

Working Environments information

The fields to the right of the Working Environments Tree display information about the working environment selected in the Working EnvironmentsTree. The information in the various fields is automatically filled in and updated after modifications. These fields can be set/modified using the **Edit > Modify...** menu command to open the Modify/New Working Environment dialog. See also [Modify/New Working Environment dialog — page 249](#)

Root fields

If the path to the selected working environment was specified using an environment variable, both this path and the expanded path are shown. If the working environment root was defined without using an environment variable, only this path is shown.

Root on Remote Host field

This field is only needed for remote compiling and debugging in a cross-platform work situation (and if you have a `SNiFF_CROSS` license). Enter the path to the remote working environment root as the remote machine would see it, e.g., if the remote working environment is on Unix, you would not use drive letters. See also [User's Guide — Compiling and Debugging in SNIFF+ — page 195](#).

Platform field

You can specify different platforms (together with debugger adaptors) for each working environment in the Preferences, see [Platform view — page 157](#). Once platforms are defined, these can be selected in the Modify/New Working Environment dialog. See also [Modify/New Working Environment dialog — page 249](#)

SSWE Hierarchy field

The **SSWE Hierarchy** field shows the root directories of SSWE(s) accessed by the selected working environment. If multiple SSWEs are accessed by the selected working environment, a colon (:) is used to separate the SSWEs from each other.

SOWE Hierarchy field

The **SOWE Hierarchy** field shows the root directories of SOWE(s) accessed by the selected working environment. If multiple SOWEs are accessed by the selected working environment, a colon (:) is used to separate the SOWEs from each other.

Version Control Configuration(s) field

Please refer to [User's Guide — Specifying Default Configurations — page 164](#).

Note

To specify the HEAD version of a branch as the default, enter HEAD_<branch_name> in this field.

Generate Directory Root field

In most cases, you would leave the **Generate Directory Root** field empty.

In the **Generate Directory Root** field, you can specify the path of the root directory for the SNIFF+ - generated directories (SNIFF+ stores project-specific information in these directories). If you leave this field blank, these directories are generated directly in the project directories.

This leads to problems when SNIFF+ tries to store the generated directories in project directories for which you don't have write permission.

A situation where this may occur is with library projects, which are generally read-only. You can avoid this problem by entering the SNIFF+ - generated directories path in the **Generate Directory Root** field. The path of the root directory must be relative to the selected working environment's root directory.

Menus

File menu

File menu command	Description
Save	Saves the settings of newly defined or modified working environments.
Reload	Reloads the files that store the working environments data (user permissions, names of working environments etc.). If you modify working environments and choose this command before saving, the changes are discarded.
New Project... >	Opens a submenu with 3 options for creating new projects. For more information please refer to User's Guide — Project Setup Overview — page 53 .
> with Defaults...	Opens the Directory dialog. In the Directory dialog, you select the directory where the source files of the new project are located. Then, an Attributes of a New Project dialog appears. You set the new project's attributes in this dialog, see New Project Options — page 165 . The defaults for new projects are set in the Preferences, see New Project Setup view — page 146 .
> with Template...	Allows you to set the attributes of a new project from a template. The Project Template Dialog opens and you can select a Project Template File (extension <code>.ptmpl</code>) to use as a template. Select a file to open the Project Attributes dialog. You can edit the opened template, have the new project created, and save the edited template under a new name. See also User's Guide — Working with new project templates — page 57 .
> with Wizard...	Starts the Project Setup Wizard, which guides you through project setup. See also User's Guide — SNIFF+ Project Setup Wizard — page 53 .
Open Project...	Opens the Open Project dialog, where you can choose a project to open in the working environment selected in the Working EnvironmentsTree.

Edit menu

You can create, delete or modify working environments with the entries in the **Edit** menu. When you select a working environment from the Working EnvironmentsTree, only permissible commands are enabled. The following commands are available:

Edit menu command	Description
Undo <i>last command</i>	Undoes the last command. You can set the number of undo levels in the Preferences.
Redo <i>last command</i>	Redoes the last command.
Cut	Cuts the selected environment to a buffer.
Paste	Pastes buffer contents to selected position. Enabled only if the buffer is not empty and a valid target is selected.
Delete	Deletes the selected working environment and all working environments hierarchically below the selected working environment.
Modify...	Opens the Modify/New Working Environment dialog — page 249 , where you can modify all the attributes you see in the information fields at the right of the tool. You can also specify the owner of a Private Working Environment.
New Repository...	Opens a dialog where you can define a new Repository (RWE). The command is enabled when you select the root of the Working Environments Tree and if you have the appropriate permissions, which are set in the Users dialog — page 250 .
New Shared Source...	Opens a dialog where you can define an SSWE. To define an SSWE, an RWE must be selected in the Working EnvironmentsTree and you must have the appropriate permissions, which are set in the Users dialog — page 250 .
New Shared Object...	Opens a dialog where you can define an SOWE. To define an SOWE, an SSWE must be selected in the Working EnvironmentsTree and you must have the appropriate permissions, which are set in the Users dialog — page 250 . The new SOWE then accesses the selected SSWE.
New Private...	Opens a dialog where you can define a PWE. To define a PWE, an RWE, SSWE or SOWE must be selected in the Working EnvironmentsTree. The new PWE then accesses the selected working environment if you have the appropriate permissions, which are set in the Users dialog — page 250 .

Edit menu command	Description
Set Default	Sets the working environment selected in the Working EnvironmentsTree to the default working environment. See Working Environments Tree — page 242 for a description of default working environments.

View menu

View menu command	Description
Collapse/Expand	Collapses/expands the selected node in the Working Environments Tree.

Utils menu

Utils menu command	Description
User Permissions	Opens the Users dialog. Users dialog — page 250 , where you can add or remove users and their permissions for working environments.

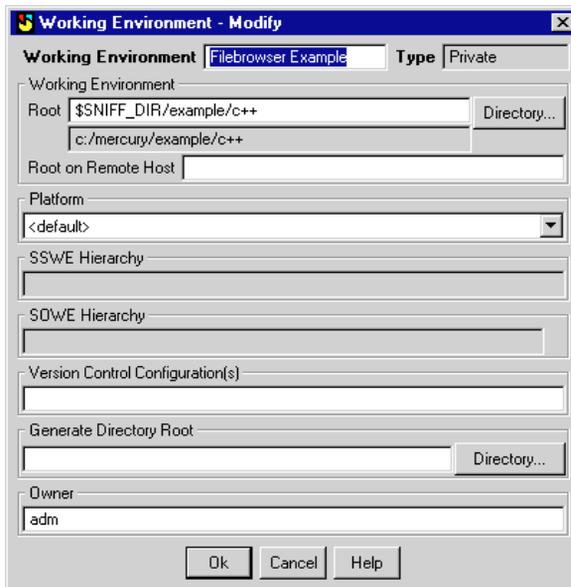
History menu

Working environments in which you have opened projects are automatically added to the history. The number of entries can be set in the Preferences, see [Tools view — page 130](#).

Modify/New Working Environment dialog

The dialog shown below opens when a PWE is selected, and you choose **Edit > Modify...** from the menu.

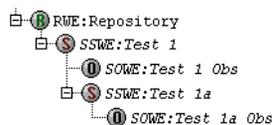
Very similar dialogs open when other working environments are selected, or when you choose one of the **Edit > New Working Environment** commands.



The fields in the dialog correspond to the information fields in the Working Environments tool - see [Working Environments information — page 244](#)

Which fields are writable depends on the type of working environment you want to modify/create.

The dialog for creating/modifying SOWEs has an additional button next to the **SOWE Hierarchy** field. The **SOWE...** button opens a dialog where you can hierarchically arrange SOWEs in complex working environment trees. This is possible where you have hierarchical SSWEs, each accessed by SOWEs, e.g., the following structure:



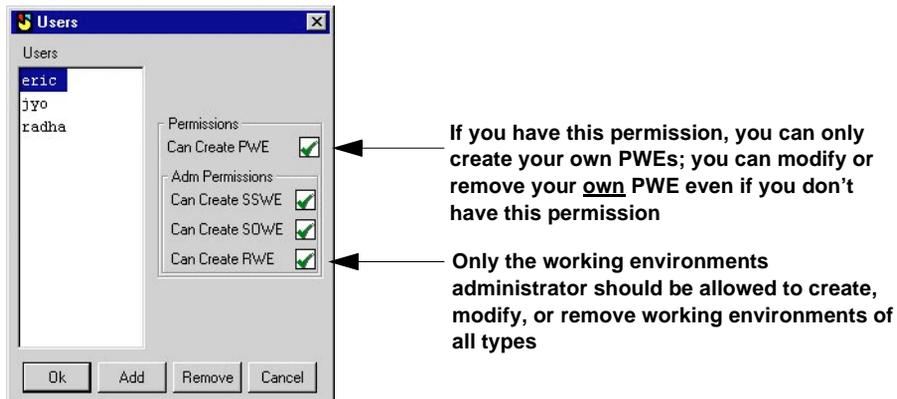
Users dialog

The Users dialog appears when you choose the **Utils > User Permissions...** menu command. The information in this dialog is stored in

`<sniff_installation_dir>/workingenvs/WorkingEnvUser.sniff`.

You can specify a different directory in the Preferences. Generally, only the Working Environments Administrator should have write permissions for this file. The Working Environments Administrator would then create working environments and set permissions for new users. For more information about working environments administration, please refer to the *User's Guide*.

As long as the Users List is empty, anyone can modify/create working environments.



Buttons

Button	Description
Add	Opens a dialog to add a new user to the Users List.
Remove	Removes the selected user from the Users List.

Part III

Advanced Reference



SNiFF+ Executables

Introduction

This chapter describes the SNiFF+ executables that are delivered with the product package and that are of interest to the user. All executables can be found in the binary directory of the SNiFF+ installation (`$SNiFF_DIR/bin`). Platform-dependent executables are links to the shell script `sniff_wrapper`, which determines the current platform with the script `sniff_arch` and calls the corresponding executable in the platform-specific subdirectory.

sniff

`sniff` is the executable of the programming environment.

Synopsis

```
sniff [-s <sniff_session_name>] [-c <license_file>] [-l  
<log_file>] [-v] [<project_file>]
```

Arguments and Options

`<sniff_session_name>` is the *session id* that is associated with a SNiFF+ session (see [page 255](#)).

`<project_file>` is the name of an existing SNiFF+ project description file.

`<log_file>` is the name of the file where SNiFF+ stores logging information. By default the log file is:

On Unix: `$HOME/.sniffrc/sniff.log`

On Windows: `%SNiFF_DIR%/Profiles/<username>/sniff.log`

The default log file name can be set in your Preferences.

`<license_file>` points to the license file to be used. Please refer to the *Installation Guide* for more information on licensing issues.

On Unix, `-v` only prints the release number and date, but does not start SNiFF+. For example:

```
% sniff -v  
SNiFF+ V2.3 Apr 10, 1997
```

On Windows

`-v` does not print the version number and the date to screen, instead it prints the version number and date to standard error.

Environment variables

SNIFF+ needs two environment variables, `SNIFF_DIR` and `PATH`. The environment variables should be set in the `.login` file of the SNIFF+ user. Furthermore, a third variable, `SNIFF_SESSION_ID`, may be set before starting a new SNIFF+ session (see [Multiple simultaneous SNIFF+ sessions — page 255](#)).

SNIFF_DIR

This environment variable can be set on Windows via the Control Panel.

On Unix:

```
setenv SNIFF_DIR <sniff_directory>           (for csh)
SNIFF_DIR=<sniff_directory>; export SNIFF_DIR (for sh or ksh)
```

where `<sniff_directory>` is the root of the directory tree of your SNIFF+ installation.

PATH

This environment variable can be set on Windows via the Control Panel.

On Unix:

```
set path = ($SNIFF_DIR/bin $path)           (for csh)
PATH=$SNIFF_DIR/bin:$PATH; export PATH      (for sh or ksh)
```

As is the case for any X-window application, the `DISPLAY` variable must be set to point to the server where the SNIFF+ windows should appear.

LM_LICENSE_FILE

The `LM_LICENSE_FILE` variable has to point to a valid license file if it is not at its default location as suggested by the *Installation Guide*. For more information please refer to the *Installation Guide* or your FLEXlm documentation. The license file can also be specified with the `-c` command line option of `sniff`. The following setting shows a configuration where the license file is located in the SNIFF+ installation directory:

```
setenv LM_LICENSE_FILE <sniff_directory>/          (for csh)
license.dat
```

```
LM_LICENSE_FILE=<sniff_directory>/license.dat;      (for sh or ksh)
export LM_LICENSE_FILE
```

If you want to use a license administrated by a license server on a remote machine, check for the host name and port number in the `license.dat` file (in the line starting with `SERVER`). Then set the `LM_LICENSE_FILE` variable to `<port number>@<hostname>`. In this way, when you start SNIFF+ on your local machine it will use the license administrated by the license server on the remote machine.

Multiple simultaneous SNIFF+ sessions

You can have multiple SNIFF+ sessions running at the same time. In order for you to start a new SNIFF+ session, type the following on the command line:

```
sniff -s <sniff_session_name>
```

`<sniff_session_name>` is a session id that uniquely identifies each SNIFF+ session. Note that the session id is a string and cannot contain any blank spaces within it.

If you start SNIFF+ without the `-s` option, the value of the `SNIFF_SESSION_ID` environment variable is used by default for the session id (see below). If `SNIFF_SESSION_ID` is not set, `session0` is used for the session id.

SNIFF_SESSION_ID

The `SNIFF_SESSION_ID` environment variable is set to uniquely specify a SNIFF+ session. When you set this variable, the session id of your next SNIFF+ session will be automatically set to the value of this variable. To set the value of `SNIFF_SESSION_ID` environment variable, type the following on the command line:

```
setenv SNIFF_SESSION_ID <sniff_session_name>
```

`<sniff_session_name>` uniquely identifies each SNIFF+ session. It is a string and cannot contain any blank spaces.

SNIFF+ without display (batch mode)

You can start a SNIFF+ session without a display. This is particularly useful on Unix, since you can then run SNIFF+ without an Xserver host during unattended updates of your working environments.

SNIFF+ runs without a display when the `SNIFF_BATCH` environment variable is set to 1. To set the variable:

On Unix:

1. Open a shell in which you will be updating your working environments.
2. In the shell, set the `SNIFF_BATCH` environment variable to 1:

```
setenv SNIFF_BATCH 1
```
3. When updating your working environments, run the appropriate update scripts without the `<Xserver_host>` parameter.

On Windows:

1. Open a Command Prompt in which you will be updating your working environments.
2. In the Command Prompt, set the `SNIFF_BATCH` environment variable to 1:

```
set SNIFF_BATCH=1
```

sniff_arch

`sniff_arch` is a shell script that determines the platform and outputs a string that exactly identifies the platform. All SNIFF+-supported platforms are handled correctly by this script. The script can also be used for setting the location of Shared Object Working Environments and platform-dependent make support files.

Synopsis

On Unix: `sniff_arch`

On Windows: `sh <sniff_dir>/bin/sniff_arch`

Output

The output has the format:

```
<Architecture>-<Vendor>-<OperatingSystem>
```

On Sun:

```
% sniff_arch
sparc-sun-sunos4.1
```

On HP:

```
% sniff_arch
pa_risc-hp-hpux9.0
```

On an unsupported platform:

```
% sniff_arch
SNiFF+ unsupported system type
MACHINE=...
RELEASE=...
SYSTEM=...
VERSION=...
```

sniff_genproj

`sniff_genproj` is a program for creating SNIFF+ project description files (PDFs) from a directory tree in batch mode.

