

SNiFF+™

Version 3.2 for Unix and Windows

C Tutorial



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
Road Map	7
The SNIFF+ C Tutorial	7

Part II Browsing

Setting up a Browsing Project	11
The Project Setup Wizard for browsing only	12
The Project Editor	15
Opening the Project Editor	15
The Project Tree - selective project information	16
Tool freezing	17
Browsing Symbols	19
Opening the Symbol Browser	19
Restricting information in Symbol List	20
Displaying signatures of symbols	21
Keyboard navigation in Lists	21
Studying symbol definitions	22
Cross References	25
Opening the Cross Referencer	25
Component browsing – Has-A relationships	26
Finding and editing all the references to a symbol	27
Other features in the Cross Referencer	28
The SNIFF+ Editor	31
Going to a symbol's declaration	31
Symbols with the same name	32
Textual Search with the Retriever	35
Opening the Retriever	35

Finding out where a function is called.	37
Retrieving a string from all projects	37
Where is a structure member referenced?	37
Code Dependencies and Impact Analysis	39
Opening the Cross Referencer	39
Code dependencies	40
Impact analysis: studying function calls	42
Understanding Include Dependencies	45
Opening the Include Browser	45
Files included by a particular file.	46
Files that include a particular file	46
Conclusion	47
Part III Edit/Compile/Debug	
Single-User Project Setup	51
Single-user Project Setup Wizard.	51
Make Attributes and Compilation	55
Setting up C Make Support attributes.	55
Building the executable.	57
Running the application	58
Editing and Compiling	61
Opening and editing a file.	61
Compiling	62
Debugging (Unix only)	65
The Debugger command line	65
Setting Breakpoints.	66
Part IV Team Setup	
Key Concepts	71
Shared projects.	71
Working environments	71
Multi-User Project Setup	75
Preparing the Environment.	75
Multi-user Project Setup Wizard	76

Setting Up the Build System in the SSWE	81
Setting up C Make Support attributes	81
Checking In the project from the SSWE	85
Checking in the project	85
Looking at the history of a file	87
First Build in the SOWE	89
Opening the shared project in the SOWE	89
Building the executable	92
Part V Developing in a Team	
Working in the PWE	97
Opening the shared project in the PWE	97
Check out and check in	98
Adding a new file to a project	100
Removing files from a project	101
Part VI Team Maintenance	
Updating Working Environments	105
Updating the SSWE	105
Updating the SOWE	106
Updating the PWE	107
Freezing the Project in the SSWE	109
Freezing the project	109
Part VII Version Controlling	
File history and locking information	115
Looking at file history information	115
Displaying locking information	116
Configuration and File Differences	119
Looking at file differences with the Diff/Merge tool	120

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.

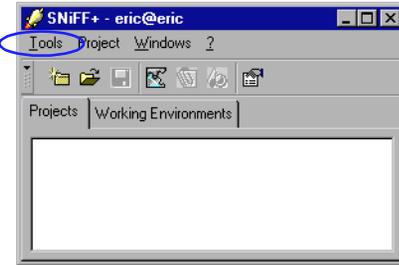
When you start SNIFF+, the first tool that appears is the Launch Pad. In this and other SNIFF+ tools, the first item in the menu bar is for launching tools.

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

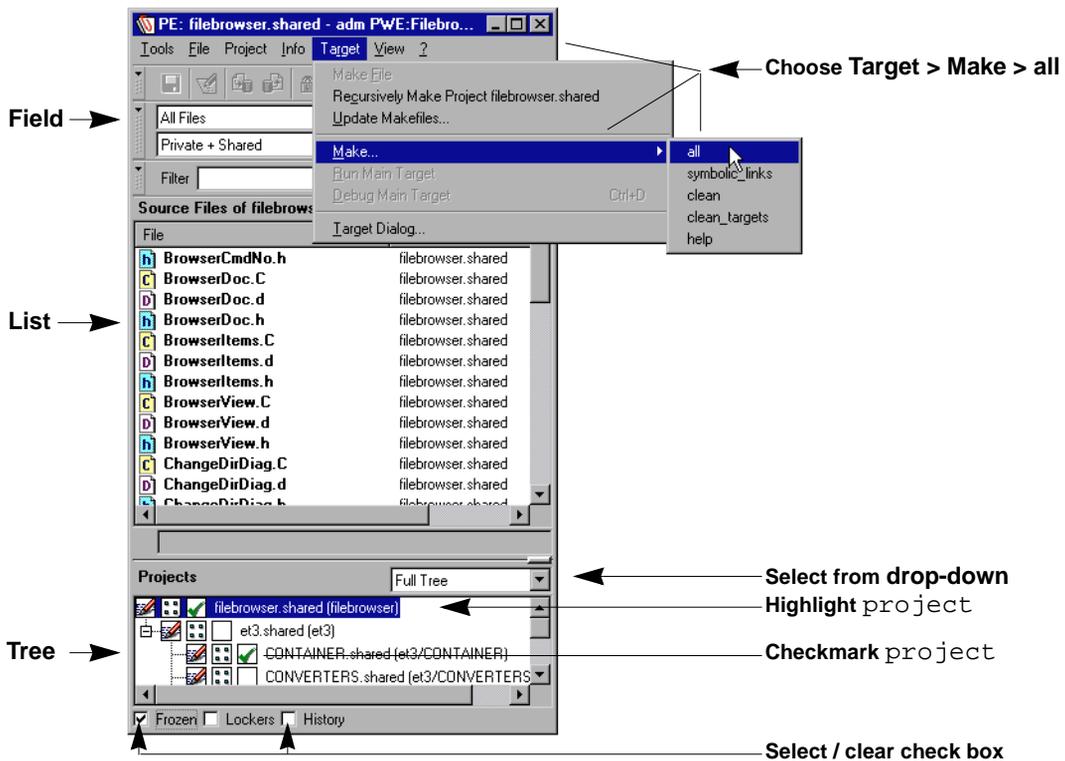
When we refer to this menu in order to launch a tool from the Launch Pad, or any other open SNIFF+ tool, we will use the notation:

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

Introduction

This manual introduces the SNIFF+ solution for C development and is centered around 6 tutorials, see [The SNIFF+ C Tutorial — page 7](#).

Each of the tutorials focuses on different SNIFF+ tools, tasks and concepts. Although each consecutive tutorial and chapter is in itself more or less modular, it is assumed that you are familiar with what has gone before.

Throughout this manual, you will be using a C **example code** called Pico, a text editor developed at the University of Washington in Seattle.

What this manual is not

This manual is not an exhaustive guide to SNIFF+, nor will it teach you C.

The SNIFF+ C Tutorial

The SNIFF+C Tutorial Guide consists of the following parts:

- Browsing
- Edit/Compile/Debug
- Team Setup
- Developing in a Team
- Team Maintenance
- Version Controlling

Note

Please note that a Log Window, displaying SNIFF+ error and control messages, may appear at several stages throughout this tutorial.

Browsing

This tutorial is for you if

- you are a new SNIFF+ user
- you want to quickly learn how to use SNIFF+ for browsing C code

Edit/Compile/Debug

This tutorial is for you if

- you want to use SNIFF+ in single-user/single-platform C development
- you want to learn about building C executables
- you want an introduction to the tools used in the C edit/compile/debug cycle

Note that this tutorial introduces concepts and tools used in developing, irrespective of whether you are working alone or as part of a team.

Team Setup

This tutorial is for you if

- you have done the previous tutorials or
- you are familiar with SNIFF+ and
- you need SNIFF+ in a multi-user and/or a multi-platform work situation
- you need SNIFF+ and RCS (included in the SNIFF+ package) for configuration management and version controlling (CMVC)
- you are responsible for setting up projects and working environments in a multi-user/multi-platform work situation (*Working Environments Administrator*).

Developing in a Team

- you have done the previous tutorials or
- you are familiar with SNIFF+ and
- you work in a team and use RCS for version controlling and configuration management. Note that the Edit/Compile/Debug cycle is described in the Edit/Compile/Debug tutorial

Team Project Maintenance

This tutorial is for you if

- you are responsible for maintaining projects and working environments in a multi-user/multi-platform work situation (*Working Environments Administrator*).

Version Controlling

This tutorial is for you if

- you have done the previous tutorials and
- you need SNIFF+ and RCS (included in the SNIFF+ package) for configuration management and version controlling (CMVC)

Part II

Browsing



Setting up a Browsing Project

In this chapter, you will:

- set up a project for browsing only

You can follow this process, analogously, for any software projects you may want to browse with SNIFF+.

We assume you have successfully installed SNIFF+, and know how to start it. If not, please refer to the *Installation Guide*.

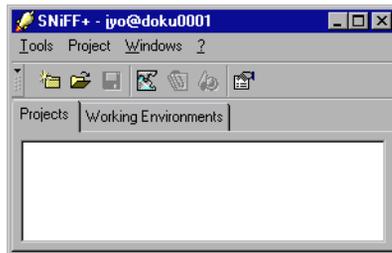
Copying the example

Copy the directory `<your_sniff_installation_dir>/example/c/pico_dir`, including subdirectories, to a directory for which you have write permissions.

We will refer to the full path of this directory as `<PICO_DIR>` in the rest of this tutorial.

- Start SNIFF+.

The Launch Pad appears.



The Project Setup Wizard for browsing only

- To start the Project Setup Wizard, in the Launch Pad, choose **Project > New Project > with Wizard...**

In the Project Setup Wizard

The Wizard starts by asking you to select how you intend to use the new SNIFF+ Project.

- Select **Browsing-only Setup**, and press **Next**.

The “Select file types” page appears.

In the “Select file types” page

- Select **C/C++** and press **Next**.

Note that, after project setup, you can add new standard file types (like the ones in the “Additional File Types Column”), or create and add your own.

In the “Specify project location and name” page

1. Press the **Browse** button next to the **Source code root directory** field and navigate to `<PICO_DIR>/user/pico`.

This is the root directory you need.

2. Double-click on the `<PICO_DIR>/user/pico` directory and press **Select**.

SNIFF+ sets the path and gives the project the default name `pico`.

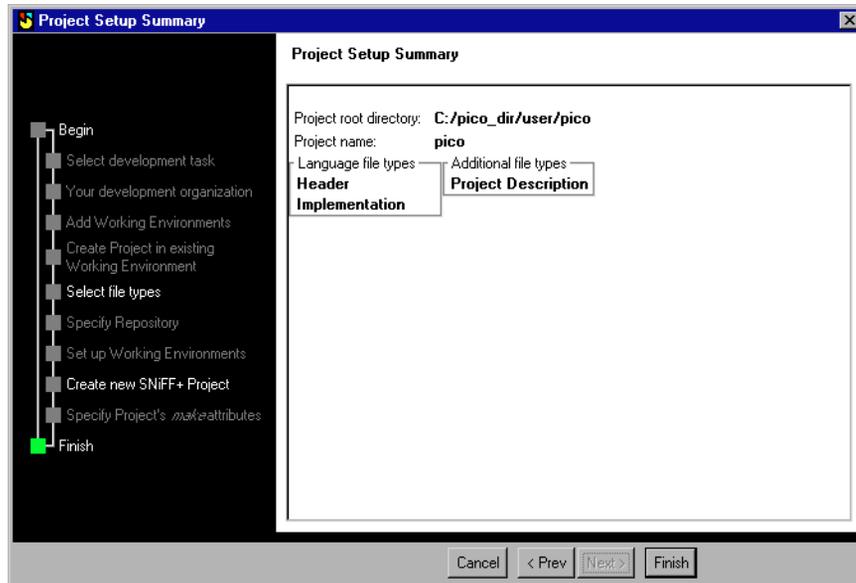
By default, **Create Subprojects** is enabled in the Wizard. This means that SNIFF+ will automatically create subprojects for all the subdirectories of `<PICO_DIR>/user/pico`.

3. Press **Next**.

In the “Project Setup Summary” page

This page summarizes your specifications for the new SNIFF+ C Project.

1. Make sure that your Project Setup Summary page is similar to the following. If it isn't, please go back to the beginning of the Wizard and start again.



If the information on the Project Setup Summary page conforms to the illustration:

1. Press **Finish**.

SNIFF+ will now create the `pico` project and all its subprojects.

2. In the dialog that appears asking if you want to generate cross reference information, press **No**.

Cross Reference information will be automatically generated when we open the Cross Referencer later on.

When SNIFF+ is finished, it opens the new project and displays its structure and contents in a Project Editor, which should look like the one shown at the beginning of the following chapter.

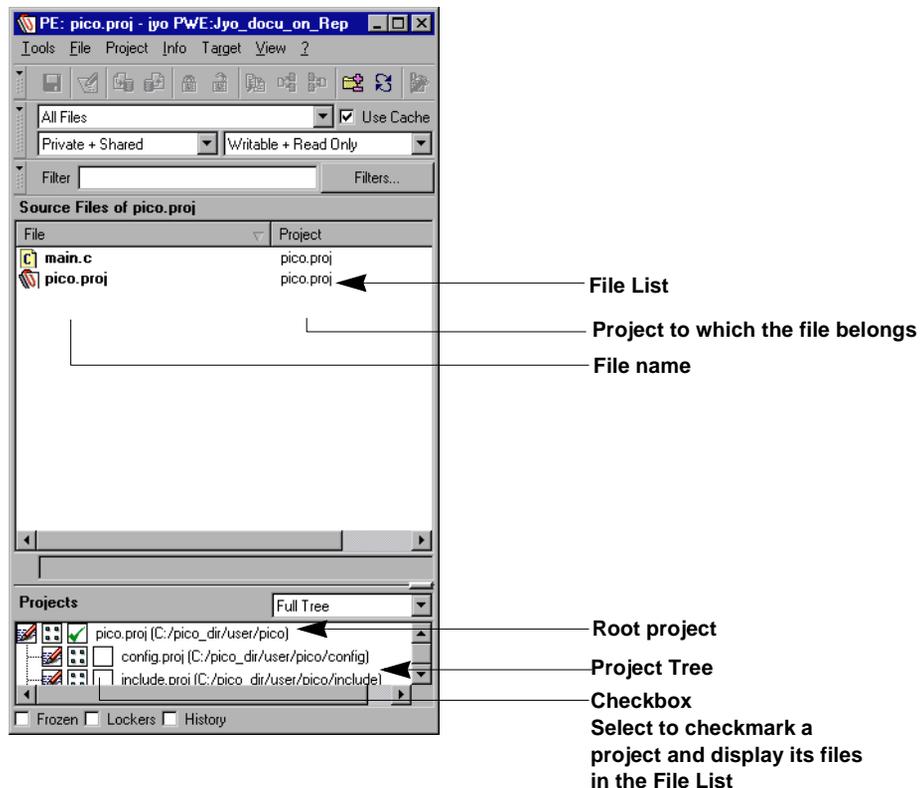
The Project Editor

This chapter is about

- tool handling features and shortcuts common to most SNIFF+ tools
- project and file filtering in the Project Editor

Opening the Project Editor

- The Project Editor is opened automatically when you create a new project.



The Project Tree - selective project information

In the Project Editor, as in other SNIFF+ tools that have such a tree, the Project Tree shows the hierarchical structure of all the subprojects that make up the project.

Checkmarking Projects

Files in the File List can be shown / hidden by clicking in the checkboxes (left of the project names). In SNIFF+, a project that has a checkmark in its checkbox is called a *checkmarked project*.

1. In the Project Tree, checkmark `lib.proj` and notice what happens in the File List.
All the files in the `lib.proj` project are now also shown in the File List.
2. Checkmark `lib.proj` again to deselect it.
The files in this project are no longer shown.

The right mouse button context menu

Checkmarking and deselecting individual checkboxes by mouse click can be useful when you want to include or exclude individual projects from actions that you perform.

For more global manipulation in the Project Tree, the right mouse button **Context menu** is usually more effective.

Right-click context menus in views within a tool are available throughout SNIFF+. These context menus offer the most commonly used commands within a given context. When an item within a view is highlighted (by a left click on its name), the context, and therefore also its menu, often changes.

Selecting from all projects

1. Right-click anywhere in the Project Tree.
2. Choose **Context menu > Select From All Projects**.
All the projects in the Project Tree are now checkmarked, and so all the files of all the projects are now displayed in the File List.

Selecting from only one project

1. Highlight `pico.proj` by left-clicking on its name (not its checkbox).
2. Right-click anywhere in the Project Tree.
3. Choose **Context menu > Select From pico.proj Only**.
Only the files of `pico.proj` are displayed; the files of all other projects are hidden.

Selecting from a tree of projects

Very often, when the project structure gets more complex and contains many subprojects, you will want to view and manipulate a tree of projects like a single project. You can do so by collapsing nodes of the Project Tree. A collapsed node is indicated by a “+” sign. A fully expanded node is shown by a “-” sign.

1. Click on the node of `lib.proj` to collapse it.
2. Try alternately checkmarking and deselecting `lib.proj`.

When the project is checkmarked, all the files in `lib.proj` and its tree of subprojects are listed. Conversely, when the project is not checkmarked, neither its own files, nor any of those in its subprojects, are shown.

3. Click again on the node of `lib.proj`, this time to expand it.

Tool freezing

By default, every SNIFF+ tool is reusable. Whenever a tool is needed, SNIFF+ tries to use an open tool of the type requested to avoid cluttering the screen.

You can, however, freeze any SNIFF+ tool by selecting the **Frozen** check box in the tool's status line. SNIFF+ cannot then reuse the tool to comply with a new request, and has to open a new tool. This can be useful when you want to compare results. To see this:

1. Select the **Frozen** check box in the open Project Editor.
2. Choose **Tools > Project Editor**.

A new Project Editor is opened, leaving your frozen one untouched. Checkmark different projects in the new Project Editor and compare the File Lists.

3. Close the new Project Editor and clear the **Frozen** check box in the open Project Editor.

Browsing Symbols

The Symbol Browser displays all the symbols used in the source files of a project. Symbols can be filtered according to symbol type and other criteria. In SNIFF+, a *symbol* is any C language construct such as a function, struct, variable, etc.

In this chapter you will:

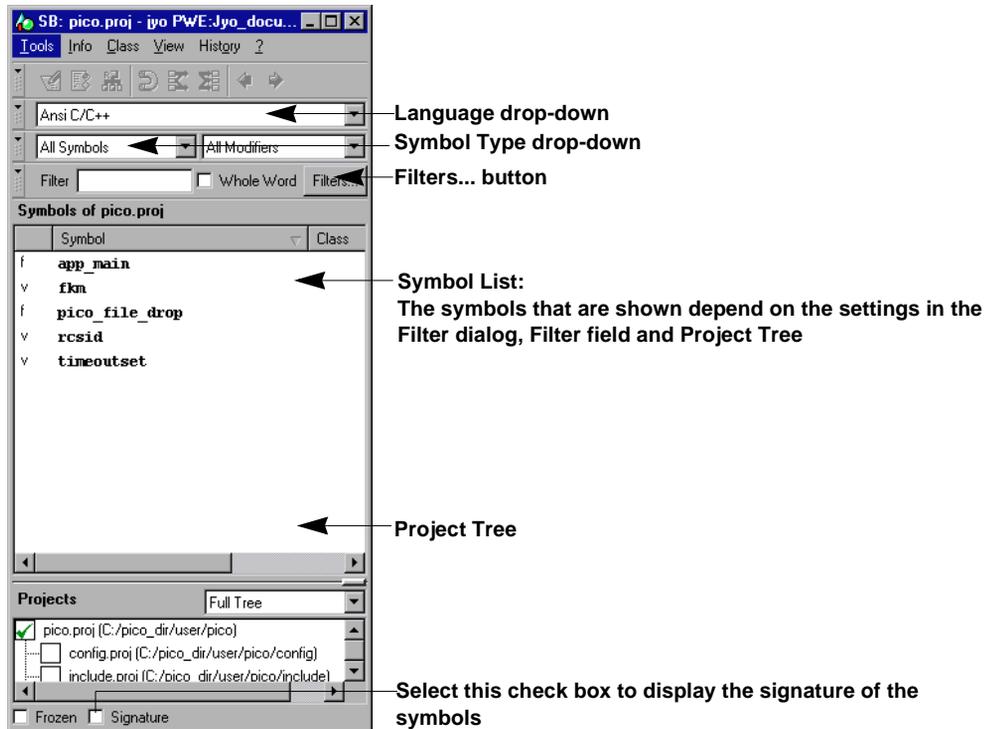
- find out which symbols are defined in the project
- look at which file or project a symbol belongs to
- navigate to a particular symbol
- go to the definition of a symbol

Opening the Symbol Browser

To open the Symbol Browser:

- In the Project Editor, choose **Tools > Symbol Browser**.