If the names of generated PDFs are not unique, `sniff_genproj` makes them unique by prefixing the generated PDF names with parent directory names.

Synopsis

```
sniff_genproj [options] <directory>
```

Arguments and Options

`<directory>` specifies the absolute path name of the directory for which SNIFF+ project(s) are to be created. For absolute projects, use quoted environment variables to create projects that are easily locatable.

By default all project attributes are taken from your Preferences. See also [New Project Setup view — page 146](#).

Option	Description
-a	Generate project description files (PDFs) for the specified directory and the complete directory tree recursively. If this option is omitted, only one PDF is created for the specified directory.
-d <pdf_dir>	Put the generated PDFs into <pdf_dir> (absolute path name). If this option is omitted, the PDFs are stored in the project directories.
-e	Generate PDFs for empty directories, i.e., ones without any files or subdirectories. If this option is omitted, no PDFs are created for empty directories.
-f <path_to_template>	Generate PDFs according to a template file. The path to the template must be either absolute, or relative to the value specified in the New Project Setup View of your Preferences.

Option	Description (cont.)
-i <ignore_dirs>	Ignore <ignore_dirs> directories. Multiple directories must be separated with a colon (:). (default: can be defined in the Preferences).
-k	Follow symbolic links neither to files nor to directories (by default, symbolic links are followed).
-n <name>	Save the root project's PDF under <name>. If this option is omitted, the file is saved under name <directory>.extension (see also -x option).
-r <abs>	Make the generated projects relative, i.e., relative to a root directory, by cutting away the absolute path <abs>. If this option is omitted, absolute projects are created.
-S <abs>	Make the generated projects shared, i.e., relative to a root directory, by cutting away the absolute path <abs>. If this option is omitted, absolute projects are created.
-t <tag>	Use <directory>_<tag>.extension as the name of the generated PDF (this is ignored for the root directory if the -n option is present).
-u <pdf>	Make name of existing PDF <pdf> and its subproject PDFs unique. You would run sniff_genproj with this option after generating PDFs in a directory that contains identically-named subdirectories.
-v	Unix only: Print version number of your current sniff_genproj release.
-x <extension>	Use <extension> as the extension for all created project files. If this option is omitted, the first extension of the File Types attribute Signatures is used. See also File Types view — page 153 .

Sniffaccess

Introduction

Aside from the graphical user interface, Sniffaccess is the means by which SNIFF+ interacts with the outside world. Sniffaccess is an executable program that connects to SNIFF+ and allows you to send requests to and receive notifications from SNIFF+. In other words, Sniffaccess provides control integration between SNIFF+ and other programs.

Sniffaccess can be used for several purposes:

- Driving SNIFF+ in batch mode. For example, this is used for unattended updates and builds of working environments.
- Integrating SNIFF+ with other third-party tools like CASE tools and GUI builders.
- Writing proprietary scripts for project generation and modification with SNIFF+.

This chapter explains how to use Sniffaccess and describes all of the available requests and notifications.

Invoking Sniffaccess

Sniffaccess can be invoked in three ways:

- by a single request on the command line
- interactively
- in batch mode with a command file

A single request on the command line

Sniffaccess sends the request to `sniff`, prints the reply and terminates.

Usage:

```
% sniffaccess [-q] REQUEST [ARGUMENTS]
```

- Note that the `-q` flag can be used for all requests. It suppresses output of `sniff` status and information comments. Data output is not suppressed.

Example:

```
% sniffaccess open_project $SHARED_WE/pe/sniff.shared
```

Interactively

Starting Sniffaccess without arguments will prompt you for Sniffaccess commands.

Usage:

```
% sniffaccess
```

Example:

```
% sniffaccess
sniffaccess: connected to sniffappcomm
sniffaccess: setting timeout to 20 seconds
> browse_class * ActionButton
sniffaccess: port number of sniff is 1000
sniff: OKAY
> edit_symbol * ActionButton::GetMinSize METHOD_IMPL
sniff: ERROR: ActionButton::GetMinSize: symbol not found
> edit_symbol */ ActionButton::GetMinSize METHOD_IMPL
sniff: OKAY
> exit
sniffaccess: disconnecting from sniffappcomm
%
```

Batch mode with a command file

Sniffaccess can also be started with input redirect from a command file.

Usage:

```
% sniffaccess < COMMAND_FILE
```

COMMAND_FILE contains a list of requests, one request per line. Comments start with '#' and end with the next newline.

Example COMMAND_FILE:

```
#
# An example file for SNIFF+ external access
#
# Sets the maximum time to wait for a reply from sniff to 60
seconds.
# Loads a project and then registers for all possible
actions inside sniff,
# waits for notifications and echoes each notification.
set_timeout 60 # timeout is 60 seconds
open_project /Projects/SNIFF+/SharedWE/HEAD/pe/sniff.shared
register * * post nowait echo
```

Multiple simultaneous Sniffaccess sessions

You can have multiple Sniffaccess sessions running at the same time. To do so, type the following on the command line:

```
% sniffaccess -s <sniffaccess_session_name>
```

<sniffaccess_session_name> is a “session id” that uniquely identifies each Sniffaccess session. Note that the session id is a string and cannot contain any blank spaces within it.

To get a list of the currently active SNIFF+ sessions, type:

```
% sniffaccess -i
```

When you execute this command, you will get output similar to the following:

```
Current active SNIFF+ sessions
session0
session1
```

To connect to an active SNIFF+ session, select one of the session id's and use it for <sniffaccess_session_name> above.

SNIFF+ external access communication protocol

Sniffaccess distinguishes between requests and notifications. Requests are sent to SNIFF+ and usually are actions for SNIFF+ to execute. Notifications are sent by SNIFF+ to Sniffaccess upon the execution of a registered action. The following is a description of the possible requests and notifications.

Request format

The format for requests from external programs to SNIFF+ is:

```
REQUEST = REQUEST_STRING [ ARGUMENTS ].
```

Arguments can be enclosed in double quotes “”.

Comment format

Request strings can contain comments. A comment starts with an unquoted ‘#’ and ends with the next newline.

Example:

```
# this is a comment
retrieve */ "menu" # retrieves the string "menu" from all
projects
```

Project name format

Projects are specified this way:

```
PROJECT = PROJECT_PATH [ "/" ].
PROJECT_PATH = PROJECT_NAME | PROJECT_PATH "/"
PROJECT_NAME .
PROJECT_NAME = PATTERN .
```

A PATTERN has the same format as filename patterns in the bourne shell.

Wildcard character	Match
*	any number of characters
?	a single character
[a-zA-Z]	a single character in the given range of characters
[^...] or [!...]	a single character NOT in the given range of characters

Project specification examples:

Project specification	Description
fb.proj/et3.proj/CONTAINER.proj	subproject CONTAINER.proj only
sniff_et3.proj	root project sniff_et3.proj only
sniff_et3.proj/	root project sniff_et3.proj and all sub-projects
sniff_et3.proj/et3.proj	subproject et3.proj of project sniff_et3.proj
sniff_et3.proj/et3.proj/*	all subprojects (nonrecursive) of et3.proj (without et3.proj)
sniff_et3.proj/et3.proj/*/*	all subprojects (recursive) of et3.proj (without et3.proj)
*/	all loaded projects recursively

Symbol format

The following is the symbol format:

```
SYMBOL_NAME = NON_MEMBER_NAME | CLASS_NAME ":" : "
MEMBER_NAME .
```

```
TYPE_SPEC = "ANY" | "CLASS" | "INSTVAR" | "METHOD_DEF" |
"METHOD_IMPL" | "FRIEND" | "VARIABLE" | "CON-
STANT" | "FUNCTION" | "ENUM" | "ENUM_ITEM" |
"TYPEDEF" | "MACRO" .
```

Path and filename format

The following placeholders are used below:

```
REPOSITORY = PATH .
SHARED_WE = PATHS .
PRIVATE_WE = PATH .
PATHS = PATH [ ":" PATHS ] .
PATH = Unix absolute pathname .
FILE = Unix absolute pathname .
```

Environment variables are allowed in the pathname specifications as long as the resulting path evaluates to an absolute path.

Version and configuration format

The following placeholders are used below:

```
CONFIGURATION = string .
VERSION = integer { "." integer } .
COMMENT = quoted string .
```

Working environment format

The following placeholders are used below:

```
WORKING_ENVIRONMENT = ' "
''' [owner] ("PWE" | "SSWE" | "SOWE" | "RWE") ":" workenv_name '''
owner = string
workenv_name = string
```

Sniffaccess requests

Working Environment requests

- **set_workingenv** WORKING_ENVIRONMENT
Set the SNIFF+ working environment. For PWEs, you have the option of entering the name of the owner of the PWE. If you don't enter the name of the owner, SNIFF+ will use the name of the current user. For all other working environments, the owner is "adm".
- **update_private_we** PROJECT [CONFIGURATION]
Updates PROJECT in the Private Working Environment to the configuration indicated by the parameter. After the update, the specified projects are reloaded. If no configuration is supplied, the default configuration of the project is taken.
Project Editor menu: **Project > Update Checkmarked Projects...**
- **add_workingenv** WORKING_ENVIRONMENT PATH [ACCESSED WORKING_ENVIRONMENT]
Creates a new working environment. An accessed base working environment can also be specified (e.g. an SOWE to be accessed by a new PWE). The working environment can only be created using this request if the user is also allowed to do so from the graphical user interface.
- **delete_workingenv** WORKING_ENVIRONMENT
Deletes a working environment together with all the working environments that access the specified working environment (e.g. deleting an SSWE also deletes all SOWEs and PWEs that access the specified SSWE). The working environment can only be deleted using this request if the user is also allowed to do so from the graphical user interface.
Working Environments menu: **Edit > Delete...**
- **get_workingenv_root** WORKING_ENVIRONMENT
Returns the directory specified for the working environment.

Project requests

Note

When you run the **make_project** and **make_file** commands via the Sniffaccess interface, you immediately receive an **Ok** message from Sniffaccess. However, this does not mean that the Make process is over. Basically, the SNIFF+ Shell tool cannot tell when a Make process has ended. As a result, in critical situations where you need to know exactly when the Make process is over, we suggest that you run Make via the update scripts that are supplied with your SNIFF+ installation

■ **open_project** PROJECT

```
[ "WITH_SYMBOLS" | "WITHOUT_SYMBOLS" ]
[ "NO_CACHE" | "USE_CACHE" ]
[ "OPEN_XREF_DB_STRATEGY (0|1|2)" ]
[ "OPEN_XREF_DB_TIMEOUT 30" ]
```

Open a project. The symbols argument determines whether the symbol table is loaded.
Default: open with symbols.

The cache argument determines whether cached symbol information is used.

Default: cache is not used.

If database-driven cross reference system is used the following database access control strategies can be used:

- Strategy 0: If locks preventing **Write** access to the cross reference database are set, abort opening the project (default).
- Strategy 1: Save and close all open files and projects in active SNIFF+ sessions preventing **Write** access to the cross reference database.
- Strategy 2: Break all locks. Not to be recommended!

Execution of the above strategies commences after the number of seconds entered after the OPEN_XREF_DB_TIMEOUT string. Default: 30 seconds.

Launch Pad menu: **Project > Open Project...**

■ **reload_project** PROJECT_NAME [WORKING_ENVIRONMENT]

Reload a project. If a working environment is specified, the project is opened there.
Default: current working environment. Launch Pad/Project Editor menu: **Project > Reload Project...**

■ **close_project** PROJECT_NAME ["SAVE" | "DISCARD" | "ASK"]

Close a project. By default, SNIFF+ discards modifications to projects.

Launch Pad menu: **Project > Close Project.**

- **check_obsolete_files** PROJECT_NAME

Note: The PROJECT_NAME argument must be unambiguous, that is, do not use wildcards; all sub-projects will always be recursively checked.

Lists all files which are in the project directory tree, but which are not used in projects, i.e. which are not recorded in any of the PDFs in the project structure. These files are listed in the SNIFF+ Log Window. To execute this command, make sure that the project is open in SNIFF+.

Project Editor menu: **Project > Check Obsolete Files...**

- **update_file_info** PROJECT_NAME

Update the file status of all files in all Project Editors of a project.

Project Editor menu: **Project > Synchronize File Status**

- **force_reparse** PROJECT_NAME

Force a reparse of all files of a project.

Project Editor menu: **Project > Force Reparse**

- **update_syntab** PROJECT_NAME

Update the symbol table of a project. This command checks for all files whether they have changed and reparses them if necessary.

Project Editor menu: **Project > Load/Update Symbol Table**

- **generate_xref_info** PROJECT_NAME

Generate and dump (cache) the cross reference information of a project

- **add_subproject** PROJECT_NAME PROJECT

Add a subproject to a project. Specify the path to the PDF.

Project Editor menu: **Project > Add Subproject...**

- **remove_subproject** PROJECT_NAME PROJECT_NAME

Remove a subproject from a project.

Project Editor menu: **Project > Remove Subproject**

- **add_file** PROJECT_NAME FILE

Add a file to a project. The project description file is saved immediately.

Project Editor menu: **Project > Add/Remove Files...**

- **remove_file** PROJECT_NAME FILE

Remove a file from a project. The project description file is saved immediately.

Project Editor menu: **Project > Add/Remove Files...**

- **make_project** PROJECT_NAME [TARGET_NAME]

Make the target of a project.

Project Editor/Source Editor/Shell menu: **Target > Make Project**

- **make_file** PROJECT_NAME FILE
Make a file of a project.
Project Editor/Source Editor/Shell menu: **Target > Make File**
- **update_makefiles** PROJECT_NAME
Update (generate) the make support files of a project.
Project Editor/Source Editor/Shell menu: **Target > Update Makefiles...**
- **get_attributes** PROJECT_NAME ATTRIBUTE
Gets Project Attributes of one or more projects. ATTRIBUTE is specified using the name of the Project Attribute in the PDF.
Project Editor menu: **Project > Attributes...**
- **set_attributes** PROJECT_NAME ATTRIBUTE VALUE
Sets Project Attributes of one or more projects. ATTRIBUTE is specified using the name of the ProjectAttribute in the PDF.
Project Editor menu: **Project > Attributes...**
- **list_dirs** PROJECT_NAME
Lists all source directories in PROJECT_NAME and its sub-projects (private and shared WEs) - one directory per line. List can be used e.g. as source path for a debugger.