A Symbol Browser is opened listing all the symbols in the Pico project. Note that the content of the Symbol List in the Symbol Browser is determined by the **Filters** dialog, the Project Tree and a regular expression matching the names of the symbols.



- Close the Project Editor.

Restricting information in Symbol List

To look at only those symbols defined in a given project, e.g. `lib.proj`:

1. In the Project Tree, highlight `lib.proj` by clicking on its name.
2. Right-click anywhere in the Project Tree, and choose **Context menu > Select from lib.proj Only**.
3. Press the **Filters...** button.
4. In the Filters... dialog that appears, select the **SymbolTypes** tab.

As you can see, all the listed symbols are selected, which means all symbols in all checkmarked projects are displayed in the Symbol List.

You will now filter the Symbol List to display only the structs in `lib.proj`.

5. Press the **None** button and then select the **struct** check box.

6. Press the **Ok** button to apply the changes and close the Filter dialog.

The Symbol List in the Symbol Browser now only displays structs in `lib.proj`.

To look at which structs are defined in `lib.proj` and its subprojects:

1. Click on the '-' sign to the left of `lib.proj` to collapse the node.

You can now view and manipulate `lib.proj` and its subprojects as a single project.

2. Click on the '+' sign to the left of `lib.proj` to expand the node.

Displaying signatures of symbols

Currently you see only the names of the various types. By selecting the **Signature** check box in the status line of the Symbol Browser, you can display the complete signature of the listed symbols.

- Select the **Signature** check box, and then drag the side of the window outward until you can see the whole text line in the Symbol List.

The Symbol List now shows the following information:

symbolType **symbolName** Filename Projectname

- `Filename.[c,h]` is the name of the file where the symbol is declared
- `Projectname.proj` is the name of the project containing this file

As an example, notice that struct `bmaster` is declared in file `browse.c` and is contained in project `lib.proj`.

Keyboard navigation in Lists

In each list of any SNIFF+ tool, you can quickly navigate to entries by clicking into the list, then typing the name of the entry you wish to find. Each consecutive keystroke immediately causes the list to position to the next matched entry.

To find the function `pico` in the Symbol List:

1. In the Project Tree, choose **Context menu > Select From All Projects**.
2. Clear the **Signature** check box.
3. Press the **Filters...** button.

In the Filters dialog

1. In the **SymbolTypes** tab, clear the **struct** check box and then select the **function** check box.
2. In the **Modifier** tab, make sure all modifiers are selected.
3. Press **Ok** to apply the settings and close the dialog.

In the Symbol Browser

1. Click into the Symbol List.
2. Press the <p> key.

As you can see, the function `packbuf` is highlighted because it is matched by the `'p'` you entered.

3. Press the <i> key.

The function `pico` is now highlighted, since it is the first symbol matched by `"pi"`.

Notes on keyboard navigation

- You can restart searches by pressing <ESC>.
- If the pressed key does not match any entry, you will be warned by a beep.
- The cursor keys can also be used for navigating in a list.
- Pressing <Return> on a highlighted entry in a list has the same effect as double clicking on that entry, i.e., the highlighted symbol definition will be loaded into a Source Editor.

Studying symbol definitions

Each symbol in the Symbol List is defined somewhere in your source code. The quickest way to get to a symbol definition is to double-click on the symbol name in the Symbol List.

To study the struct `pico_struct` in the Source Editor:

1. Press the **Filters...** button.
2. In the **SymbolTypes** tab, select only the **struct** check box and press **Ok**.

- In the Symbol List, navigate to `pico_struct` and double-click on it.

A Source Editor opens, the source file `pico.h` is loaded and the struct `pico_struct` is highlighted.

```

SE: pico.h [writable] - include.proj - jyo PWE:Jyo_docu_on_Rep
Tools File Edit Show Info Class Target Debug History ?

/*
 * Master pine composer structure. Right now there's not much checking
 * that any of these are pointing to something, so pine must have them pointing
 * somewhere.
 */
typedef struct pico_struct {
    void *msgtext; /* ptrs to malloc'd arrays of char */
    char *pine_anchor; /* ptr to pine anchor line */
    char *pine_version; /* string containing Pine's version */
    char *alt_ed; /* name of alternate editor or NULL */
    char *alt_spell; /* Checker to use other than "spell" */
    char *oper_dir; /* Operating dir (confine to tree) */
    char *quote_str; /* prepended to lines of quoted text */
    int fillcolumn; /* where to wrap */
    int menu_rows; /* number of rows in menu (0 or 2) */
    PATMT *attachments; /* linked list of attachments */
    long pine_flags; /* entry mode flags */
    void (*helper)(); /* Pine's help function */
    int (*showmsg)(); /* Pine's display_message */
    void (*keybinit)(); /* Pine's keyboard initializer */
    int (*raw_io)(); /* Pine's Raw() */
    long (*newmail)(); /* Pine's report_new_mail */
    long (*msgntext)(); /* callback to get msg n's text */
    int (*upload)(); /* callback to rcv uplaoded text */
}

```

Frozen Line: 157 File: pico.h - C:/pico_dir/user/pico/include

- Close the Source Editor.

The Source Editor is discussed in more detail in [The SNIFF+ Editor — page 31](#).

Cross References

The Cross Referencer provides symbol cross reference information across files and projects. All different kinds of cross references are displayed. In addition, it provides a component view (has-a hierarchy) of structs.

In this chapter you will learn more about

- symbol types used as components of a given symbol
- all the symbols that refer to a given symbol (“Referred-By”)

Opening the Cross Referencer

- In the Symbol Browser, highlight `pico_struct`.

Choose **Info > pico_struct Refers-To Components**.

The screenshot shows the Cross Referencer tool interface. The main window displays a tree view of symbols. The `pico_struct` symbol is selected and highlighted in blue. To the right of the tree view is a list of symbols, with `pico_struct (st)` selected and highlighted in blue. Below the symbol list is a list of projects, with `pico.proj` selected. At the bottom of the window, the code view shows the definition of `pico_struct` in a C file, with the struct name highlighted in blue. The interface includes a menu bar, a toolbar, and a status bar at the bottom.

Labels pointing to the interface elements:

- Depth Field
- Symbol List
- Code view

Component browsing – Has-A relationships

You opened the Cross Referencer by choosing **Info > pico_struct Refers-To Components**. The Cross Referencer therefore opens to display all the components of `pico_struct`.

Look at the following node in your Reference view:

```
st pico_struct > H st headerentry
```

This means that:

the **struct** `pico_struct` **Has struct** `headerentry`

The component types that are displayed in this view are primitive C data types (`pc`), type-defs (`td`) and structs (`st`). If a component type is referenced more than once, the number of references is indicated in brackets, e.g., [5].

Now you know what components are used by `pico_struct`. As you just found out, one of these is the struct `headerentry`. You might also like to know if it is referred to by other symbols in the project.

1. In the Reference view, highlight the node:

```
H st headerentry
```

2. Right-click and choose **Context menu > headerentry Referred-By as Component**.

You now see all the symbols that refer to `headerentry`.

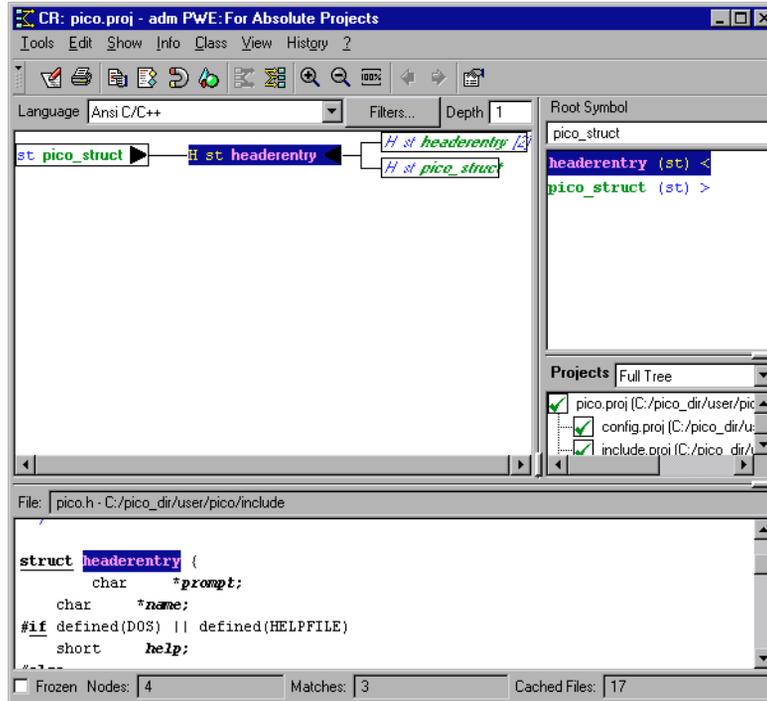
Notice that `headerentry` and `pico_struct` appear in the two nodes pointing to `headerentry`, and that both nodes are displayed in italics. This is because nodes that already exist somewhere else in the Reference view are displayed in italics.