Configuration management requests

- **update_cms_info** PROJECT_NAME
Update and dump (cache) the configuration information of a project. Configuration Manager menu: **Configuration > Update Information**
- **checkout_config** PROJECT_NAME [CONFIGURATION]
Check out all files belonging to a configuration of a project. **Note:** Before issuing this command, issue the update_cms_info command.
If no configuration is supplied, the HEAD configuration is taken. The checked-out files are read-only and the repository is not locked.
Configuration Manager menu: **Configuration > Check Out...**
- **delete_config** PROJECT_NAME CONFIGURATION
Deletes a project configuration name.
Configuration Manager menu: **Configuration > Delete Configuration Name...**

- **freeze_config** PROJECT_NAME CONFIGURATION [“OVERRIDE”]
Freezes the default configuration of a project (as defined in the working environment). The OVERRIDE argument allows a previous configuration name to be associated with the configuration to be frozen.
Configuration Manager menu: **Configuration > Freeze Default Configuration...**
- **rename_config** PROJECT_NAME CONFIGURATION_OLD CONFIGURATION_NEW
Renames a project configuration.
Configuration Manager menu: **Configuration > Rename Configuration**

File requests

- **show_difference** PROJECT_NAME FILE VERSION1 VERSION2 [VERSION3]
Starts a diff on FILE. VERSION1 is the left version, VERSION2 is the right version, and VERSION3 is optional — if present a 3-way diff is started. Version can be either a version label (e.g. HEAD) or a version number (e.g. 1.1.2).
* Menu: **File > Show Differences...**
- **edit_file** PROJECT_NAME FILE [LINE_NUMBER]
Load a file into a reusable Source Editor. If necessary, open a new Source Editor. If a line number is supplied, select that line and position to it.
Project Editor: **double-click** on file.
- **save_file** PROJECT_NAME FILE
Save a file. The file is only saved if it is loaded in a Source Editor and modified.
* Menu: **File > Save.**
- **file_changed** PROJECT_NAME FILE
Tell SNIFF+ that a file has changed. SNIFF+ reparses the file and updates the symbol information.
- **checkout_file** PROJECT_NAME FILE (“EXCLUSIVE” | “CONCURRENT” | “NOLOCK”)
[VERSION_OR_SYMBOL BRANCH_FLAG]
Check out a file to the Private Working Environment. BRANCH_FLAG can be 'T' or 'F'. If BRANCH_FLAG is set to 'T', a branch is created from the specified version. Default: version HEAD and no branching (BRANCH_FLAG is 'F').
* Menu: **File > Check Out...**

Note

The value of BRANCH_FLAG depends on your underlying version control system:

checkout_file request: T for SCCS, F for RCS

checkin_file request: F for SCCS, T for RCS

- **checkin_file** PROJECT_NAME FILE (“UNLOCK” | “LOCK”) [VERSION_OR_SYMBOL CHANGE_SET COMMENT BRANCH_FLAG BRANCH_NAME]

Check in a file. BRANCH_FLAG can be 'T' or 'F'. If BRANCH_FLAG is set to 'T', a branch is created from the specified version. Default: HEAD version, no change set, no comment, no branching.

* Menu: **File > Check In...**

- **lock_file** PROJECT_NAME FILE (“EXCLUSIVE” | “CONCURRENT”) [VERSION_OR_SYMBOL]

Lock a file. Default version: HEAD.

Project Editor/Source Editor/DiffMerge menu: **File > Lock...**

- **unlock_file** PROJECT_NAME FILE [VERSION_OR_SYMBOL]

Unlock a file. Default version: HEAD.

* Menu: **File > Unlock....**

Symbol requests

- **browse_class** PROJECT_NAME CLASS

Load a class into the Class Browser.

* Menu: **Class > Browse Class** *class*.

- **hierarchy** PROJECT_NAME CLASS (“RESTRICTED” | “FULL”)

Load a class into the Hierarchy Browser and select it. Default mode: FULL.

* Menu: **Class > Show** *class in Hierarchy*.

- **find_symbol** PROJECT_NAME SYMBOL

Find a symbol in the Symbol Browser.

* Menu: **Info > Find Symbols Matching** *selection*.

- **edit_symbol** PROJECT_NAME SYMBOL [TYPE_SPEC FILE]

Load a symbol into the Source Editor. If the symbol is unambiguous, TYPE_SPEC and FILE can be omitted. If the symbol is ambiguous, the first symbol found is taken.

Source Editor menu: **Show > Symbol(s)** *symbol*.

- **retrieve** PROJECT_NAME STRING [“CASE_SENSITIVE” “WHOLE_WORD”]

Retrieve a string in the Retriever. Default: retrieve case-insensitive and also part of a word. * Menu: **Info > Retrieve** *string*.

- **documentation** PROJECT_NAME SYMBOL [TYPE_SPEC FILE]

Load the documentation of a symbol into the Documentation Editor. If the symbol is unambiguous, `TYPE_SPEC` and `FILE` can be omitted. If the symbol is ambiguous, the first symbol found is taken.

*Menu: **Info > Show documentation of *symbol***

Data query requests

- **list_projects**

List all open projects (including complete path) to `stdout`.

- **list_classes** PROJECT_NAME

List all classes of a project to `stdout`.

- **list_files** PROJECT_NAME ["FULLPATH"]

List all open projects (including complete path) to `stdout`.

General requests

- **set_timeout** SECONDS

Set the timeout for reply waits between SNIFF+ and Sniffaccess.

- **iconify** [PROJECT_NAME]

Iconify the windows of a project. If `PROJECT_NAME` is omitted, all open projects and SNIFF+ itself are iconified.

- **normalize** [PROJECT_NAME]

Normalize (de-iconify) the windows of a project. If project is omitted, all open projects and SNIFF+ itself are normalized.

- **wait** SECONDS

Wait for the specified time and then process the next request; SNIFF+ continues processing in the meantime.

- **quit** ["SAVE" | "DISCARD" | "ASK"]

Quit SNIFF+ and Sniffaccess. By default, SNIFF+ discards modifications to files and projects.

- **exit**

Quit only Sniffaccess, but don't quit SNIFF+.

Request replies

SNiFF+ replies to each request with a message like this:

```
sniff: OKAY
```

or

```
sniff: ERROR: filebrowser.proj: no such project
```

The format of the reply is:

```
RequestReply = SimpleOK | DetailedReply.
```

```
SimpleOK = "sniff: OKAY".
```

```
DetailedReply = "sniff: " ("OK" | "WARNING" | "ERROR")[": "
    MessageText].
```

```
MessageText = string.
```

Sniffaccess notifications

Sniffaccess can register to be notified upon the execution of actions in SNiFF+. The notification registration requests allow you to register for specific actions.

Notification registration requests

- **register** PROJECT_NAME ACTION_PATTERN ("PRE" | "POST") ("WAIT" | "NOWAIT") SHELL_COMMAND

Register for a notification. Please refer to [Notification ACTIONS and ARGUMENTS — page 272](#) for a list of possible ACTIONS. The SHELL_COMMAND is called (notified) upon the execution of a registered action.

- **unregister** PROJECT_NAME ACTION_PATTERN ("PRE" | "POST")

Unregister for a notification. Below you can find a list of possible ACTION_PATTERNS. If the action is not registered, a warning is displayed.

ACTION_PATTERN is a pattern that must match one or more ACTIONS as shown in [Notification ACTIONS and ARGUMENTS — page 272](#) and can contain the following wildcard characters:

Wildcard character	Match
*	any number of characters
?	a single character
[a-zA-Z]	a single character in the given range of characters
[^...] or [!...]	a single character NOT in the given range of characters

What happens when SNIFF+ sends a notification?

When SNIFF+ sends a notification, Sniffaccess calls the shell command (SHELL_COMMAND) that has been supplied with a notification registration. The notification registration arguments PRE and POST determine when a shell command is called. PRE means that Sniffaccess is notified before a registered action is executed in SNIFF+; POST means that Sniffaccess is notified after a registered action is executed.

A PRE notification can have either WAIT or NOWAIT status. When Sniffaccess calls the shell command of a PRE WAIT notification, SNIFF+ waits for the shell command to finish before the action is executed. If the return value of the shell command is zero (0), the action is executed; otherwise SNIFF+ does not execute the action. Thus a PRE WAIT notification shell command can control whether SNIFF+ can execute registered actions. When an action is blocked via the Sniffaccess interface, a dialog informs the user about the block.

The shell command is called with the following arguments:

```
SHELL_COMMAND ("PRE" | "POST") ("WAIT" | "NOWAIT") ACTION
{ ARGUMENT }
```

The possible ACTIONS and ARGUMENTS are described below.

Notification ACTIONS and ARGUMENTS

The following shows the possible ACTIONS and ARGUMENTS for notification registration requests. Each item also represents the argument of the shell command which is called by Sniffaccess upon notification of an action execution.

- **save_file** PROJECT_NAME FILE
SNIFF+ saves a file.
- **edit_file** PROJECT_NAME FILE
SNIFF+ loads a file into the Source Editor.
- **checkout_file** PROJECT_NAME FILE ("EXCLUSIVE" | "CONCURRENT" | "NOLOCK")
[VERSION_OR_SYMBOL]
SNIFF+ checks out a file.
- **checkin_file** PROJECT_NAME FILE ("UNLOCK" | "LOCK") [VERSION_OR_SYMBOL
COMMENT]
SNIFF+ checks in a file.
- **lock_file** PROJECT_NAME FILE ("EXCLUSIVE" | "CONCURRENT")
[VERSION_OR_SYMBOL]
SNIFF+ locks a file.
- **unlock_file** PROJECT_NAME FILE [VERSION_OR_SYMBOL]
SNIFF+ calls *make* for a file.

- **make_file** PROJECT_NAME FILE
SNIFF+ calls *make* for a file.
- **make_project** PROJECT_NAME [TARGET]
SNIFF+ calls *make* for a project.
- **open_project** PATH
SNIFF+ opens a project.
- **close_project** PROJECT_NAME
SNIFF+ closes a project.
- **quit**
SNIFF+ quits.

The following shows the possible ACTIONS for POST notification registration requests. Each item also represents the argument of the shell command which is called by Sniffaccess after notification of an action execution.

- **add_subproject** PROJECT_NAME PATH
SNIFF+ adds a subproject to a project.
- **remove_subproject** PROJECT_NAME PROJECT_NAME
SNIFF+ removes a subproject from a project.
- **add_file** PROJECT_NAME FILE
SNIFF+ adds a file to a project.
- **remove_file** PROJECT_NAME FILE
SNIFF+ removes a file from a project.

HP Softbench BMS bridge (Unix only)

The SNIFF+ package contains a generic and extendable bridge between the Sniffaccess interface and the HP Softbench Broadcast Message Server (BMS). The bridge consists of two commands that route messages between Sniffaccess and the BMS. The two commands are implemented as shell scripts, are named `sniff_to_softbench.sh` and `softbench_to_sniff.sh`, and are located in `$SNIFF_DIR/bin`.

This section explains the usage of the bridge and also shows how to extend the bridge with your own commands.

Using the HP Softbench BMS bridge

The commands should be started while SNIFF+ and the Softbench BMS are running and can be executed as background processes. For example, a shell script that starts SNIFF+, the HP Softbench BMS and the bridge could look like this:

Script for starting SNIFF+, Softbench and the bridge	
<code>#</code>	
<code># Script for starting SNIFF+,</code>	
<code># Softbench and the bridge</code>	
<code>#</code>	
<code>sniff &</code>	Start SNIFF+ in the background.
<code>softbench &</code>	Start Softbench and the BMS in the background.
<code>sleep 45</code>	Wait until both programs come up.
<code>sniff_to_softbench.sh &</code>	Start the bridge for communication from SNIFF+ to Softbench.
<code>softbench_to_sniff.sh &</code>	Start the bridge for communication from Softbench to SNIFF+.

sniff_to_softbench.sh

The shell script `sniff_to_softbench.sh` registers for SNIFF+ notifications, transforms them, and routes them to Softbench BMS messages via the `ciclient` program.

The following is the content of `sniff_to_softbench.sh` with an explanation of its structure and how to extend it:

sniff_to_softbench.sh

```
#!/bin/sh
#
# sniff_to_softbench
#
# Convert SNIFF+ action notifications to
# SoftBench requests.
#

PROG=`basename $0`

if [ $# = 0 ]; then

(echo register '*/'file_changed post nowait $0;\
echo register '*/'checkout_file post nowait $0;\
echo register '*/'checkin_file post nowait $0;\
echo register '*/'lock_file post nowait $0;\
echo register '*/'unlock_file post nowait $0)\
| sniffaccess

else
  PRE_POST=$1; shift
  WHEN=$1; shift
  ACTION=$1; shift
  ARGS="$@"
```

If invoked without arguments, register for all commands. This script is invoked again when a notification from SNIFF+ is received, but this time with arguments (see else branch below).

Register self with SNIFF+ for all `file_changed` notifications.

If you want to register for other notifications, you have to extend this list (and also the `case` statement below where the notifications are handled).

Script is invoked with arguments on a notification from SNIFF+. Arguments are: ("PRE"|"POST") ("WAIT"|"NOWAIT") ACTION [ARG S]

sniff_to_softbench.sh (cont.)

```

case "$ACTION" in
  file_changed | checkout_file | checkin_file
|\
  lock_file | unlock_file)
  # $1 is PROJECT, $2 is PATHNAME
  DIR=`dirname $2`
  FILE=`basename $2`
  echo send_notify FILE-MODIFIED $FILE NULL\
    PASS NULL | \
    ciclient -directory $DIR
  ;;
*)
  echo $PROG: cannot convert action $ACTION
  ;;
esac
fi
exit 0

```

On any of the file changed notifications from SNIFF+, send a message to the BMS.

You have to extend this case statement if you want to forward other notifications from SNIFF+. This list must match the registration notifications above.

softbench_to_sniff.sh

The shell script `softbench_to_sniff.sh` registers for Softbench BMS notifications, transforms them, and uses Sniffaccess to react to these notifications.

The following is the content of `softbench_to_sniff.sh` with an explanation of its structure and how to extend it:

softbench_to_sniff.sh

```

#!/bin/sh
#
# Convert SoftBench notifications to SNIFF+
# requests.
#
if [ -z "${child_proc:-}" ]; then
  child_proc=1
  export child_proc

  exec ciclient -toolclass SNIFF -mode name \
    -o 5 -e $0
fi

```

Restart this program as a child of `ciclient`.

softbench_to_sniff.sh (cont.)

```

unset child_proc

PROG=`basename $0`
HOSTNAME=`uname -n`

echo make_reply_trigger -host $HOSTNAME -action\
FILE-MODIFIED -status PASS >&5
echo make_reply_trigger -action STOP -status\
PASS >&5

while read COMMAND ARGS
do
    eval $ARGS
    case $COMMAND in
    nvreply_trigger)
        case $action in
        FILE-MODIFIED)
            sniffaccess file_changed '*'/' `basename\
                $operand`

                ;;
        STOP)
            if [ $toolclass = MSG-SERVER ]; then
                exit 0
            fi
            ;;
        *)
            echo $PROG: unknown action: $action
            ;;
        esac
        ;;
    *)
        echo $PROG: unknown command: $COMMAND
        ;;
    esac
done
exit 0

```

Register self with Softbench for FILE-MODIFIED messages.

If you want to register for other messages, you have to add additional echo lines (and also add the corresponding case branches below).

Read and parse Softbench messages and send the corresponding commands to SNIFF+ via Sniffaccess.

You have to extend this case statement if you want to handle other messages from Softbench. This list must match the registrations above.