In the Cross Referencer

If you are concentrating on a particular symbol, in our case `headerentry`, you might want to hide distracting information:

1. Make sure `headerentry` is highlighted in the Reference view.

- Right-click in this view and choose **Context menu > Show Restricted Tree of headerentry**.



Finding and editing all the references to a symbol

Situation: You need to make changes in a particular function. Before you start editing, you should know about all the references to the function.

We will start again from the Symbol Browser and select a function to work with.

- To open the Symbol Browser from any tool, choose **Tools > Symbol Browser**.
- Press the **Filters...** button.
- In the **SymbolTypes** tab, select only the **function** check box and press **Ok**.
- In the Symbol List, navigate to and then highlight `update`.
- Choose **Info > update Referred-By**.

In the Cross Referencer, you now see where `update` is referred to by other symbols in project. You also see the number of references in each case.

Notice that the arrowhead now points at the class `update`. This tells you the reference direction.

- Close the Symbol Browser.

Other features in the Cross Referencer

The Root Symbol Field

First, we will re-create the view you started the chapter with:

1. In the **Root Symbol** field, type `pico_struct` and hit <Return>.
2. Choose **Info > pico_struct Refers-To Components**

The Reference view is now the same as the one at the beginning of this chapter.

The Depth Field

Now, we will follow the references starting from struct `pico_struct` to the next deeper level:

1. In the **Depth** Field enter <2>
2. Choose **Info > pico_struct Refers-To Components**.

A refers-to call graph is displayed starting at `pico_struct` and extending to two levels.

The screenshot shows the Cross Referencer interface for the project 'pico.proj'. The 'Depth' field is set to 2, and the 'Root Symbol' field contains 'pico_struct'. The main window displays a call graph starting from 'st pico_struct'. The graph shows references to various components: 'H pc void [5]', 'H pc char [13]', 'H pc int [6]', 'H td PATMT (cl pico_atmt)', 'H pc long [3]', 'H pc short [4]', 'H st headerentry', 'H pc char [13]', 'H pc short', 'H pc int [3]', and 'H st headerentry'. The 'Info' panel on the right shows the 'pico_struct (st)' component selected, with its members listed: 'long (pc)', 'PATMT (td)', 'pico_atmt (st)', 'pico_struct (st)', 'short (pc)', 'unsigned (tud)', 'unsigned short (pc)', and 'void (pc)'. The 'Projects' panel shows the project structure. The bottom status bar indicates 'Frozen Nodes: 18', 'Matches: 3', and 'Cached Files: 17'.

```

typedef struct pico_struct {
    void *msgtext; /* ptrs to malloc'd arrays of char */
    char *pine_anchor; /* ptr to pine anchor line */
    char *pine_version; /* string containing Pine's version */
    char *alt_ed; /* name of alternate editor or NULL */
    char *alt_editor; /* char to use other than "#!" */
};

```

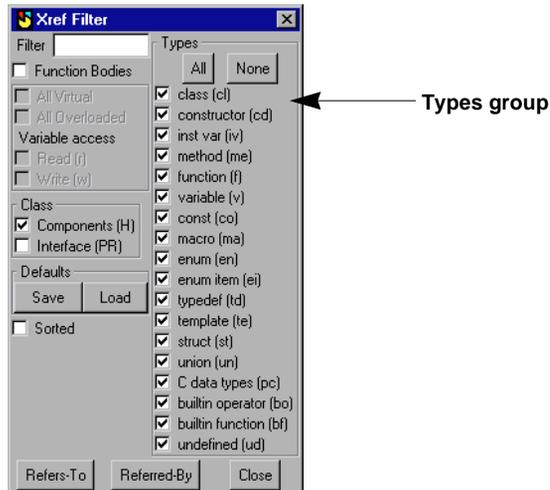
The Xref Filter dialog

Sometimes you may just be interested in looking at a subset of all the references to or from a particular symbol. You can use the Xref Filter dialog to do so.

Let's use the Xref Filter dialog to show only structs and typedefs referred to by `pico_struct` to a depth of 2.

1. Press the **Filters...** button.

The Xref Filter dialog appears.



2. Under **Types**, press the **None** button and then choose `typedef` (`td`) and `struct` (`st`).

Now only these two symbol types are selected for the next cross reference query.

3. Press the **Refers-To** button.

The Reference view now only shows the typedefs and structs found in the query.

Note that the filtering criteria in the Xref Filter dialog now apply to all future queries. To change the parameters of any new queries, you must change the settings in the Xref Filter dialog.

4. Let's look more closely at one of the references in the Reference view. Highlight the node:

```
H td PATMT (cl pico_atmt)
```

5. Choose **Context menu > Show Reference**.

A Source Editor is opened and is positioned at the reference to typedef `PATMT`.

6. Close the Cross Referencer and the Xref Filter dialog. We will continue working with the Source Editor in the next chapter.

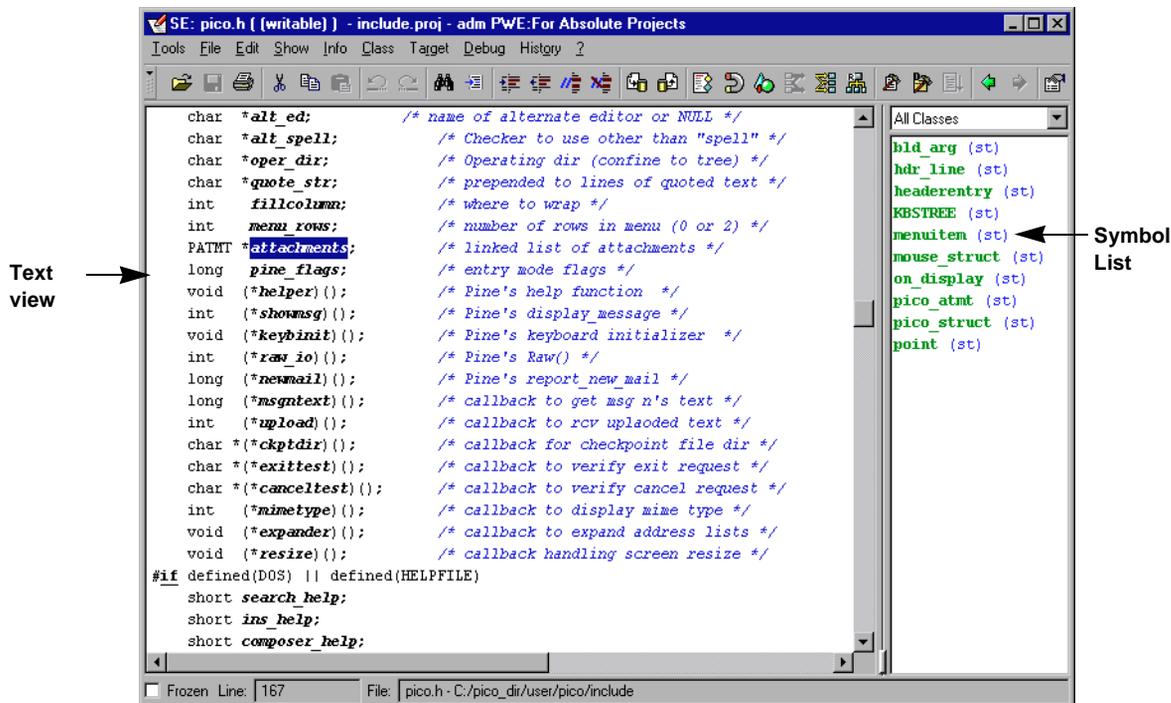
The SNIFF+ Editor

The integrated Source Editor is mouse- and menu-driven. It understands C syntax, provides browsing support and automatically highlights structurally important information, such as struct names, function names and comments. When a source file is modified and saved, its symbol information is immediately updated.

In this chapter you will:

- use the Source Editor to jump to a symbol's declaration
- find out if there are more symbols with the same name as the selected symbol

Continuing from the last chapter, the Source Editor is positioned at the definition of `attachments`, which is a pointer to the typedef `PATMT`.



Going to a symbol's declaration

Let's now find out more about the typedef `PATMT`, which is the symbol type that `attachments` points to. To do so:

1. Double-click on the symbol `PATMT` in the Source Editor's Text view.
`PATMT` is now highlighted.
2. Choose **Show > Symbol(s) PATMT...**
SNIFF+ positions the Source Editor to the declaration of `PATMT`.
As you can see by scrolling up a little bit, `PATMT` is indeed a typedef, representing the struct `pico_atmt`.

Symbols with the same name

1. In the definition of struct `pico_atmt`, notice that one of its fields is the unsigned short `flags`. Let's look to see if this symbol name is unique in the project.
2. In the Text view, double-click on `flags`.
`flags` is highlighted.
3. Choose **Show > Symbol(s) flags....**
SNIFF+ opens the **Choose Symbol** dialog. The **Choose Symbol** dialog opens when there is more than one symbol in the project called `flags`. As you can see, 6 different symbols in the project have the name `flags`.
If there was only one symbol called `flags`, the Source Editor would now be positioned at its declaration.



You may now want to see in which files the symbols are declared.

- Select the **Show listing of files** checkbox.

File and project information are now added to the entries.

You may want to limit the list to show only entries in the current file in the Source Editor (`pico.h`).

1. Enable the **Scan only included files** button .
Now only two entries are shown.

2. Double-click on the second entry in the dialog.

The Source Editor appears and highlights the declaration of `int flags` in the struct `mouse_struct`.

Review

In this chapter, you started from the typedef `PATMT` and browsed its declaration. You then learned that `PATMT` represented the struct `pico_atmt` and then looked at field `flags` of the struct. By choosing the **Show > Symbol(s) flags** command, you determined that the symbol name `flags` was not unique in the project. The last thing you did was to load another declaration of `flags` in the Source Editor.