Introduction

SNiFF+ supports customization at three different levels:

- **Site level**—Each installation of SNiFF+ can have its own settings. These settings are used by all users that access this installation. Location of the `SitePrefs.sniff` file:
 - `your <sniff_installation_dir>`
- **User level**—Each user can have private settings that override or merge with the site level settings. Location of the `UserPrefs.sniff` file:
 - On Unix: `$HOME/.sniffrc/`
 - On Windows: `%SNIFF_DIR%\Profiles\\`
- **Project**—Each project has its own project attributes that are stored in the project description file (PDF) and can be edited by the Project Attributes dialog. [Project Attributes — page 163](#).
- Most of the various customization settings can be done in your Preferences. For details, please refer to [Preferences — page 123](#).
- **On Windows NT/95**, you can set SNiFF+'s look and feel in your Preferences file. For details, see [Setting SNiFF+'s look and feel \(Windows NT/95 only\) — page 290](#).

Customizing the SNIFF+ <Meta> key (Unix only)

By default, SNIFF+ recognizes the Xwindows modifier key 1 (`mod1`) as <Meta>. Some vendors use the same modifier key for generating special characters. Using the same <Meta><key> combination for generating special characters and as a SNIFF+ keyboard shortcut can cause a conflict. In such a case the special character is generated and the SNIFF+ keyboard shortcut is executed as well. If you want to use a special modifier for SNIFF+ keyboard shortcuts, you can redefine the Xwindows modifier keys and tell SNIFF+ to recognize a different key as <Meta>.

The following text shows a standard keyboard mapping on an HP as output by the `xmodmap` utility:

```
% xmodmap -pm
```

```
xmodmap:      up to 3 keys per modifier, (keycodes in parentheses):

shift         Shift_R (0xc), Shift_L (0xd)
lock          Caps_Lock (0x37)
control       Control_L (0xe)
mod1          Meta_R (0xa), Meta_L (0xb), Mode_switch (0x36)
mod2
mod3
mod4
mod5
```

The output shows that both the left and the right <Meta> keys are translated to `mod1`. To change the mapping, follow these steps:

- Modify the Xwindows key map with `xmodmap`. To associate `mod2` with the left <Meta> key in the example above, you could type the following:

```
% xmodmap -
clear mod1
clear mod2
add mod1 =      Meta_R Mode_switch
add mod2 =      Meta_L
<CTRL>D
```

You can apply the modification at every start-up of your Xwindows server by adding the appropriate lines to the `.xinitrc` file. (Please refer to the Xwindows documentation for more information.) Please note that the `Mode_switch` key may not be necessary on your machine.

- **On Unix:**

Load the `$HOME/.sniffrc/UserPrefs.sniff` (or the `$SNIFF_DIR/SitePrefs.sniff`) into an editor.

- Search for the second occurrence of `UseModifierKey` and change it to the modifier you have selected with `xmodmap`. For example, modify the entry as follows:

```
"*.UseModifierKey" [ 2 ]
```

Please be careful that you are not modifying the schema entry

```
"*.UseModifierKey" [ ! 1 "AnyInt" ], which needs to stay exactly as shown.
Also make sure that you insert a blank before and after the modifier number. You can set
modifier numbers from 1 to 5.
```

- Restart SNIFF+.

SNIFF+ now uses the changed modifier key as <Meta>, which does not interfere with the other modifier key that can be used for generating special characters.

Template files

When new files are created in SNIFF+ by means of the New File dialog, they are filled with templates. The template that is used is determined by the extension of the new file. Templates must be called `template.extension`, whereby `extension` is one of the allowed file type extensions. The location of template files can be specified in the **Projects Template Directory** field in the New Project Setup view of the Preferences dialog.

The Documentation Editor uses special documentation templates for generated documentation. You can customize these template files. The location of the documentation template files can be specified in the **Documentation Template Directory** field in the Documentation Editor view of the Preferences dialog.

To learn how to customize template files, please refer to [Creating documentation templates files — page 300](#)

Parser config file

The Parser configuration file contains special configuration instructions for the Parser. The location of the parser configuration file(s) can be specified in the **Parser Configuration File(s)** field in the New Project Setup view of the Preferences dialog and in the Parser view of the Project Attributes dialog. The Parser considers the configuration file both when preprocessing is enabled and when it is disabled. If preprocessing is enabled, the directives in the configuration file are evaluated and executed after preprocessing. For a list of directives, please refer to [User's Guide — Parser configuration file](#).

Filter file

The Filter file describes filters that are added in the Find and Replace Filters dialog of the Retriever and consists of a sequence of lines of ""- delimited string pairs. The first string is added to the menu and the second string is the regular filter expression that is applied on selecting the corresponding menu entry. In formulating a filter criterion, `%s` can be inserted several times, which will be expanded with the actual match for every retrieved source line. The location of the filter file can be specified in the **Filter File** field in the Retriever view of the Preferences. By default, the filter file is located in the `$SNIFF_DIR/config/Filters` directory.

Custom menus

SNiFF+ allows the definition of custom menus for the Source Editor and Project Editor. Custom menus are defined in a custom menu file which can be located either in a user's home directory or in the SNiFF+ installation directory for site-wide custom menus. The custom menu file is called,

- On Unix:

```
$HOME/.sniffrc/UserMenus.sniff or $SNiFF_DIR/config/SiteMenus.sniff.
```

- On Windows:

```
%SNiFF_DIR%\Profiles\<<Username>\UserMenus.sniff           or
%SNiFF_DIR%\config\SiteMenus.sniff.
```

SNiFF+ searches first for the user-level and then for the site-level custom menu file. The custom menu files are loaded during start-up.

If there are more than 20 entries, the menu can be scrolled by dragging the mouse below the menu border.

Custom menu file format

The custom menu file can define more than one custom menu for each SNiFF+ tool. A new menu is defined by preceding its title with the ">" character. If the first menu does not have a title, its title will be set to **Custom**.

CustomMenuFile	=	{ ToolFrame }
ToolFrame	=	ToolSpec { MenuSpec { MenuEntry } }
ToolSpec	=	"^" ("Editor" "ProjectEditor") NL
MenuSpec	=	">" MenuLabel NL
MenuEntry	=	(SimpleAction InteractiveAction Separator) NL
SimpleAction	=	("shell" "filter" "debugger" "python") "" MenuEntryName "" "" MenuEntryAction ""
InteractiveAction	=	("FileNameDialog" "DirectoryNameDialog" "YesNoDialog") "" MenuEntryName "" "" MenuEntryAction "" "" PromptMessage "" "AskQuestionDialog" "" MenuEntryName "" "" MenuEntryAction "" "" PromptMessage "" [""DefaultValue"]
MenuEntryName	=	MenuEntryString [Accelerator]

MenuEntryAction	=	<i>String that can contain % variables</i>
Accelerator	=	<i>"@" Character</i>
Separator	=	<i>"-"</i>
NL	=	<i>newline</i>

A `shell` command is executed in a SNIFF+ Shell tool.

A `debugger` command is sent to the Debugger.

A `python` command is executed by the python interpreter.

A `filter` command is any kind of process. Its input is the current selection in the Source Editor and its output replaces the current selection.

A separator causes the insertion of a line in the menu. It is used for aesthetic reasons only.

Strings may be delimited with double quotes (""") if they contain blanks.

All menu actions (`'shell'`, `'debugger'`, `'FileNameDialog'`, etc.) can now be extended by `'**'` or `'***'`, for example, `shell*`, `python**` (no blanks before `*` and `**`):

- * all file-dependent variables in the command will be expanded for the selected files. Project dependent variables are expanded using the root project data. The menu action will be executed once only.
- ** the selected files are grouped by project. For each project, the file dependent variables are expanded for all selected files (of the project). The menu action will then be executed for each project that is referenced by the selected files.

Note

This convention is disabled for `'filter'` commands.

Commands are executed in the shell and can contain the following variables:

%A	repository file of selected file
%C	file category directory (dot, if there isn't one)
%d	full path of project description file (PDF)
%D or %W	absolute private project directory
%e	sniffaccess project name format (see Project name format — page 262)
%E	working environment name
%f	full path of source file
%F	name of source file
%g	Shared Object Working Environments (same as in CMVC interface)
%G	Shared Source Working Environments (same as in CMVC interface)
%H	Shared Source Working Environment root directories, separated by semicolon
%i	session id of the SNIFF+ session
%k	selected class/union/struct
%l	locking path of project
%m	selected method
%N	number of expanded files (only for * or ** commands)
%p	Root Main Target (of the current root project)
%P	Main Target (of the project whose files you are currently working on)
%R	Repository Working Environment root directory
%s	a selection, single-quoted and with conversions
%S	a selection, as is
%t	the parameter, single-quoted and with conversions
%T	the parameter, as is
%V	selected file version (in the Project Editor's History window)
%w	Private Working Environment root directory

Lines in the menu files may contain comments. A comment starts with an unquoted number sign (#) and ends with the next newline.

Examples:

```
# this is a comment
shell "echo %s" "echo %s"      # Just output the current
selection
```

Interactive Actions

Shell commands can get information from the user via dialog boxes. The format of the commands is:

```
<command> <menu_item_string> <shell_command>
<prompt_string> [<default_value>]
```

Notice that the format is the same as the shell command, but there is a supplementary prompt string.

The <command> may be one of: `AskQuestionDialog`, `YesNoDialog`, `FileNameDialog` or `DirectoryNameDialog`.

AskQuestionDialog	Presents a prompt string to the user and expects an answer. If Cancel is hit, the command is aborted and the script is not invoked. If OK is pressed the input from user is passed to the shell as %t.
YesNoDialog	Displays a question and prompts for Yes/No/Cancel . Cancel aborts without running the script; otherwise the %t variable is set to YES or NO and the shell command is called.
FileNameDialog	Presents SNIFF+'s standard File dialog with the specified prompt. The name of the selected file is passed to the shell as %t.
DirectoryNameDialog	Prompts the user to select a directory using SNIFF+'s standard Directory dialog. The name of the selected directory is passed to the shell as %t.
<menu_item_string>	Displayed in the . Variables (for example %f) as shown above can be used.
<shell_command>	Command given to the shell. There is a supplementary variable %t. Its value is the user input (depending on the <command>). In %t, newline characters are replaced with the string "\n" to protect them from being interpreted by the shell.

- <prompt_string>** Prompt string which is displayed in the dialog boxes. Variables (for example %f) as shown above can be used.
- <default_value>** Valid for the AskQuestionDialog and is the default value which can also contain the above mentioned variables.

Examples

The following example defines a menu called **Custom** in the Project Editor

```
# Example file for a "Custom" menu in the Project Editor
#
^ ProjectEditor
> Custom
shell "RCS diff" "rcsdiff -kk %l/RCS/%F,v %f"
shell "SCCS diff" "cd %D; sccs -d%l diffs %F"
-
filter Date date
shell "Load File Into vi" "cmdtool vi %f"
-
FileNameDialog "Launch Document..." "open_document %t"
"Choose a document to launch:"
AskQuestionDialog "Special Checkin..." "check_in -version %t %f"
"Enter checkin version:" "%V"
YesNoDialog "Cancel Checkout..." "cancel_checkout %t %f"
"Are you sure you want to cancel your checkout of %f?"
-
debugger "Info Files" "info files"
```

The following example defines two menus for the Source Editor. The first menu is called **Echo**; the second is called **Misc**:

```
# Example menu file for two Editor menus
#
^ Editor
> Echo
shell "echo %s" "echo %s"
shell "echo %d" "echo %d"
shell "echo %f" "echo %f"
shell "echo %D" "echo %D"
shell "echo %F" "echo %F"
shell "echo %l" "echo %l"
-
filter "date" "date"
> Misc
DirectoryNameDialog "Clean Directory..." "clean_directory %t"
    "Choose a directory to clean:"
-
shell "Command 1" "echo 1"
shell "Command 2" "echo 2"
```

It is now possible to use back slashes at the end of a line, inside double quotes, to logically continue the line. For example:

```
python "Greeting"
    "print 'Welcome to SNIFF+' \
    print 'This is an example Custom menu'"
```

Error formats

SNIFF+ integrates various compilers and other tools (like Purify™). The Shell tool and the Debugger are able to interpret the output messages of such tools based on a configurable error formats file. The file `$(SNIFF_DIR)/config/ErrorFormats` contains a list of regular expressions for the most common error formats. If the error messages of your compiler are not covered by an entry in that file, you can add the corresponding regular expression. Regular expressions are explained in [Regular Expressions in SNIFF+ — page 307](#).

Supplied ErrorFormats file (extract)

```
# SNIFF+ - regular expressions for compiler error messages

# "file.c", line 123
"\([^" ]+\)",[ ]+line[ ]+\([0-9]+\)
# file.c, line 123
\([^" ]+\),[ ]+line[ ]+\([0-9]+\)
# Purify: [line 123, file.c,
line[ ]+\([0-9]+\),[ ]+\([^" ]+\)
# file.c:123
\([^: ]+\):[ ]*\([0-9]+\)
# file.c(123)
\([^" ]+\.\.[^" ]+\)\([0-9]+\)
```

The parts of the regular expression that match the file name and the line number must be enclosed in a `\(\)` construct. Each regular expression must have exactly two such constructs.

Setting SNIFF+'s look and feel (Windows NT/95 only)

In general, all Windows system settings are valid in SNIFF+. Flags set in your Preferences file (located in `<sniff_dir>\.UserPrefs`) tell SNIFF+ to apply these system settings to its look and feel. You can, however, change the value of these flags as desired.

The following flags affect SNIFF+'s look and feel:

- `WindowSystem.Look`
- `WindowSystem.Feel`
- `Env.Source`
- `WindowSystem.NativeMenu`

These flags are set in the following lines of code in `<sniff_dir>\.UserPrefs`:

```
"WindowSystem.Look" [ ]
"WindowSystem.Feel" [ ]
"Env.Source" [ ]
"WindowSystem.NativeMenu" [ ]
```

A flag's value is given by a number in the brackets ([]). Initially, these brackets are empty, meaning that the flag's default value applies.

Setting SNIFF+'s look

SNIFF+'s look is set by the `WindowSystem.Look` flag. SNIFF+ can have one of three looks:

- Windows NT/95 (default look)
- Motif
- ET++

To use the Windows NT/95 look

- set `WindowSystem.Look` to 2, like this:

```
"WindowSystem.Look" [ 2 ]
```

To use the Motif look

- set `WindowSystem.Look` to 1, like this:

```
"WindowSystem.Look" [ 1 ]
```

To use the ET++ look

- set `WindowSystem.Look` to 0, like this:

```
"WindowSystem.Look" [ 0 ]
```

Setting SNIFF+'s feel

SNIFF+'s feel is automatically set to match the corresponding look. For example, if the Windows 95 look is set, the Windows 95 feel will also be set.

If you want to change SNIFF+'s feel, you can do so with the `WindowSystem.Feel` flag.

SNIFF+ can have one of three feels:

- Windows NT/95
- Motif
- ET++

To use the Windows NT/95 feel

- set `WindowSystem.Feel` to 2, like this:

```
"WindowSystem.Feel" [ 2 ]
```

To use the Motif feel

- set `WindowSystem.Feel` to 1, like this:

```
"WindowSystem.Feel" [ 1 ]
```

To use the ET++ feel

- set `WindowSystem.Feel` to 0, like this:

```
"WindowSystem.Feel" [ 0 ]
```

Overriding operating system color settings

By default, your operating system's color settings are applied to SNIFF+'s user interface. You can use the `Env.Source` flag to apply the settings in your Preferences instead.

To use the color settings in your Preferences

- set `Env.Source` to 0, like this:

```
"Env.Source" [ 0 ]
```

Using ET++ menus

By default, native Windows 95 menus are used in SNIFF+. You can use the `WindowSystem.NativeMenu` flag to use ET++ menus instead.

To use ET++ menus

- set `WindowSystem.NativeMenu` to 0, like this:

```
"WindowSystem.NativeMenu" [ 0 ]
```


Working with IDL Projects in SNIFF+

Introduction

This chapter describes and gives instructions on how to use the SNIFF+ IDL Parser. Note that the Parser is included in your installation kit. However, it requires a separate license and must also be purchased separately. Please note that the SNIFF+ IDL Make Support is specific for IONA ORBIX.

What the SNIFF+ IDL Parser does

The SNIFF+ IDL Parser parses IDL files according to the CORBA 2.0 standard. Symbol information is extracted from the IDL files during parsing. This symbol information is then mapped into C/C++ data types and placed in the Symbol Table.

The symbol information in the Symbol Table is persistent between sessions, meaning that you do not have to reparse IDL source files (and source files of all other languages that SNIFF+ supports, for that matter) whenever you open a project that contains IDL source files.

Integration of the SNIFF+ IDL Parser with SNIFF+

As just mentioned, the symbol information of IDL source files is stored in the Symbol Table. SNIFF+ accesses this information from the Symbol Table whenever you open a project that contains IDL source files. The Symbol Table is the central data repository that is used by the various SNIFF+ tools for displaying information.

How IDL information is mapped by the SNIFF+ IDL Parser

The SNIFF+ IDL Parser maps symbol information from IDL files to C/C++ data types according to the following mapping scheme:

Information in IDL	C/C++ data type
definition of an interface and its basis interfaces	class definition and base classes
definition of a type	type definition
definition/usage of a constant	constant
definition/usage of an attribute	instance variable
definition/usage of an operation	function

Note

Modal parameters (in, out, inout) and context arguments in IDL do not have corresponding C/C++ data types.

Using the SNIFF+ IDL Parser without SNIFF+

To check the syntax of an IDL file, you can launch the SNIFF+ IDL Parser independently of your SNIFF+ session on the command line interface. Note that this is the only way to use the SNIFF+ IDL Parser independently of a SNIFF+ session.

To run the SNIFF+ IDL Parser, enter the following from the command line interface:

```
sniffidl [options] <name_of_idl_file>
```

You can include one or more of the following options when running `sniffidl`:

- h[elp] message with command line syntax and options is printed
- v[ersion] version number of `sniffidl` is printed
- e<file> errors and warnings found during parsing are written to <file>

Using SNIFF+'s Make Support for compiling IDL files

What makes IDL files so unique?

It is not possible to create a simple implicit rule in *Make* to create target files (source and object) out of IDL files (of the type `*.idl`). This is because an IDL compiler has three types of source targets: `*C.cc`, `*S.cc` and `*.hh`.

Files of the type `*C.cc` are *client* files, and files of the type `*S.cc` are *server* files. Files of the type `*.hh` are *header* files. There is one set of header files for both client and server files. In addition to source targets, IDL compilers also have object targets of the type `*C.o` and `*S.o`.

IDL file types

The source targets of the IDL compiler are *derived files*. Derived files are associated with SNIFF+ file types that are automatically added to and removed from projects. (The **Add/Remove Automatically** attribute of these file types is set to **TRUE**.)

The following table lists the file types that are associated with IDL derived files:

IDL derived file	Derived file type
<code>*C.cc</code>	IDL Client Implementation
<code>*S.cc</code>	IDL Server Implementation
<code>*C.o</code>	IDL Client Object
<code>*S.o</code>	IDL Server Object
<code>*.hh</code>	IDL Header

IDL Projects

You will need to create two types of projects (server project and client project) for your IDL project files in the same directory (`grid_server`) and configure Make support for each of them.

Differences between client and server projects

Client projects contain the following files:

- `*.idl`
- `*C.cc`
- `*.hh`
- `*C.o`

When you compile the `*.idl` files in client projects, the derived files `*C.cc` and `*.hh` are added automatically to the projects.

Server projects contain the following files:

- `*.idl`
- `*S.cc`
- `*S.o`
- `*.hh`

When you compile the `*.idl` files in server projects, the derived files `*S.cc` and `*.hh` are added automatically to the projects.

Note

The IDL interface files (`*.idl`) and header files (`*.hh`) are the same in both client and server projects.

Client projects have their own `.sniffdir` directory (e.g., `.sniffdir_client`) for derived files. Server projects also have their own `sniffdir` directory (e.g., `.sniffdir_server`) for derived files.

Finally, client and server projects have different Makefiles (e.g., `Makefile_client` and `Makefile_server`, respectively). Each project's Makefile is unique to the project and allows *Make* to create the project's target source files.

Editing \$\$SNIFF_DIR/make_support/<platform>.mk

Before you begin creating the server and client projects and configuring Make support for them, you will have to edit \$\$SNIFF_DIR/make_support/<platform>.mk.

In this file, enter the paths of your Orbix IDL compiler installation and IDL binaries, library and include directories. The following is an example of what these lines in <platform>.mk should look like (highlighted in **boldface**):

```
# orbix installation directory
IDL_DIR = <installation_directory>

# binary, library and include directories
IDL_BINDIR = $(IDL_DIR)/<binaries_subdirectory>
IDL_LIBDIR = $(IDL_DIR)/<library_subdirectory>
IDL_INCDIR = $(IDL_DIR)/<include_subdirectory>

# IDL compiler
IDL = $(IDL_BINDIR)/idl
IDLFLAGS =
# server and client libraries
IDL_SERVER_LIB = -lITsrv
IDL_CLIENT_LIB = -lITclt
```

Creating a server project and configuring Make support for it

For example, we assume that:

you have a directory called `grid_server` containing a file called `grid.idl`.

- Copy the `template.Makefile` file from your `SNIFF_DIR/config` directory to your `grid_server` directory and rename it `Makefile.server`.

This is your server Makefile.

You will now create a project for the server.

1. Start SNiFF+.
2. In the Launch Pad, select **Project > New Project... > with Defaults...**
3. In the Directory dialog that appears, navigate to the `grid_server` directory.
4. Press **Open** and then **Select**.

The Attributes of New Project dialog appears.

In the Attributes of a New Project dialog

1. Select the **General > Advanced** node.
2. In the **Advanced view > Generated Files Directory** field, enter `.sniffdir_server`.
3. Select the **Build Options** node.
4. In the Build Options view, enter a Make command in the **Make Command** field, so that *Make* uses the project Makefile `Makefile.server` instead of the default Makefile.

Example:

```
make -f Makefile.server
```

5. Select the **Build Options > Project Targets** node.
6. In the **Executable** field, enter the executable target name (e.g., `grid_server`)
7. In the **+Libraries Linked** field (below the **Executable** field), enter `$(IDL_SERVER_LIB)`.
8. Select the **File Types** node.
9. In the File Types view, select the **Make** file type (by single-clicking it) and delete the entry in the **General tab > Signatures** field. Then enter `Makefile.server` in this field.
10. Press the **Show All** button to see all the file types.
11. For each of the following file types, select the file type (in the order given) and then press the **Add File Type** button:
 - IDL Interface
 - IDL Header
 - IDL Server Implementation
 - IDL Server Object

Now, files of this file type will be added to the `grid_server` project during project creation.

12. Press **OK** to create the project.
13. In the dialog that appears asking if you want to generate cross reference information, press **No**.
14. Add server source files to the project by copying these files to the `grid_server` directory and then choosing **Project > Add/Remove Files to/from grid_server.shared...** in the Project Editor. Remove client source files from the project and make sure that only server files are part of the project.

You are now ready to edit and then run the Makefile for your server project.

15. Edit Makefile.server.

The following example shows part of your Makefile. The IDL-specific commands are highlighted in **boldface**. Make sure that these commands are in your Makefile. If they are not, please add them:

```
# Project makefile
# Location of make support files
SNIFF_MAKEDIR = .sniffdir_server

# Include the generated make support files
include $(SNIFF_MAKEDIR)/macros.incl

# The following macros are defined in the Make view of the
project attributes

# Include directive(s)
$(SNIFF_INCLUDE) += -$$(IDL_INCDIR)

# Executable target
LINK_TARGET = $(SNIFF_LINK_TARGET)

# Relinkable object target
RELINK_TARGET = $(SNIFF_RELINK_TARGET)

# Library target
LIB_TARGET = $(SNIFF_LIB_TARGET)

# Libraries received from subprojects.
# Linked to executable and relinkable object target.
SUB_LIBS = $(SNIFF_SUB_LIBS)

# Relinkable objects received from subprojects.
# Linked to executable, relinkable object and library
target.
SUB_RELINK_OFILES = $(SNIFF_SUB_RELINK_OFILES)

# Other libraries linked to executable target
```

```
OTHER_LIBS = $(SNIFF_OTHER_LIBS)
# Other libraries linked to relinkable object target
RELINK_OTHER_LIBS = $(SNIFF_RELINK_OTHER_LIBS)

# Recursive make directories
SUBDIRS = $(SNIFF_SUBDIRS)
# you can define the following additional flags here
#
#OTHER_OFILES = <other object files>
#OTHER_CFLAGS = <other C compiler flags>
#OTHER_CXXFLAGS = <other C++ compiler flags>
#OTHER_YACCFLAGS = <other yacc compiler flags>
#OTHER_LEXFLAGS = <other lex flags>

# IDL-specific macros
OTHER_IDLFLAGS = -B

#OTHER_FFLAGS = <other fortran compiler flags>
OTHER_LDFLAGS = -L$(IDL_LIBDIR)

# PRE_TARGETS is currently used only for IDL make support.
# The PRE_TARGETS macro is needed, since it is not possible
to
# create a simple implicit rule to compile the IDL files
PRE_TARGETS = $(SNIFF_PRE_TARGETS)

# IDL-specific macro
# "S" stands for server and "C" for client
IDL_CFILE_TYPE_SPEC = S

# Common makefile definitions
include $(SNIFF_DIR)/make_support/general.mk

# Include the generated dependencies file
include $(SNIFF_MAKEDIR)/dependencies.incl
```

In the Project Editor

1. In the Project Tree, double-click on `grid_server.proj`.
The Project Attributes dialog appears.
2. In the Project Attributes dialog, select the **Build Options > Directives** node.
3. In the Directives view, press the **Generate...** button to the right of the Include Directive(s) field.
The include directives for this project are now generated and appear in the Include Directive(s) field.
4. In the Project Editor, choose **Target > Update Makefiles...** to update the generated Make support files.
5. Press **Yes** in the dialog that appears.
6. Choose **Target > Make... > grid_server** to build the project's target.
`gridS.cc` and `grid.hh` will be created by the IDL compiler and then `gridS.o` will be created by the C++ compiler.
7. Make sure that `grid_server.proj` is selected in the Project Tree and choose **Project > Reload Project... > In Current Working Environment**.
You should now see the `gridS.cc` and `grid.hh` files in the Project Editor's File List. You have just completed configuring Make support for the server project. You will now have to create the client project and configure Make support for it.

Creating a client project and configuring Make support for it

- Copy the `template.Makefile` file from your `SNIFF_DIR/config` directory to your `grid_server` directory and rename it `Makefile.client`.

This is your client Makefile.

You will now create a project for the client.

1. Start SNIFF+.
2. In the Launch Pad, select **Project > New Project... > with Defaults...**
3. In the Directory dialog that appears, navigate to the `grid_server` directory.
4. Press **Open** and the **Select**.
The Attributes of New Project dialog appears.

In the Attributes of a New Project dialog

1. In the **General view > File Name and Type** field, enter `grid_client`.
2. Select the **General > Advanced** node.
3. In the **Advanced view > Generated Files Directory** field, enter `.sniffdir_client`.
4. Select the **Build Options** node.

5. In the Build Options view, enter a Make command in the **Make Command** field, so that *Make* uses the project Makefile `Makefile.client` instead of the default Makefile.

Example:

```
make -f Makefile.client
```

6. Select the **Build Options > Project Targets** node.
7. In the **Executable** field, enter the executable target name (e.g., `grid_client`)
8. In the **+Libraries Linked** field (below the **Executable** field), enter `$(IDL_client_LIB)`.
9. Select the **File Types** node.
10. In the File Types view, select the **Make** file type (by single-clicking it) and delete the entry in the **General tab > Signatures** field. Then enter `Makefile.client` in this field.
11. Press the **Show All** button to see all the file types.
12. For each of the following file types, select the file type (in the order given) and then press the **Add File Type** button:
 - IDL Interface
 - IDL Header
 - IDL Client Implementation
 - IDL Client Object

Now, files of this file type will be added to the `grid_client` project during project creation.
13. Press **OK** to create the project.
14. In the dialog that appears asking if you want to generate cross reference information, press **No**.
15. Add client source files to the project by copying these files to the `grid_server` directory and then choosing **Project > Add/Remove Files to/from grid_client.shared...** in the Project Editor. Remove server source files from the project and make sure that only client files are part of the project.

You are now ready to edit and then run the Makefile for your client project.

16. Edit `Makefile.client`.

The following example shows part of your Makefile. The IDL-specific commands are highlighted in **boldface**. Make sure that these commands are in your Makefile. If they are not, please add them:

```
# Project makefile
#      Location of make support files
SNIFF_MAKEDIR = .sniffdir_client

#      Include the generated make support files
include $(SNIFF_MAKEDIR)/macros.incl
```

```
# The following macros are defined in the Make view of the
project attributes
```

```
#      Include directive(s)
$(SNIFF_INCLUDE)+= -${IDL_INCDIR}
```

```
#      Executable target
LINK_TARGET = $(SNIFF_LINK_TARGET)
```

```
#      Relinkable object target
RELINK_TARGET = $(SNIFF_RELINK_TARGET)
```

```
#      Library target
LIB_TARGET = $(SNIFF_LIB_TARGET)
```

```
#      Libraries received from subprojects.
#      Linked to executable and relinkable object target.
SUB_LIBS = $(SNIFF_SUB_LIBS)
```

```
#      Relinkable objects received from subprojects.
#      Linked to executable, relinkable object and library
target.
SUB_RELINK_OFILES = $(SNIFF_SUB_RELINK_OFILES)
```

```
#      Other libraries linked to executable target
OTHER_LIBS = $(SNIFF_OTHER_LIBS)
#      Other libraries linked to relinkable object target
RELINK_OTHER_LIBS = $(SNIFF_RELINK_OTHER_LIBS)
```

```
#      Recursive make directories
SUBDIRS = $(SNIFF_SUBDIRS)
```

```
# you can define the following additional flags here
#
#OTHER_OFILES = <other object files>
#OTHER_CFLAGS = <other C compiler flags>
```

```
#OTHER_CXXFLAGS = <other C++ compiler flags>
#OTHER_YACCFLAGS = <other yacc compiler flags>
#OTHER_LEXFLAGS = <other lex flags>

# IDL-specific macros
OTHER_IDLFLAGS = -B

#OTHER_FFLAGS = <other fortran compiler flags>
OTHER_LDFLAGS = -L$(IDL_LIBDIR)

# PRE_TARGETS is currently used only for IDL make support.
# The PRE_TARGETS macro is needed, since it is not possible
to
# create a simple implicit rule to compile the IDL files
PRE_TARGETS = $(SNIFF_PRE_TARGETS)

# IDL-specific macro
# "S" stands for server and "C" for client
IDL_CFILE_TYPE_SPEC = C

# Common makefile definitions
include $(SNIFF_DIR)/make_support/general.mk
# Include the generated dependencies file
include $(SNIFF_MAKEDIR)/dependencies.incl
```

In the Project Editor

1. In the Project Tree, double-click on `grid_client.proj`.
The Project Attributes dialog appears.
2. In the Project Attributes dialog, select the **Build Options > Directives** node.
3. In the Directives view, press the **Generate...** button to the right of the Include Directive(s) field.
The include directives for this project are now generated and appear in the Include Directive(s) field.
4. In the Project Editor, choose **Target > Update Makefiles...** to update the generated Make support files.
5. Press **Yes** in the dialog that appears.
6. Choose **Target > Make... > grid_client** to build the project's target.
`gridC.cc` and `grid.hh` will be created by the IDL compiler and then `gridC.o` will be created by the C++ compiler.
7. Make sure that `grid_client.proj` is selected in the Project Tree and choose **Project > Reload Project... > In Current Working Environment**.
You should now see the `gridC.cc` and `grid.hh` files in the Project Editor's File List.
You have just completed configuring Make support for the client project.
This completes the Make support setup for your IDL server and client projects.