In the next chapter you'll learn how to use the Retriever - a textual search tool - for browsing. The string that you'll be searching for is "MOUSEPRESS".

Textual Search with the Retriever

The Retriever is a fast source code retrieval tool with filtering. It can be used to obtain information about where a certain string or symbol is used in the source code. It lists all occurrences of strings matching a regular expression in a set of projects. A semantic filter can then be applied to the matches.

The Retriever also allows you to globally find and replace strings in code lines, and to edit code in the integrated Source Editor. For details about these features, please refer to the *Reference Guide*.

This chapter is about using the Retriever to:

- find every line in your Source files containing a given string
- find out where the string is assigned a value
- find out where a string is allocated on a heap

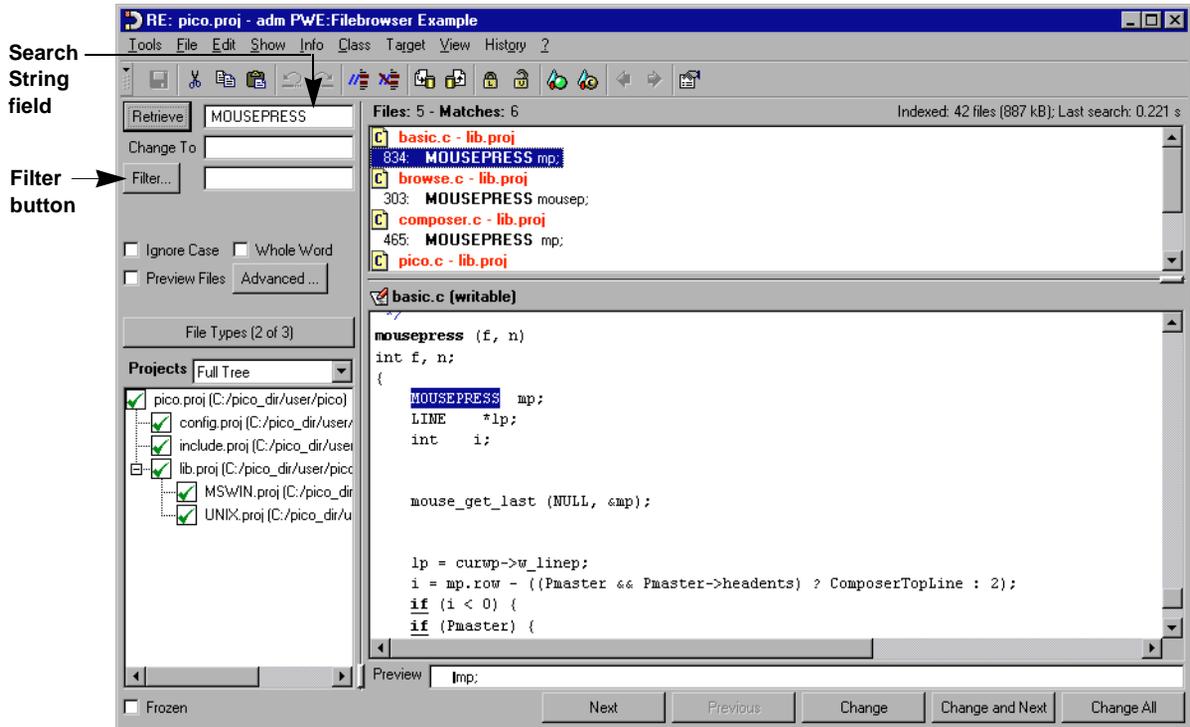
Opening the Retriever

To open the Retriever:

1. In the Text view of the Source Editor, double-click on `MOUSEPRESS`, which is the typedef that represents `struct mouse_struct`.

2. Choose **Info > Retrieve MOUSEPRESS From All Projects.**

The Retriever opens.



3. Close the Source Editor.

When the Retriever first appears after being asked to retrieve the string "MOUSEPRESS" from all projects, notice that:

- In the File Matches List, you are informed that 6 matches were found in 5 files. Each match is listed (in bold print) in its source line context.
- The **Ignore Case** check box is cleared - this means that the search is case-sensitive.
- The **Whole Word** check box is cleared - this means that the compound words with MOUSEPRESS as a substring are also retrieved.
- All the projects in the Project Tree are selected. This is because you opened the Retriever with the command: **Info > Retrieve MOUSEPRESS From All Projects.**

1. Select the **Ignore Case** check box.
2. Press the **Retrieve** button.

In the File Matches List, you are now informed that 15 matches were found in 7 files.

Finding out where a function is called

You may be interested in knowing where any functions among the list of matches are called. You can do so using predefined regular expressions filters in the Find and Replace Filters dialog.

As an example, let's see where function `mousepress` is called. To do so:

1. In the Retriever, highlight the first match in the list of matches. It should be

```
831: mousepress ( f , n )
```

2. Press the **Filter...** button.

The Find and Replace Filters dialog appears.

3. From the list of regular expression filters, select **call** and press **Ok**.

Now the list of matches only displays those matches in which function `mousepress` is called. You should notice that there are 5 matches in 4 files.

Retrieving a string from all projects

The Retriever is a very powerful tool for formulating fuzzy queries.

Let's try this out by getting information about menu handling. To do so:

1. Clear the **Filter** field.

2. In the **Search** field type `menu`, then press **Retrieve**.

As you can see, there are 518 matches in 24 files.

Note

After the first retrieve, the source code is cached and all further queries are much faster. You can switch off caching in your Preferences.

Let's further restrict the search using the assignment filter:

1. Press the **Filter...** button.

The Find and Replace Filters dialog appears.

2. From the list of regular expression filters, select **assignment** and press **Ok**.

Notice that there are 26 matches in the project where a variable called `menu` or similar is assigned a value.

Where is a structure member referenced?

Continuing with the substring "menu", let's look at all the references to structure members that have this substring in their names. To do so:

1. Press the **Filter** button.

The Find and Replace Filters dialog appears.

2. From the list of regular expression filters, select **>MEMBER** and press **Ok**.

Notice that there are 24 matches in the project with references to structure members that have “menu” in their names.

Review

Although the Retriever can help you search for strings in your source code, it can only provide limited cross reference information. In the next chapter you will be working with the Cross Referencer to get more cross reference information about one of the matches of the string “MOUSEPRESS”.

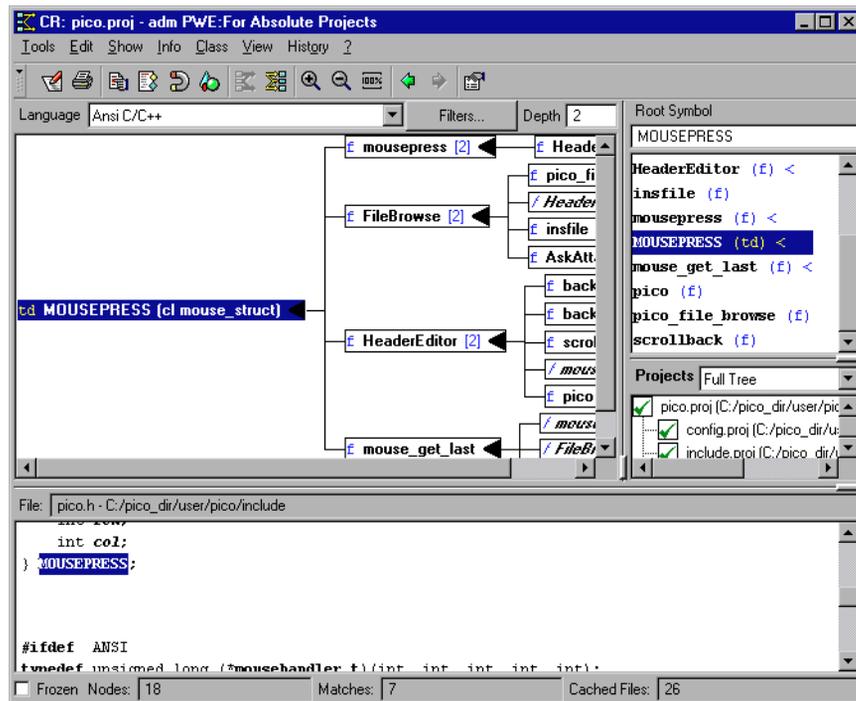
Code Dependencies and Impact Analysis

This chapter is about using the Cross Referencer to:

- look at code dependencies
- look at where symbols are called

Opening the Cross Referencer

1. In the Retriever, clear the **Filter** field and then enter **MOUSEPRESS** in the **Search** field.
2. Press **Retrieve**.
3. Highlight the second match in the list of matches. It should be
834: **MOUSEPRESS** mp
4. Choose **Info > MOUSEPRESS Referred-By**.
The Cross Referencer opens.



5. Close the Retriever.

Code dependencies

As you can see in the Reference view, `MOUSEPRESS` is taken as the root symbol and symbols that refer to it are displayed as its nodes. One of these symbols is the function `mousepress`. Let's take a closer look at the symbols that `mousepress` refers to.

1. In the Reference view, highlight node

```
f mousepress [2]
```

2. We want this node to be the root node of the next query, so right-click in the Reference view and choose **Context menu > Start from mousepress**.
3. Right-click and choose **Context menu > mousepress Refers-To**.

The Reference view now shows all symbol types that function `mousepress` refers to. Let's filter the view to show only functions, structs and typedefs. To do so:

1. Press the **Filters...** button.

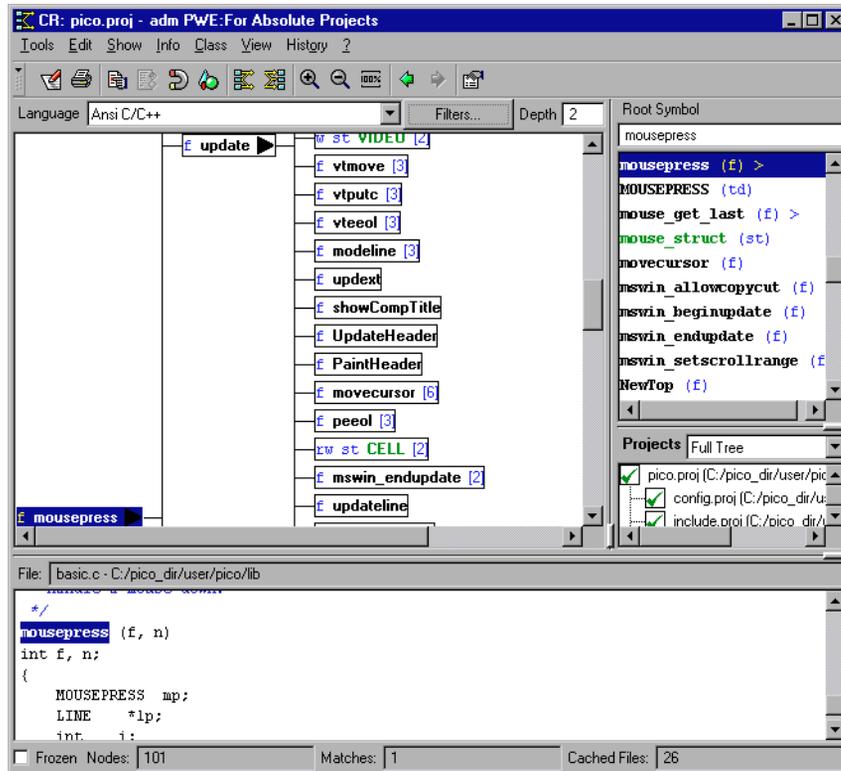
The Filter dialog appears.

2. Under **Types**, press the **None** button and then choose `function (f)`, `typedef (td)` and `struct (st)`.

Now only these three symbol types are selected for the next cross reference query.

3. Press the **Refers-To** button.

The Reference view now only shows functions, typedefs and structs referred to by `mousepress` to a depth of 2.



Impact analysis: studying function calls

You may want to see where the function `ToggleHeader` is called:

1. Highlight `f ToggleHeader [2]` in the Reference view.

Note that the `[2]` that appears after the function name means that `ToggleHeader` is called twice in function `update`.

2. Choose **Context menu > ToggleHeader Referred By**.

As you can see in the illustration below, the backward references of `ToggleHeader` are added to the graph (you may have to resize the Reference view a bit). Remember that you selected only functions, structs, and typedefs in the Xref Filter dialog, so only matching symbols of these types are shown.

The screenshot displays the CR IDE interface. The main window shows a reference graph for the function `ToggleHeader`. The graph is structured as follows:

- `update` (function) calls `ToggleHeader [2]` (function).
- `ToggleHeader [2]` calls `scrollfrow` (function), `scrollback` (function), and `scrollto [2]` (function).
- `scrollfrow` calls `forwpage` (function) and `scrollto` (function).
- `scrollback` calls `backpage` (function) and `scrollto` (function).
- `scrollto [2]` calls `pico_scrollto` (function).

The interface includes a menu bar (Tools, Edit, Show, Info, Class, View, History), a toolbar, and a language dropdown set to "Ansi C/C++". A "Filters..." button and a "Depth" dropdown set to "2" are also visible. On the right, a "Root Symbol" pane shows a list of symbols, with "ToggleHeader (f) <" selected. Below this, a "Projects" pane shows a tree view of project files. At the bottom, a code editor shows the definition of `ToggleHeader (show)`:

```

/*
ToggleHeader (show)
int show;
{
    /*
     * check to see if we need to display the header...
     */
}

```

At the bottom of the IDE, there are status indicators: "Frozen: Nodes: 110", "Matches: 0", and "Cached Files: 26".

Note

Try and work with a **Depth** of 1 wherever possible. Setting a large value for the **Depth** can lead to huge graphs. This is not very informative and can take a very long time.

Let's now look directly at all references to `ToggleHeader`. To do so:

1. `<SHIFT>`click on `f_scrollforw`.

The Code view is positioned to the first reference.

2. `<SHIFT>`double-click on `f_scrollforw`.

A Source Editor appears, and the first reference is now displayed in it.

3. Position the Source Editor and the Cross Referencer on your screen so that you can see both.

4. In the Source Editor, choose **Show > Next Match**.

The Source Editor is positioned at the next reference, and the node `f_scrollto` is now highlighted in the Cross Referencer.

5. To visit all locations, repeat the previous step until there are no more matches.

Understanding Include Dependencies

The Include Browser graphically displays include references made in project source files. It can be used 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.

This chapter is about using the Include Browser to:

- find out which header files are included by a particular implementation file
- find out which implementation files include a particular header file

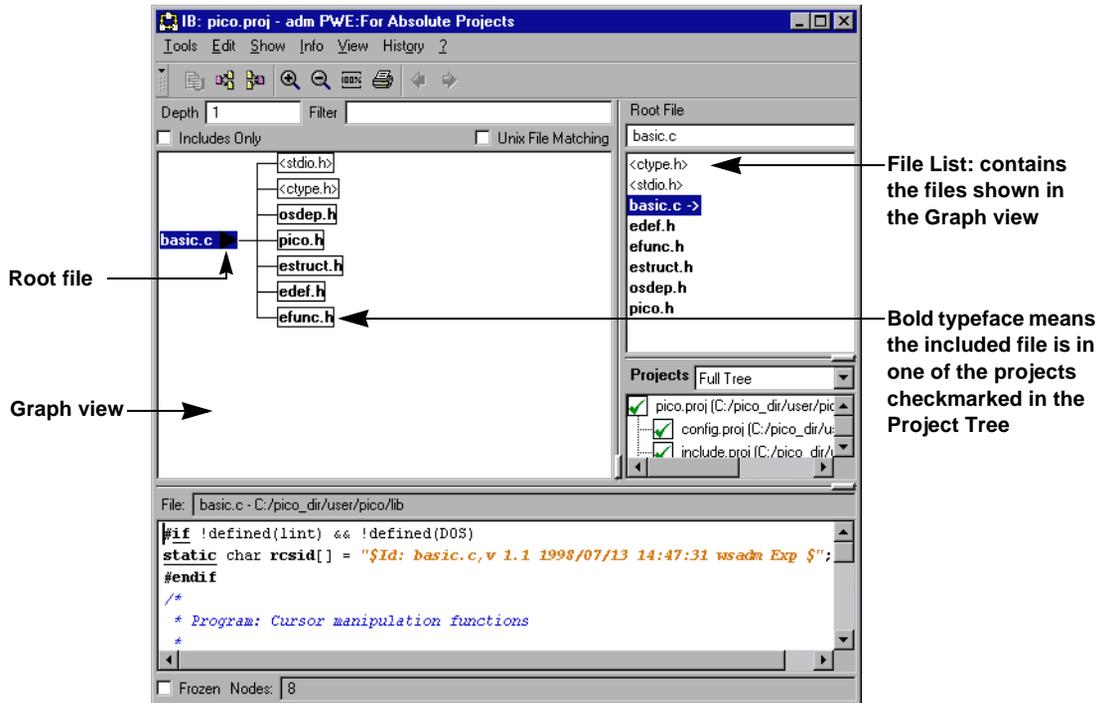
Opening the Include Browser

To open the Include Browser:

1. In the Cross Referencer, highlight the root symbol `f mousepress`.
2. Choose **Show > Implementation of mousepress**.

The Source Editor is positioned at the implementation of `mousepress` (in the file `basic.c`).

- In the Source Editor, choose **Info > basic.c Includes**.
The Include Browser opens.



- Close the Source Editor and the Cross Referencer.

Files included by a particular file

As you can see in the above illustration, `basic.c` includes seven header files. Let's look at the include statement of one of the header files. To do so:

- Highlight `pico.h` in the newly created tree.
- Choose **Context menu > Show Include Statement**.

The Source Editor opens and is positioned at the include statement in `basic.c`.

- Close the Source Editor.

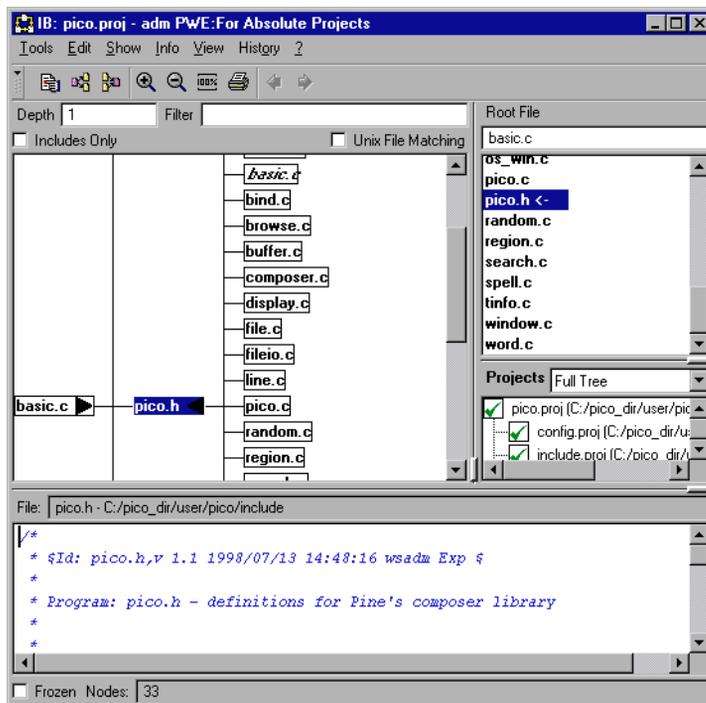
Files that include a particular file

Let's see which files include the header file `pico.h`. To get this information:

- In the Include Browser, make sure that `pico.h` is highlighted in Graph view.

2. Choose **Context menu > Included-By**.

In the illustration below, you can see all the files which include `pico.h`.