Default Makefile

When you have the server and client project in the same directory and if Make is called from some place higher up in the directory structure, you need to create the following additional Makefile in your `grid_server` directory and call it `Makefile`. This Makefile in turn calls `Makefile.server` and `Makefile.client`.

```
all:
    @if [ -f Makefile.server ]; then\
        $(MAKE) $(MFLAGS) -f Makefile.server all;\
    fi;\
    if [ -f Makefile.client ]; then\
        $(MAKE) $(MFLAGS) -f Makefile.client all;\
    fi

.DEFAULT:
    @if [ -f Makefile.server ]; then\
        $(MAKE) $(MFLAGS) -f Makefile.server $@;\
    fi;\
    if [ -f Makefile.client ]; then\
        $(MAKE) $(MFLAGS) -f Makefile.client $@;\
    fi
```

Regular Expressions in SNIFF+

Introduction

Regular expressions (regex) are a powerful means to specify patterns for filters and search strings in the various SNIFF+ tools. The syntax conforms largely to the GNU regular expression syntax used in the Emacs editor, with some SNIFF+ specific enhancements.

Basically, regular expressions are a system of matching character patterns. How you use regular expressions depends on what you need. This introduction to regular expressions, following the [Quick Reference - Syntax](#) table, groups regular expression usage by functionality.

[Quick Reference - Syntax — page 308](#)

[Literals and metacharacters — page 309](#)

[Escape character — backslash \(\\) — page 310](#)

[Do not match — exclamation point \(!\) \(in SNIFF+\) — page 310](#)

[Single character wild card — period \(.\) — page 310](#)

[Quantifiers — how often to match — page 311](#)

[Position — where to match — page 312](#)

[Nonprinting or whitespace characters — page 314](#)

[Character classes or lists — page 314](#)

[Choosing from a range of alphanumeric characters — page 314](#)

[Excluding a character list — page 315](#)

[Metacharacters inside square brackets — page 315](#)

[Special character classes in SNIFF+ — page 315](#)

[Example — page 316](#)

[Groups, alternatives and back references — page 317](#)

[Examples — page 317](#)

Quick Reference - Syntax

The following table summarizes metacharacters and sequences (`\<char>`) used in SNIFF+ regular expressions.

Regex	matches
<code>\</code>	escape
<code>.</code>	any character except newline
<code>*</code>	zero or more occurrences of preceding
<code>+</code>	at least 1 occurrence of preceding
<code>?</code>	zero or one occurrence of preceding only
<code>^</code>	beginning of line
<code>\$</code>	end of line
<code>[...]</code>	character list
<code>[^...]</code>	complement of character list
<code>!<regex></code>	everything except <regex> — only in SNIFF+ filter fields
<code>\b<regex></code>	word begins with <regex>
<code><regex>\b</code>	word ends with <regex>
<code>\B<regex></code>	word does not begin with <regex>
<code><regex>\B</code>	word does not end with <regex>
<code>\`<regex></code>	file begins with <regex>
<code>\`<regex></code>	file ends with <regex>
<code>\f</code>	formfeed
<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\t</code>	tab
<code>\v</code>	vertical tab
<code>\s</code>	any nonprinting character, that is, [<code>\f\n\r\t\v</code>]
<code>\S</code>	any printing character, that is [<code>^ \f\n\r\t\v</code>]

Regex	matches
<code>\d</code>	any digit, that is [0-9]
<code>\D</code>	any non-digit [^0-9]
<code>\w</code>	any word constituent, that is [A-Za-z0-9_]
<code>\W</code>	any non-word constituent, that is [^A-Za-z0-9_]
<code>\(<regex>)</code>	groups
<code>\1...\19</code>	back references to groups
<code>\ </code>	alternative
<code>%s</code>	used in filters to reference a retrieved string in the Retriever

Literals and metacharacters

Literals in regular expressions are ordinary characters that are literally matched, that is, they match only themselves. Metacharacters are characters that are not matched literally; they are a kind of shorthand for defined functionalities.

Expression	Matches	Does not match
<code>a</code>	a	b, ...
<code>ABC</code>	ABC	123, ...

Metacharacters — escape with backslash

Characters that are used as metacharacters must be preceded by a backslash (\) to be literally matched. These are:

- | | |
|---------------------|-----------------------------|
| ■ Backslash (\) | ■ Period (.) |
| ■ Asterisk (*) | ■ Plus sign (+) |
| ■ Question mark (?) | ■ Square brackets ([...]) |
| ■ Dollar sign (\$) | ■ Caret (^) |

Escape character — backslash (\)

The backslash (\) has “escape” functionality. That is, a literal character following a backslash can escape its “literalness” and, in combination with the backslash, attain new functionality (if defined). Conversely, a metacharacter following a backslash escapes its meta-meaning and is literally matched.

- If a backslash-literal sequence is not defined, the backslash is ignored and the following character is literally matched.

Expression	Matches	Does not match
<code>\</code>	nothing, because not followed by anything (undefined)	<code>\</code> , ...
<code>\\</code>	<code>\</code> (first <code>\</code> escapes meta-functionality)	<code>\</code> <code>\</code> , ...
<code>\n</code>	newline (a defined sequence)	<code>n</code> , <code>\</code> , ...
<code>\a</code>	<code>a</code> (because <code>\a</code> not defined)	<code>\</code> , ...

Do not match — exclamation point (!) (in SNIFF+)

This applies only in filter fields of SNIFF+ tools: An exclamation point at the beginning of a regular expression means “match everything except the following regex”.

Note that this is not a metacharacter in the usual sense (does not have to be escaped, unless it is in position one of the regex). Note also that this is a SNIFF+ specific implementation and not usually part of the regular expression syntax.

Single character wild card — period (.)

Matches any single character, **except** newline (`\n`)

Expression	Matches	Does not match
<code>.et</code>	<code>get</code> , <code>Get</code> , <code>set</code> , <code>2et</code> ...	<code>got</code> ...

Quantifiers — how often to match

Zero or more occurrences — asterisk — *

Note that the asterisk is not a wild card, but a *quantifier*. A regex followed by an asterisk (*) matches **zero or more** occurrences of the regex. A period followed by an asterisk (.*) therefore matches “any character (except newline) occurring any number of times, or not at all”.

One or more occurrences — plus sign — +

A regex followed by a plus sign (+) matches **one or more** occurrences of the regex. A period followed by a plus sign (.*+) therefore matches “any character (except newline) occurring at least once”.

Zero or one occurrence only — question mark — ?

A regex followed by a question mark (?) matches **zero or one** occurrence only of the regular expression. A period followed by a question mark (.*?) therefore matches “any character (except newline) occurring only once or not at all”.

Expression	Matches	Does not match
Do*Command	DCommand myDoCommand DoooCommand ...	DoMenuCommand abc ...
Do+Command	DoCommand myDoooCommands ...	DCommand DoMenuCommand ...
Do?Command	DCommand DoCommand	(anything else)

Position — where to match

Matches can be restricted to their position in words, lines and files.

Beginning or end of word — \b

\b *followed* by a regex matches only at the beginning of a word.

\b *preceded* by a regex matches only at the end of a word.

Not beginning or end of word — \B

A regex *preceded* by **\B** matches everywhere except at the beginning of a word.

A regex *followed* by **\B** matches everywhere except at the end of a word.

Expression	Matches	Does not match
\bCommand	Command Commander	DoCommand ...
Command\b	Command Do Command	Commander ...
\bCommand\b	Command (only)	(anything else)
get\b	get Date, fo get ful	get, forget...
\Bget	fo get , fo get ful	get, getDate...

Beginning of line — caret — (^)

The caret means “match the following regex only if it is at the beginning of a line”. Note that the caret has a different meaning (negation) when it is used within [Character classes or lists — page 314](#).

End of line — dollar sign — (\$)

The dollar sign means “match the preceding regex only if it is at the end of a line”.

Expression	Matches	Does not match
<code>^void</code>	<code>void</code> (only if <code>void</code> is the first text in the line)	<code>// void, avoid, void</code> preceded by any characters...
<code>)\$</code>	<code>foo(a)</code> (only if the <code>)</code> is the last character in the text line)	anything where <code>)</code> is followed by any characters, e.g. <code>);</code>

First in file — \accent grave — (^)

The `^` means “match the *following* regex only if it is at the beginning of a file”.

Last in file — \accent acute — (\$)

The `$` means “match the *preceding* regex only if it is at the end of a file”.

Expression	Matches
<code>^.*</code>	the first line in every file (e.g. in the Retriever)
<code>.*\$</code>	the last line on every file (e.g. in the Retriever)

Nonprinting or whitespace characters

Nonprinting characters are represented as follows in SNIFF+ regular expressions:

- **Any nonprinting character** — `\s` (lower case)
This is a special character class (see [page 314](#)), namely `[\f\n\r\t\v]`, the listed items are:
 - **Space** — `<space>`
 - **Formfeed** — `\f`
 - **Newline** — `\n`
 - **Carriage-return** — `\r`
 - **Tab** — `\t`
 - **Vertical tab** — `\v`

Expression	Matches
<code>[\t]+\$</code>	all (unnecessary) space and tab characters at the end of lines.

Character classes or lists

Character classes — enclosed in square brackets — `[...]`

A character class is a list of characters, any of which can be matched. The list can also be excluded from matches. Ranges of ASCII characters can also be specified.

Expression	Matches	Does not match
<code>[gs]et</code>	<code>get</code> , <code>set</code> (only)	(anything else)

Choosing from a range of alphanumeric characters

A minus sign (-) within square brackets indicates a range of consecutive ASCII characters. For example, `[0-9]` is the same as `[0123456789]`.

Expression	Matches	Does not match
<code>Do[A-Za-z]*Command</code>	<code>DoCommand</code> <code>DoMenuCommand</code> <code>DoMouseCommand...</code>	<code>Do-Command</code> <code>Do88Command</code> <code>abc...</code>

Excluding a character list

If the first character in the square brackets is a caret (^), any character except those in the square brackets will match.

Expression	Matches	Does not match
<code>Do[^\s(Cc]</code>	DoMenuCommand DomouseCommand Domino	Do (followed by space) Do (DoCommand abc...

Metacharacters inside square brackets

To include the **minus sign** itself in a range, it must be the first character (after an initial ^, if any — see [Excluding a character list — page 315](#)), e.g., `Do[-A-Za-z]*Command` would also match `Do-Command`.

If a **right square bracket** (]) immediately follows a left square bracket, it does not terminate the set but is considered to be one of the characters to match.

The **caret** (^) in first position negates the rest of the character class.

All other metacharacters, such as backslash (\), asterisk (*), or plus sign (+) etc., are also matched literally if they are inside square brackets.

Special character classes in SNIFF+

Special character classes can be used in SNIFF+ as a kind of shorthand to make writing (and reading) regular expressions easier.

- **Any word constituent character** — `\w` (lower case) which includes all alphanumerics and underscore, that is, `[A-Za-z0-9_]`
- **Any non word-constituent character** — `\W` (upper case), that is, `[^A-Za-z0-9_]`
- **Any digit** — `\d`, that is, `[0-9]`.
- **Any non-digit** — `\D`, that is `[^0-9]`.
- **Any nonprinting (whitespace) character** — `\s` (lower case), that is, `[\f\n\r\t\v]`. This class is described under [Nonprinting or whitespace characters — page 314](#).
- **Any printing character** — `\S` (upper case) which is everything except nonprinting or whitespace characters, that is, `[^\f\n\r\t\v]`. See also [Nonprinting or whitespace characters — page 314](#).

Example

- The following example tries to match “any call of 'foo' that takes at least one parameter”. The first expression tries to achieve this by excluding ')', but neglects the possibility of nonprinting characters, compound words and nested parentheses. The second expression covers these eventualities and matches the entire call upt, and including, the final closing parentheses.

Expression	Matches	Does not match
<code>foo([^\)]+)</code>	<code>foo(a)</code> <code>foo()</code> <code>myfoo(int a, int b, c)...</code>	<code>foo (a)</code> <code>foo()</code> ...
<code><foo\s*(\s*[\^\s][.\n]*)</code>	any <code>foo</code> with at least 1 parameter	(anything else)

- In the second expression above, recall:
 - `<` — beginning of word
 - `foo` — three ordinary literal characters that match themselves
 - `\s` — any nonprinting character
 - `*` — zero or more occurrences of preceding, here `\s`
 - `(` — an ordinary literal character that matches itself
 - `[\^\s]` — anything except ')' and nonprinting characters
 - `[.\n]*` — any character and newline, zero or more occurrences
 - `)` — an ordinary literal character that matches itself

Groups, alternatives and back references

Groups — enclosed in `\parentheses` — `\(...\)`

Alternatives — `\pipe` — `|`

Back references — `\1...\\9`

Groups enclosed in `\(...\)` serve three purposes

- To group an expression so that it can be treated as if it were a single character. For example, a group will be governed by a quantifier like `*` as if it were a single character:

Thus, **`ba(na)*`** matches `ba`, `bana`, `banana`, `bananana`, etc.

- To enclose a set of `|` alternatives.

For example, **`my(foo|bar)`** matches either `myfoo` or `mybar`.

The alternation (`|`) applies to the largest possible surrounding expressions. Only a surrounding `\ (... \)` grouping can limit the grouping power of `|`.

- To mark a matched substring for back references. The first nine groups in a regex can be referenced using `\1` through `\9`.

For example, **`\(.*)\1`** will match any string that is composed of two identical halves. The **`\(.*)`** matches the first half, which can be anything (except newline), but the **`\1`** that follows must match the exact text matched by the **`\(.*)`** group.

Groups, alternation and back references can be useful in find/change operations either for global editing in the Retriever or in the Source Editor's Find/Change dialog.

Examples

1. This example matches

`SetupMenu` or `SetupStyles` and
changes `Setup` to `Update`
concatenates this with `Menu` or `Styles` (whichever was matched) and
adds the substring `Always` at the end.

Find: **`Setup(Menu|Styles)`**

Change to: **`Update\1Always`**

Good results

Found `DoSetupMenu` and changed to `DoUpdateMenuAlways`

Found `SetupStyles` and changed to `UpdateStylesAlways`

Potential pitfall

Found `SetupMenuStyles` and changed to `UpdateMenuAlwaysStyles`

Remember: "The first match always wins."

2. This example matches `foo(parameter1,parameter2)` and changes the parameter order. Nonprinting characters are not taken into account, nor are nested parentheses considered.

Find: `<foo(\([^,]+\), \([^,]+\))`

Change to: `foo(\2,\1)`

Good results

Found `foo(a,b)` and changed to `foo(b,a)`

Found `foo(int a,char* b)` and changed to `foo(char* b,int a)`

Potential pitfall

Found `foo(f(x,y),int b)` and changed to `foo(y),f(x,int b)`

You could of course “fix” the regex by matching `;` at the end, but that does not solve the actual problem. As always, it’s safer to check before globally changing things.

Regular expressions are cumbersome for nested parentheses of any depth, and cannot handle arbitrary depths.

SNiFF+ - Generated Files

Introduction

SNiFF+ generates files for keeping information persistent between sessions and for supporting the build process. Make Support Files are described in *User's Guide — Build and Make Support*. This section focuses on the other generated files:

- [Symbol tables](#) (*projectname.symtab*)
- [Temporary lexical analysis files](#) (*sourceFilename.lex*)
- [Preprocessor cache files](#) (*sourceFilename.cpp*)
- [Cross reference files](#) (*sourceFilename.ref*)
- [Cross reference indexes](#) (*project.index*)
- [Configuration management cache](#) (*project.VersionTool.cmi*)
- [Custom targets file](#) (*project.user.customtargets*)
- [Tool status file](#) (*project.user.state2*)
- [Retriever index files](#)

(*RetrieverIndex.fii*, *RetrieverIndex.wbi*, *RetrieverIndex0001S.idx*)

These files are stored in the directory indicated by the **Generate Directory** project attribute. See also [General view — page 164](#). By default, this attribute is set to `.sniffdir`. In the rest of this Appendix, we will use this value for the directory.

Note

All generated files can be safely deleted, since SNiFF+ will automatically regenerate them on demand. If they are deleted, actions will take longer to execute the first time while the files are regenerated. Choosing **Delete Symbol Files** from the **Project** menu of the Project Editor will delete all generated files of a project except the Window layout file.

Sharing of symbol information

The symbol information of a project is stored in `.sniffdir`. For shared projects, the symbol information is shared whenever possible; only information which is different from the shared workspace symbol information is stored in the private workspace. SNiFF+ manages the sharing of symbol information transparently for the user by creating and deleting the symbol files in the private workspaces when the workspace changes.

Generated files

Symbol tables

Whenever SNIFF+ loads the source code of a project for the first time, it parses the source files and extracts the symbol information into a symbol table. When the project is closed, this symbol table is stored to disk in the file *projectname.symtab*. The next time this project is opened, the files are not parsed again, but the symbol information is directly loaded from the symbol table file. SNIFF+ checks whether any of the source files have been modified and reparses files when necessary. Thus the symbol information is always kept up-to-date.

Temporary lexical analysis files

During parsing, SNIFF+ generates temporary lexical analysis files for each parsed source file. The files are called *sourceFilename.lex* and are used as input for the generation of the cross reference information. The lexical analysis files are removed when the cross reference information has been successfully generated and stored to disk.

Preprocessor cache files

If preprocessing is enabled, SNIFF+ creates preprocessor files for all preprocessed source files. This speeds up preprocessing, since header files are preprocessed only once and are then directly loaded from the cached file. The cache files are called *sourceFilename.cpp*. The preprocessor cache files are regenerated transparently whenever a source file is modified, reparsed and preprocessed again.

Cross reference files

Cross reference files are generated on demand when cross reference queries are executed. When a cross reference query for a particular project is executed for the first time, SNIFF+ processes the temporary lexical analysis files and generates cross reference tables for fast cross referencing. After the information has been successfully created, it is stored to disk in files called *sourceFilename.ref* and the temporary lexical analysis files are deleted. The next time a cross reference query is executed, the cross reference information is directly loaded from disk. The reference files are updated dynamically and transparently whenever files are modified and reparsed.

Cross reference indexes

To speed up referred-by queries, SNIFF+ generates a cross reference index for each project called *project.index*. This index allows the efficient and fast access of cross reference information for large projects. Cross reference information is stored in *.ref* files. The index is created the first time a comprehensive referred-by query is executed and is then stored to disk. The next time a referred-by query is executed, the index is used to decide what symbols and files need to be considered in order to answer the query. The index is updated dynamically and transparently whenever a file is modified and reparsed.

Configuration management cache

To speed up the access to the configuration management information of underlying configuration management tools, SNIFF+ uses configuration management cache files. The first time the Configuration Manager displays the configuration management information of a project, this information is extracted from the underlying configuration management tool. When the project is closed, the information is dumped to a file called *project.VersionTool.cmi*. The next time the Configuration Manager displays configuration information, it is loaded directly from this file. The cache is updated dynamically and transparently whenever the configuration information of a project changes.

Custom targets file

SNIFF+ stores the user-specific list of custom targets for making, execution and debugging in a file called `project.user.customtargets`. The file can be modified in the Target dialog. See also [Target dialog — page 32](#).

Tool status file

When a root project is closed, SNIFF+ stores the window layout for each user in a file called *project.user.state2*. Whenever a root project is reopened, SNIFF+ restores the window layout to the same state it had when the project was closed. The following information is stored in the tool status file:

- Window locations and sizes
- Window contents
- Active selections
- History menus

By default, the file is located in the `.sniffdir` directory of the project. You can specify another directory under **Tool Status Files** in the project attributes. See also [Tool Status Files — page 168](#)

Retriever index files

If the **Create Index** checkbox is selected in the Preferences [Retriever view — page 137](#), a system of indexes (referred to as “the index” in the following) is created as soon as the first retrieve query is run.

This index serves to restrict text retrievals to only those files where there is high probability of matching a query, thus greatly reducing retrieval time in large projects.

The index is created for the files in the queried projects, and includes also files in projects physically outside the current Working Environment if these have been added as subprojects (e.g. libraries).

Thus, if you run a query over all projects in a given Working Environment (including associated external projects), a complete index of words and files is created.

Alternately, if you query only a subset of the projects, only this subset is indexed. Over time, and as you query different subsets, the index will be incrementally built up.

Index files are stored in the `.RetrieverIndex/` directory in the Working Environment root. If absolute projects are queried outside of Working Environments, the index files are stored in the project root.

The index is made up of the following files:

- `RetrieverIndex.fii`

A list of files that have been indexed.

- `RetrieverIndex.wbi`

A case-insensitive list of words in the indexed files.

The Retriever's **Show Only Similar Words** option provides a selective view to this list.

- `RetrieverIndex0001S.idx`

Word — file mappings. There can be one or more of these files because the size of individual files is optimized for faster access. The Retriever's **Preview File List** option provides a selective view to this list by showing the files where a given word (case-insensitive and not filtered in any way) occurs.

Part IV

Glossary and Index



Glossary

Adaptor is the specific implementation of a SNIFF+ interface. For example, the generic CMVC interface of SNIFF+ has adaptors for CMVC tools. The SNIFF+ debugger interface has adaptors for debuggers (gdb, dbx, etc.).

Branches occur in a version tree when you create new versions of a file from the middle instead of the end of the tree. Basically, SNIFF+ allows you to perform the same operations on branches that you can perform on the main trunk of a version tree.

Browser is a tool that is used for viewing (and not editing) data only. SNIFF+ offers several browsers like the Symbol Browser, the Class Browser and the Hierarchy Browser. The information displayed in browsers can be filtered in several ways.

Build is the process of creating the targets of a project. The build steps are usually described in makefiles which are executed by programs like Make. A build can involve translations of source files and the construction of binary files by compilers, linkers and other tools.

Check-in is the process of checking in a working file from a working environment, thereby creating a new version of the file in the Repository. A complete project can be also checked in. Typically, after a file has been checked in, locks made on the file are removed from the Repository. A file check-in can be associated with a change set. Note that SNIFF+ doesn't check in files itself. It delegates the operation to your underlying CMVC tool (by means of CMVC adaptors).

Check-out is the process of creating an editable working file in the working environment from a specific version of the file in the Repository. Depending on what actions are planned with the file, a lock of that file in the Repository is set. A check-out can be associated with a change set. SNIFF+ delegates the actual check out operation to your underlying CMVC tool (by means of CMVC adaptors).

CMVC is the abbreviation for configuration management and version control.

Concurrent lock is a lock that, unlike an exclusive lock, does not prevent others from locking the same version of a file. Versions that are concurrently locked must be merged back into the Repository.

Configuration is a coherent and consistent state of a system or project. A configuration has a name and refers to specific versions of files in the Repository. Typically, a configuration is a buildable state of a system.

Configuration management is the process of controlling and administrating the components of configurations. Configuration management includes the freezing (baselining) of configurations.

Default Configuration is the version of your software system that you work on. You can set your Default Configuration when you define your working environments. SNIFF+ uses your Default Configuration for the default value when you choose one of the various version

control commands (e.g., checking out file versions, locking/unlocking file versions), and during the updating of Shared Source and Private Working Environments. Source files are made up-to-date with respect to the Default Configuration.

By default, the HEAD version of your software system is the Default Configuration for your Private Working Environment.

Dependency as used in Make is a relationship between two files that says that one file must be updated or rebuilt when the other one changes. In the Makefile, a dependency is a word listed after the colon ':' on the same line as a target. Source-level dependencies can be extracted from source code and are typically stored in dependency files. SNIFF+ generates dependency files that are included by Makefiles as part of its Make Support feature. Build-order dependencies must always be specified explicitly.

Derived file is a file that can be generated (derived) from another file. A typical example of a derived file is an object file that is generated from a source file after compilation.

Documentation template is a file that describes the structure and content of documentation frames. Each documented symbol type has its own documentation template. When documentation for a symbol is generated, SNIFF+ creates a documentation frame for the symbol. You can customize documentation templates.

Documentation frames are created when you tell SNIFF+ to document a symbol in your source code. Empty documentation frames represent the initial, undocumented state of a symbol's documentation. The structure and content of documentation frames are described by documentation templates.

Editor is a tool that is used for both viewing and changing data. SNIFF+ offers a number of editors (e.g., Source Editor and Project Editor). Tools that just show, but do not modify data, are called browsers.

Exclusive lock is set on a version in the Repository when a file is checked out for modification. Each version can have only one exclusive lock, thus preventing other developers from modifying the same version.

File is a component of a project. Each file is associated with a file type.

File type is associated with a file and determines several attributes of the file. Every file of a SNIFF+ project has a file type. SNIFF+ comes with a set of predefined file types, but there can be any number of file types in a project. Examples of file types are C++ implementation, C++ header, makefile, yacc source, shell script, etc. A file's file type determines how SNIFF+ treats the file and what operations may be performed on the file.

Freezing a configuration is the process of creating a "virtual snapshot" of the system (or, to be exact, of its source files) at special times during the software development process. You do this in SNIFF+ by associating the current state (configuration) of all project source files with a single symbolic name. The process of creating a single configuration and associating it with a symbolic name is called "freezing a configuration".

HEAD is the latest version on the trunk or branch of a file's version tree.

History reflects all the different versions of a file and is stored in the Repository file.

Inheritance is a directed relationship between two classes in which one class inherits the attributes of another classes. In single inheritance, a class inherits from only one class. Multiple inheritance means that a class inherits from several classes.

INIT is used by SNIFF+ as name to refer to the initial version of a file in the Repository. The **INIT** version is created when a file is checked into the Repository for the first time.

Interfaces are intermediaries between two components or tools or between the user and the machine. SNIFF+ uses interfaces to interact with the outside world and external tools. A SNIFF+ interface can have multiple specific implementations. called adaptors. For example, the generic CMVC interface of SNIFF+ has adaptors for CMVC tools. The SNIFF+ debugger interface has adaptors for debuggers like gdb and dbx.

Lock is a mechanism that controls access to versions of files in the Repository. SNIFF+ distinguishes between exclusive locks (only one developer can modify a specific version) and concurrent locks (multiple developers can simultaneously modify the same version).

Locking is the process of setting locks.

Main branch is the starting branch of a file's version tree. Unless otherwise specified by your default version control configuration, the main branch is the default branch for CMVC operations.

Make is the program that reads Makefiles and drives the building process. SNIFF+ integrates a wide range of different Make implementations.

Makefile is a text file read by Make programs that describes the building of targets. A Makefile contains source-level dependencies and build-order dependencies. As part of its Make Support feature, SNIFF+ generates Make Support Files that contain both kinds dependency information.

Make macro is a variable in a Makefile which can be assigned a string value. The value can be set in the Makefile itself, on the command line or by setting an environment variable with the same name. SNIFF+ uses Make macros to separate general, platform-specific, project-specific, and team-specific information. A coherent, generic and extendable set of Make macros is part of SNIFF+'s Make Support feature.

Make Support File is either generated out of the project's source code or supplied with the SNIFF+ package. Make Support Files contain the following information: generic Make rules, source-level dependencies, build-order dependencies, project-specific macros and platform-specific macros.

Merge is the process of combining the contents of two or more files into a single file. Typically, the files involved in a merge are versions of a single Repository file. A merge can be done automatically, but often requires manual intervention to resolve conflicts. SNIFF+'s Diff/Merge tool is used for merging files.

Object file is a derived file that is generated from source code after a build. SNIFF+ maintains a list of all object files for a project and generates a make support file containing this list.

Shared Object Working Environment is a working environment that, in contrast to a source working environment, stores only platform-specific files. Typically, a Shared Object Working Environment stores all object files of a project. A Shared Object Working Environ-

ment always accesses a corresponding Shared Source Working Environment. As a result, it must also have the same directory structure as the common (accessed) part of the corresponding Shared Source Working Environment.

Obsolete file is a file that is located in a project's directory but is not part of any SNIFF+ project. Obsolete files are generated by continuous development and changes to the project structure and should be deleted from time to time in order to keep a project's working environment clean. SNIFF+ Make Support feature offers mechanisms for finding and deleting obsolete files.

Owner is a developer that owns a file or a working environment.

PDF see **Project description file (PDF)**.

Platform is the combination of architecture, vendor and operating system. SNIFF+ executes on all supported platforms. Object working environments are platform-specific. The targets of SNIFF+ projects can be platform-specific. SNIFF+ executes the `sniff_arch` script to determine which platform its running on.

Preferences are the customizable attributes of SNIFF+. SNIFF+ supports user-level and site-level preferences. Most preferences can be edited with the Preferences dialog.

Private Working Environment (PWE) is a directory tree that contains the projects and working files of a single developer. A Private Working Environment is accessible and changeable by only one developer. All check-in and check-out operations work with files in the Private Working Environment. A Private Working Environment must have the same directory structure as the common (accessed) part of the corresponding Shared Source Working Environment.

Projects consists of files, attributes and subprojects. A project is the main organizational element in SNIFF+ and is described by a Project Description File (PDF). Project hierarchies can be built by adding one project to another project, thus creating a superproject-subproject relationship between the two projects.

Project Description File (PDF) is the file that describes a project's attributes, structure and contents. A PDF is a structured ASCII file that is created, saved and opened by SNIFF+. PDFs can also be generated with the `sniff_genproj` batch program.

Project history is the set of all configurations of a project.

PWE see **Private Working Environment (PWE)**.