Let's now look at which projects these files belong to. To do so:

- Choose **View > Show Project Name**.

Scroll to the right of the Graph view. You can now see the file names as well as the projects to which they belong.

Lastly, let's look at the header file `pico.h`:

1. Make sure that `pico.h` is highlighted in the Graph view.

2. Choose **Show > pico.h**.

The Source Editor opens.

3. Close all open SNiFF+ tools except for the Launch Pad.

Conclusion

This was the last of the SNiFF+ browsing tools to be introduced in this tutorial.

In this part of the C Tutorial you were introduced to:

- Opening the project
- Working with the following SNiFF+ tools:
 - Symbol Browser

- Cross Referencer
- Source Editor
- Retriever
- Include Browser

In the tutorial we can only give you a few hints and tips to help you on your way. To get a better understanding of the tools, we recommend experimenting. To really appreciate what SNIFF+ can do for you, you really need to work with it.

Deleting the project

You will no longer be working with the browsing-only project that you set up in this part of the C Tutorial, so you can delete it at this time. If, however, you would like to keep the project for browsing at a later time, we suggest that you skip the following instructions.

To delete the project:

1. In the Launch Pad, delete the project by highlighting it and choosing **Project > Delete Project pico.proj**.

A dialog appears, in which SNIFF+ asks you if it should delete SNIFF+ - related files and directories.

2. Select the **Repeat** check box and press **Delete**.

With the **Repeat** check box selected, SNIFF+ will also delete SNIFF+ - related files and directories in all subprojects of `pico.proj`.

What's next

The only SNIFF+ tool that should now be open is the Launch Pad.

The next part of the C tutorial introduces you to

- Editing, compiling and debugging with SNIFF+

Part III

Edit/Compile/Debug



Single-User Project Setup

This chapter introduces you to setting up a single-user/single-platform project without version controlling. Multi-user/multi-platform projects using RCS for configuration management and version controlling (CMVC) are described in the next tutorial, “Working in Teams”.

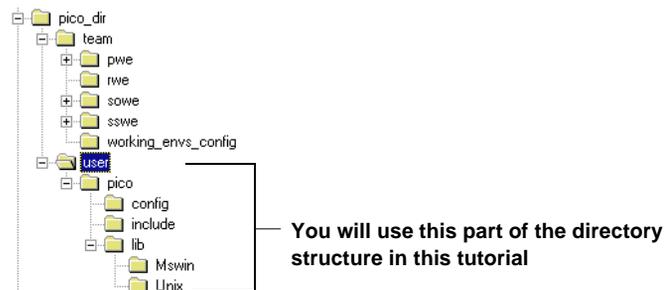
This chapter is about

- using the Project Setup Wizard for setting up a SNIFF+ single-user/single-platform project for C development.

If you aren't continuing on from the [Browsing](#) tutorial

We assume you have successfully installed SNIFF+, and know how to start it. If not, please refer to the *Installation Guide*.

1. In the Browsing tutorial, you copied the directory `<your_sniff_installation_dir>/example/c/pico_dir`, including subdirectories, to a place where you have write permissions. If you haven't done so, please do so now. You should have the following directory structure:



In the rest of this tutorial, we will use `<PICO_DIR>` to refer to the complete path to this directory.

2. Start SNIFF+.

The Launch Pad appears.

Single-user Project Setup Wizard

- To start the Project Setup Wizard, in the Launch Pad, choose **Project > New Project > with Wizard....**

In the Project Setup Wizard

The Wizard starts by asking you to select how you intend to use the new SNIFF+ Project.

- Accept the default selection, **Standard Setup**, and press **Next**.

The “Select development task” page appears.

In the remaining steps, we will refer to the names of Wizard pages. You can find a page’s name in the title bar of the Wizard.

In the “Select development task” page

- Select **Create a new SNIFF+ Project from scratch** and press **Next**.

In the “Your development organization” page

This tutorial is for single-user/single platform development without CMVC, so:

- accept the defaults (**No/No/None**) and press **Next**.

In the “Select file types” page

- Select **C/C++** and press **Next**.

Note that, after project setup, you can add new standard file types (like the ones in the “Additional File Types Column”), or create and add your own.

In the “Specify Private Working Environment” page

You are asked to specify your *Private Working Environment* (PWE) root directory.

1. Press **Browse**, and in the Directory dialog, navigate to <PICO_DIR>/user and double-click on it and then press **Select**.
2. In the **PWE name** field, enter a name for the PWE, e.g., `pico`.

Notice that your username is entered next to the enabled **Owner** button. SNIFF+ needs your username to correctly handle permissions. Being the owner of the PWE means that you are the only one who is allowed to modify the working environment’s attributes.

3. Press **Next**.

In the “Create New SNIFF+ Project” page

SNIFF+ has set your **Project root directory** to <PICO_DIR>/user. Also by default, **Create Subprojects** is enabled.

1. Press **Browse...**
2. In the dialog that appears, select `pico`, press **Open** and then **Select**.

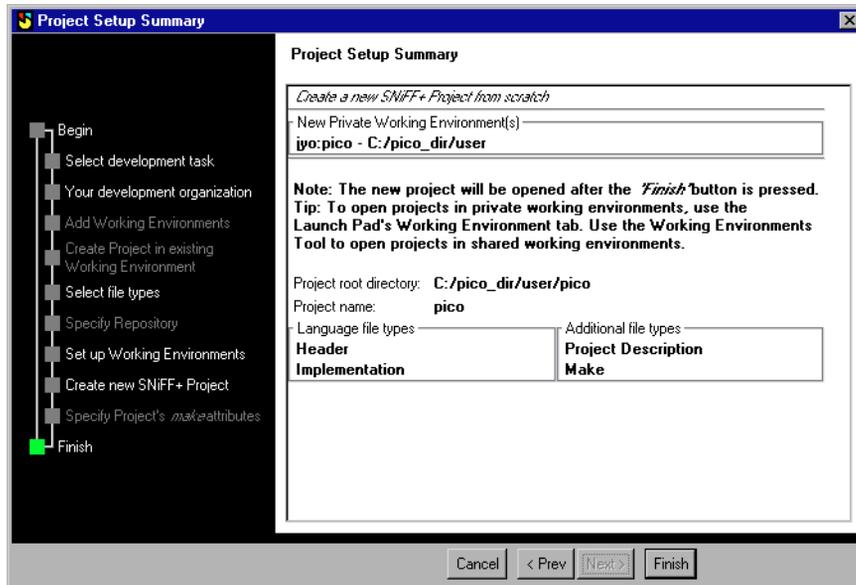
`pico` is entered in the Project name field. We suggest that you accept the default project name.

3. Select the **Use SNIFF+’s Makefiles** checkbox.
4. Press **Next**.

In the “Project Setup Summary” page

This page summarizes your specifications for the new SNIFF+ C project and required Working Environments.

- Make sure that your Project Setup Summary page is similar the following. If it isn't, please go back to the beginning of the Wizard and start again.



- Press **Finish**.

SNIFF+ will now create the new `pico` project and all its subprojects.

- In the dialog that appears asking if you want to generate cross reference information, press **No**.

When SNIFF+ is finished, it opens the new project and displays its structure and contents in the Project Editor.

Note

If you did not delete the browsing-only project in the last part of the C Tutorial ([Deleting the project — page 48](#)), the Project Description Files of the browsing-only project and its subprojects will appear in the File List of the Project Editor (with extension `.proj`). You can safely ignore these files.

Make Attributes and Compilation

This chapter is about

- setting C Make attributes for a single-user project
- building and running the executable

Setting up C Make Support attributes

1. In the Project Tree of the Project Editor, checkmark `pico.shared`, `lib.shared`, `MSWIN.shared` on Windows and `UNIX.shared` on Unix.
2. Choose **Project > Attributes of Checkmarked Projects...**

The Attributes of Checkmarked Projects dialog appears. You can use this dialog to set/modify the attributes of multiple projects at the same time.

In the Attributes of Checkmarked Projects dialog

Setting up Make Support for `pico.shared`

1. Under the **Build Options** node, select **Project Targets**.
2. Make sure that **pico** is highlighted in the Project List.
3. In the **Executable** field of the Ansi C/C++ tab, enter `pico` on Unix and `pico.exe` on Windows. This will be the name of the project's executable.
4. Under the **Build Options** node, select **Directives**.
5. Select the checkbox to the right of the Generate button and press the **Set For All** button.
This attribute is now set for all projects.
6. Under the **Build Options** node, select **Build Structure**.
7. In **Recursive Make Dirs** field, enter **lib**.

In order to build the project's executable, SNIFF+ must first build the target of the `lib.shared` subproject (a library). SNIFF+ uses the project information in **Recursive Make Dirs** field to determine the order in which to execute Make.

Setting up Make Support for `lib.shared`

1. Highlight **lib** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Library** field of the of the Ansi C/C++ tab, enter `libpico.a`. This will be the name of the library built in this project.
4. Under the **Build Options** node, select **Build Structure**.

5. From the **Passed to Superproject** drop-down, choose **Library**.

The project's library is exported to `pico.shared` and is used to build the Pico executable.

6. In **Recursive Make Dirs** field, enter **UNIX/MSWIN** on Unix/Windows.

The library is built using objects built in the platform-specific subdirectory of `lib.shared`.

Setting up Make Support for MSWIN.shared (Windows only)

1. Highlight **MSWIN** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Relinkable Object** field of the Ansi C/C++ tab, enter `osdep.o`. This will be the name of the relinkable object built in this project.
4. Press **Ok** to apply the changes to the project attributes.
5. A dialog appears asking you if you want to update Makefiles. You will be updating Makefiles later on, so press **No**.

Setting up Make Support for UNIX.shared (Unix only)

1. Highlight **UNIX** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Relinkable Object** field of the of the Ansi C/C++ tab, enter `osdep.o`. This will be the name of the relinkable object built in this project.
4. Press **Ok** to apply the changes to the project attributes.
5. A dialog appears asking you if you want to update Makefiles. You will be updating Makefiles later on, so press **No**.