RCS is a widely used revision control system that is licensed under the GNU public license. SNIFF+ integrates RCS as an underlying version control tool and also supplies it with the package.

Repository contains all Repository files of version-controlled projects. The Repository is typically directly accessed only by the managing CMVC tool and usually stores the different versions in an optimized delta format to save space.

Repository file is the file in the Repository that saves all the complete version tree of a file.

Root directory see **Working environment root directory**.

SCCS is the widely used source code control system that is supplied with most Unix implementations. SNIFF+ integrates SCCS as an underlying version control tool.

Shared file is a source file that is shared among the members of a development team. Shared files are located in a Shared Source Working Environment.

Shared working environment (SWE) is a directory tree that contains the files (source or object) shared in a team. Shared working environments are accessed (shared) among several developers in a team. There are two types of shared working environments: Shared Source and Shared Object.

Shared Source Working Environment (SSWE) is a shared working environment that contains source files only. Typically the platform-specific files are contained in a corresponding Shared Object Working Environment.

SOWE see **Shared Object Working Environment (SOWE)**.

SSWE see **Shared source working environment (SSWE)**.

Symbol is a named language construct in the source code.

Symbol information is extracted from the source files of a project. A project's symbolic information is stored in a Symbol Table, which is saved to disk and transparently managed by SNIFF+.

Symbol Table is the information base that contains information about the declaration, definition and use of named program elements such as classes, methods, variables and functions of a project. Each project has its own Symbol Table that is generated and maintained by the appropriate language parser. Symbol Tables are kept in memory and are persistently stored to disk.

Symbolic link is a symbolic reference to a file in the Unix file system. In contrast to hard links, symbolic links can span different file systems.

Target is the result of a build process. SNIFF+ allows multiple targets to be built in a single project.

Team is a group of software developers working together on a set of projects and sharing a set of common working environments.

Update is the process of checking out all new HEAD versions of projects in a working environment. Typically, updates are done automatically overnight and are followed by automatic builds of all Shared Object Working Environments and Private Working Environments.

VCS is the abbreviation for version control system.

Version is a particular revision and an element of the version tree of a file. A version is created by checking in a working file. The version of a file that you check out is your working file.

Version control is the process of managing and administrating versions of files. The Project Editor in SNIFF+ is the main tool for version control.

Version tree is the hierarchical structure in which all versions of a file are organized. A version tree has one main trunk and can have several branches. The version tree is typically stored in a Repository file.

Working file is a file that has been checked out of the Repository in a working environment (usually in a Private Working Environment). A working file can be directly accessed. Each working file has a corresponding Repository file.

Working Environment is a directory tree that contains projects and working files. SNIFF+ distinguished between private and shared working environments. A shared working environment is accessed among several developers in a team and is overridden by their Private Working Environments. Shared working environments can be split into Shared Source and Shared Object Working Environments in order to separate platform-independent from platform-dependent files. Shared working environments can override other shared working environments, resulting in multiple levels of overriding working environments. The common part of overridden and overriding working environments must have the same directory structure.

Working Environments Administrator is the person who is responsible for the setup, administration and maintenance of working environments. An administrator is informed of the results of automatic updates and builds. Typically, shared working environments are administered by experienced developers with a thorough understanding of all projects that reside in a working environment.

Working Environment root directory is the root directory of a working environment. All root projects that are located in the working environment are subdirectories of the working environment root directory. If many projects need to be managed in a working environment, groups of projects can be located in subdirectories of the working environment root directory.

Index

Symbols

\$LM_LICENSE_FILE 255
\$PATH 254
\$SNIFF_DIR 254

A

Absolute Projects
 default working environment 145
Adaptor 325
Add/Remove Files dialog 206
Adding subprojects 200
Analyzing component information 65
Assignment, filter in Retriever 220
Attributes of Checkmarked Projects 187
attributes of multiple projects 187

B

Batch mode project creation 257
Bean 7
Branches 325
Breakpoints 80
Browser 325
Build 325
Build Options view of the Project Attributes
 dialog 169

C

C++ templates, viewing in the Hierarchy
 Browser 103
Cache
 Project Editor - Use Cache check box 195
Call, filter in Retriever 220
Check In dialog 34
Check Out dialog 35
Check-in 325
Checkmarked projects, definition of 192
Check-out 325
Choose Symbol dialog 30
Class Browser 47
 Class menu 22
 displaying signatures of members 53, 70,

104
Filters 50
 hiding overridden methods 51
History menu 23
Info menu 20
Interface pop-up menu 52
 sorting members alphabetically 70, 104
 using the Inheritance Graph 50
Visibility pop-up menu 52
Class menu 22
CMVC 325
Colors, customizing with Color Picker
 dialog 128
Comparison, filter in Retriever 220
Compilers, and error formats file 288
Concurrent lock 325
Configuration 325
Configuration management 325
Configuration Manager 55
 change sets, displaying 59
 Configuration menu 61
 Differences menu 63
 displaying files and versions 58
 File menu 60
 merging branch configurations 61
 selecting what type of changes to display 59
 using the Change List 57
Configuring Context menu in Project Editor's
 File List 184
Context menus 24
Creating projects in batch mode 257
Cross Reference
 database 138
Cross Referencer 65
 analyzing component information 65
 Class menu 22
 cross referencing function body information 65
 cross referencing interface information 65
 entering the root symbol of the graph 69
 History menu 23
 Info menu 20
 using the *Filter* dialog to set the scope of a
 query 73
 using the Graph view 69
Cross referencing function body
 information 65
Cross referencing interface information 65

Current working environment, updating 202
Custom menus 283

D

Debugger 75
 Class menu 22
 Display menu 79
 Execution menu 78
 History menu 23
 Info menu 20
 Print menu 79
 selecting a process for attaching 81
 Stack menu 80
 using a different debugger back end 77
 using the Breakpoints window 80, 81
Default Configuration 325
default for absolute projects 145
Default working environment
 selecting 144, 145
default working environment 145
Deleting versions in the Project Editor 197
Dependency 326
Derived file 326
Diff/Merge tool 83
 Class menu 22
 Edit menu 16
 File menu 15
 History menu 23
 Info menu 20
 Show menu 18
Differences dialog 36
Directory dialog 38
Displaying change sets 59
Documentation Editor 89
 Class menu 22
 Class pop-up 91
 Documentation View 91
 Edit menu 16
 File menu 15
 History menu 23
 Info menu 20
 modes of operation 91
 Show menu 18
 Status menu 92
 Symbol List 91
 updating documentation
 see Documentation Synchronization Dialog

 updating documentation, *see* Documentation
 Synchronization Dialog

Documentation frames 326
Documentation Synchronization Dialog 93
 Edit menu 96
 Export menu 98
 exporting documentation 98
 Synchronize menu 96
Documentation template 326
Dragging text, in the Source Editor 228

E

Edit menu 16
Editing shortcuts 227
Editor 326
Editor, *see* Source Editor
Emacs 259
Environment variables, needed by
 SNiFF+ 254
Error formats file 288
Exclusive lock 326
Exporting documentation
 in HTML format 99
 in MIF format 98

F

Fast copying, in Source Editor 228
File 326
File dialog 40
File menu 15
File type 326
File types
 adding to existing project 182
 configuring Context menu in Project Editor's File
 List 184
 removing from existing project 182
File Types List 182
File Types view of the Project Attributes
 dialog 182
Files
 checking in using Check In dialog 34
 checking out using Check Out dialog 35
Files, adding/removing to projects 199
Files, deleting versions in Project Editor 197
Filters 11
 Assignment 220

- Call 220
- Comparison 220
- Filter fields in tools 11
- New 220
- Find/Change dialog 30
- Force reparsing source files 201
- Freezing a configuration 326
- Frozen check box 12

G

- General view of the Project Attributes dialog 164
- Generated files 320
 - configuration management cache 321
 - cross reference files 320
 - custom targets file 321
 - lex files 320
 - preprocessor cache files 320
 - Symbol table 320
 - tool status file 321
 - types of 320
- Generated files directory 167
- Group Project Attributes dialog 187
- GUI Builder 7

H

- Having multiple SNIFF+ sessions at the same time 255
- HEAD 326
- Help menu 23
- Help Targets
 - name, defining in the preferences 169
- Help Targets, defining in the preferences 147
- Hierarchy Browser 101
 - Class menu 22
 - Hierarchy menu 72, 105
 - History menu 23
 - Info menu 20
- History 326
- History menu 23
- HP Softbench BMS bridge 274

I

- IDL 293
 - compiling files 295
 - parsing in SNIFF+ 293

- IDL Parser 293
 - compiling IDL files 295
 - using without SNIFF+ 294
- Include Browser
 - Graph menu 111
 - History menu 111
 - Reference menu 111
- Info menu 20
- Inheritance 327
- INIT 327
- Integration, VisaJ 7
- Interfaces 327
- is 329

J

- JAR 7
- Java 7

K

- Keyboard macros in the Source Editor 229
- Keyboard shortcuts
 - in the Source Editor 228

L

- Launch Pad 113
 - Project menu 115
 - Windows menu 117
- Layout handle 12
- License Info dialog 45
- Loading/updating Symbol Table 201
- Lock 327
- Lock dialog 37
- Locking 327
- Log Window 121
- Log window 121
- Looking at the selected working environment 244

M

- Main branch 327
- Make 327
 - using to compile IDL files 295
- Make macro 327
- Make Support File 327
- Makefile 327

Manually loading/updating Symbol Table 201
Matching brackets 227
Merge 327
Merging
 branch configurations in the Configuration
 Manager 61
Meta key, customizing 280
Mouse shortcuts
 copying text by dragging 228
 fast copying in the Source Editor 228
Multiple inheritance 103
Multiple SNIFF+ sessions 255

N

New, filter in Retriever 220

O

Object file 327
Obsolete file 328
Opening projects in Projects dialog 26
Owner 328

P

Parser view of the Project Attributes dialog 179
Parsing
 using SNIFF+ IDL Parser 293
Parsing, forcing a reparse 201
PDF 328
Platform 328
Predefined filters 220
Preferences 123, 328
 customizing Meta key 280
 general SNIFF+ 123
 Preferences dialog 124
Preferences dialog 126
 Appearance view 126
 Cross Referencer view 138
 Documentation Editor view 140
 Editor view 132
 File Types view 153
 New Project Setup view 146
 Others view 160
 Platform view 157
 Retriever view 137
 Tools view 130
 Version Control System view 149

Working Environments view 144
Private Working Environment (PWE) 328
Project
 settings for multiple projects 187
Project Attributes 163
Project Attributes dialog
 Build Options view 169
 File Types view 182
 General view 164
 Parser view 179
 Version Control System view 181
Project description file (PDF) 328
Project Editor 189
 Add/Remove Files dialog 206
 configuring Context menu in File List 184
 Custom menu 283
 deleting file versions 197
 File menu 197
 File Status drop-down 194
 force reparsing source files 201
 Info menu 20
 Project menu 199
 right-click Context menu 24
 right-click context menu in File List 192
 Target menu 19
 typography of entries in File List 190
 updating current working environment 202
Project File dialog 41
Project history 328
Project Tree 192
Projects 328
 adding as a subproject 200
 adding/removing files 199
 creating in batch mode 257
 opening in the Projects dialog 26
Projects dialog
 opening project in 26
Purify, understanding messages from 288
PWE 328
PWE see Private Working Environment
 (PWE) 328

R

Regular expressions 307
 Filter fields in tools 11
 syntax 307
Removing subprojects 200

- Reparsing
 - forcing a reparse 201
 - Repository 328
 - Repository file 328
 - Retriever 209
 - Class menu 22
 - History menu 23
 - Info menu 20
 - Right-click context menus 24
 - Root directory 328
 - Run command, in the Debugger 232
 - Running SNIFF+ without display 256
- S**
- Searching, with the Find/Change dialog 30
 - Selecting default working environment 144
 - Selecting text, in the Source Editor 227
 - Shared file 329
 - Shared Object Working Environment 327
 - Shared Source Working Environment (SSWE) 329
 - Shared working environment (SWE) 329
 - Shell 223
 - Class menu 22
 - Edit menu 223
 - Info menu 20
 - Shell menu 224
 - Target menu 19
 - Shell view 143
 - Shortcuts
 - in the Source Editor 227
 - Show menu 18
 - SitePrefs file 123
 - SNIFF+ sessions, having multiple at the same time 255
 - SNIFF+J 7
 - sniff_arch 256
 - SNIFF_BATCH 256
 - sniff_genproj 257
 - Sniffaccess 259
 - about requests and notifications 261
 - having multiple sessions 261
 - invoking 259
 - notifications 271
 - requests 264
 - Softbench BMS bridge 274
 - Source Editor 18, 225
 - Class menu 22
 - Debug menu 231
 - dragging text 228
 - Edit menu 16
 - fast copying 228
 - File menu 15
 - History menu 23
 - Info menu 20
 - keyboard shortcuts 228
 - matching brackets 227
 - selecting text with the mouse 227
 - Target menu 19
 - Source files, and forcing a reparse 201
 - SOWE 329
 - SSWE 329
 - Statistics dialog 208
 - Status line 12
 - Subprojects
 - adding 200
 - removing 200
 - Super class, quick positioning to in the Source Editor 18
 - Symbol 329
 - Symbol Browser 233
 - Class menu 22
 - Filters 235
 - History menu 23
 - Info menu 20
 - using the Language drop-down 235
 - using the Language pop-up menu 103
 - Symbol information 329
 - Symbol Table 329
 - Symbolic information
 - loading/updating manually 201
 - Symbolic link 329
 - Syntax, for regular expressions 307
- T**
- Target dialog 32
 - Target menu 19
 - Targets 329
 - name, defining in the preferences 147, 169
 - Team 329
 - Tool menu 13
 - Tool status file 321

Typography of File Types List 182

U

Unattended updates, and `SNIFF_BATCH` 256

Undo levels, specifying in Preferences 130

Unix shell, Shell 223

Unlock dialog 37

Update 329

Updating current working environment 202

Updating outside `SNIFF+`, and
`SNIFF_BATCH` 256

Updating Working Environments

without Xserver host 256

User interface basics 11

UserPrefs file 123

Using a different debugger back end 77

V

VCS 329

Version 329

Version control 329

Version Control System view of the Project

Attributes dialog 181

Version tree 330

VisaJ 7

W

Working Environment 330

Working Environment Config. Directory 144

Working Environment root directory 330

Working Environments

files in Config Directory 144

selecting default 144, 145

updating current 202

Working environments

looking at the selected working environment 244

Working Environments Administrator 330

Working Environments tool

looking at the selected working environment 244

saving changes 246

using the menus

File menu 246

History menu 248

User menu 248

Working Environments menu 247

using the Project dialog 26

Working file 330

X

X-Ref

database 138

Z

Zap command, in the Source Editor 229

Colophon

This manual was produced with FrameMaker.

We at TakeFive have tried to make the information contained in this manual as accurate as possible. We cannot, however, guarantee that it is error-free.

© 1992-1999 TakeFive Software GmbH.
All rights reserved.



sniff \ˈsnɪf\ *vb* -ED/-ING/-S

[ME *sniffen*; prob. akin to ME *snivelen* to snivel]

vt (14c)

3: to recognize or detect by or as if by smelling
<German shepherd dogs are parachuted in the
Austrian Alps to *sniff* out survivors of avalanches
— P.T.White>

Webster's Unabridged Third New International Dictionary