Saving changes

- In the Launch Pad, save the changes made to `pico.shared` and its subprojects:
 - Select `pico.shared` in the Project List.
 - Choose **Project > Save Project pico.shared**.
 - In the Alert dialog that appears, press the **Save all** button.

Building the executable

Note

SNiFF+ doesn't have its own compiler therefore you must have a compiler installed on your computer to compile SNiFF+ projects. By default, the gnu compiler is specified on Unix and Microsoft Developer is specified on Windows. If you are using another compiler, it must be specified in your Platform Makefile. For more information, see *User's Guide — Build and Make Support*.

In the Project Editor

Before building, make sure that the projects' Make Support information is up-to-date. Makefiles should be updated whenever structural changes are made to the projects, or when projects are first opened.

Note that SNiFF+ needs to know where to start Make execution. You tell SNiFF+ this by highlighting the appropriate project. In the example project, Make execution starts in `pico.shared`, where you specified the project's executable. So:

1. In the Project Tree, make sure `pico.shared` is highlighted.
2. Select from all projects by right-clicking anywhere in the Project Tree and choosing **Context menu > Select From All Projects**.
3. Choose **Target > Update Makefiles**.

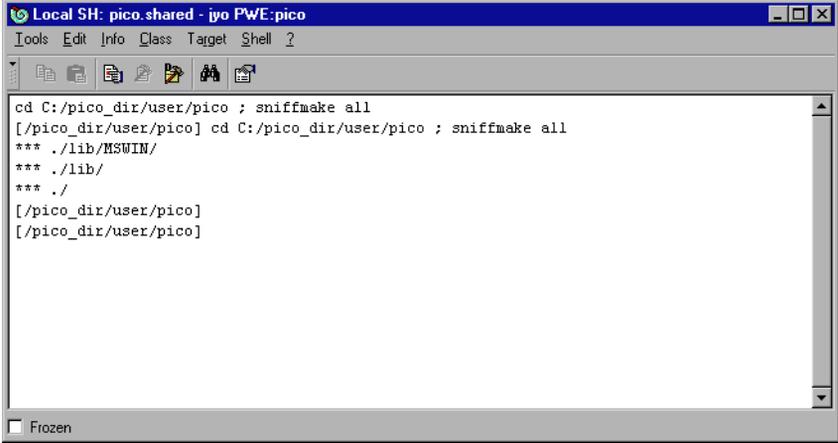
A dialog appears asking you whether the dependencies information should also be updated.



4. Press **Yes**.

5. Choose **Target > Make > all**.

A Shell Tool appears, and at the end of the build it should look similar to this:



```
Local SH: pico.shared - iyo PWE:pico
Tools Edit Info Class Target Shell ?
cd C:/pico_dir/user/pico ; sniffmake all
[/pico_dir/user/pico] cd C:/pico_dir/user/pico ; sniffmake all
*** ./lib/MSWIN/
*** ./lib/
*** ./
[/pico_dir/user/pico]
[/pico_dir/user/pico]
```

If compiler errors are reported in the shell at this stage, something went wrong with the setup of the project's C Make attributes. We recommend that you go through the steps in this tutorial again, carefully check them, compare screenshots, and try compiling again.

Running the application

In the Project Editor

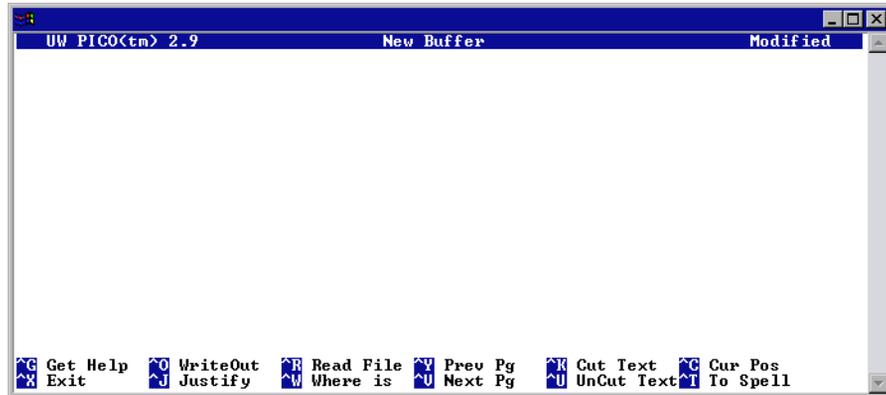
1. Make sure that `pico.shared` is highlighted in the Project Tree.
2. Choose **Target > Run pico**.

The Program Arguments dialog appears:



3. Press **Ok**.

The application is started and appears on your screen:



4. Close the application by choosing **<CTRL> X**.

5. Close the Shell Tool.

Editing and Compiling

This chapter is about

- using the Source Editor
- how to proceed when compiler errors are reported

Opening and editing a file

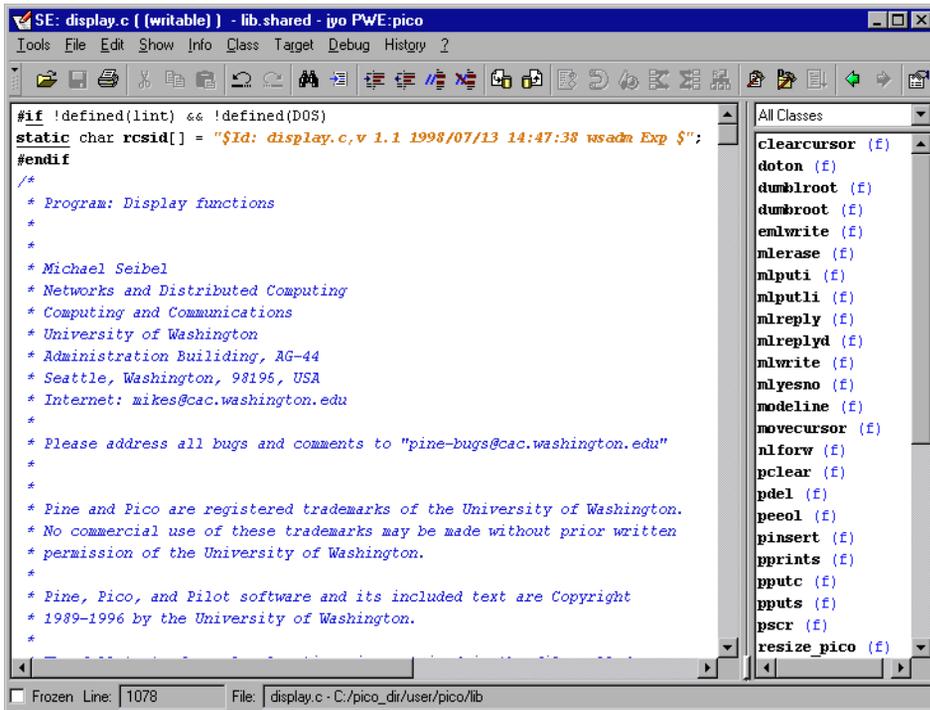
After using the Project Editor to open a file in the Source Editor, you will edit the file so as to induce a compilation error.

After attempting to compile, the error will be reported in the Shell tool. From the error message, you can go straight to the point in the source code where the error was found.

In the Project Editor

1. In the Project Tree, make sure that `lib.shared` is highlighted.
2. Right-click anywhere in the Project Tree and choose **Context menu > Select From lib.shared Only**.

3. In the File List, double-click on `display.c` to open the file in the Source Editor.



Now, to edit the file so as to induce a compilation error:

1. In the Source Editor's Symbol List, click on `clearcursor (f)`.

The Editor is positioned at the `clearcursor` function.

2. To comment out the line, choose **Edit > Comment**.

Note that you can comment out any number of lines by highlighting them and choosing this menu item.

3. For the changes to take effect, choose **File > Save**.

Compiling

1. In the Source Editor, choose **Target > Make > all**.

A Shell Tool appears and SNIFF+ tries to compile the modified file.

The error is reported in the Shell Tool.

- In the Shell tool, click on the reported error (highlighted in the illustration) and choose **Edit > Show Error**.

```

Local SH: pico.shared - iyo PWE:pico
Tools Edit Info Class Target Shell ?
[/pico_dir/user/pico] cd C:/pico_dir/user/pico/lib ; sniffmake all
*** ./MSWIN/
*** ./
make: Making C object display.o ...
display.c
display.c(1082) : error C2449: found '{' at file scope (missing function header?)
display.c(1084) : error C2059: syntax error : ''
gmake[1]: *** [display.o] Error 2
gmake: *** [main_all] Error 2
[!]+ Done ./pico.exe (wd: /pico_dir/user/pico)
(wd now: /pico_dir/user/pico/lib)
[/pico_dir/user/pico/lib]
Frozen

```

The Source Editor is opened at the detected error, and the error line is highlighted. To uncomment the line that induced the detected error:

- Click into the line you commented out earlier and choose **Edit > Uncomment**.
- For the changes to take effect, choose **File > Save**.
- Choose **Target > Make > all** again to re-compile.
- Close the Shell tool and the Source Editor.

Conclusion (Windows only)

This concludes the Edit/Compile/Debug tutorial. In this part of the C Tutorial you were introduced to:

- Setting Make attributes for a C project
- Building a SNIFF+ project and all the subprojects that it depends on
- Using the Shell tool to jump to errors in source files

In the tutorial we can only give you a few hints and tips to help you on your way. To get a better understanding of how to edit and compile projects in SNIFF+, we recommend experimenting. To really appreciate what SNIFF+ can do for you, you really need to work with it.

In the Launch Pad

- Close all open SNIFF+ tools except for the Launch Pad.
- In the Launch Pad, close the `pico.shared` project by highlighting it and choosing **Project > Close Project pico.shared**.

What's next

The only SNIFF+ tool that should now be open is the Launch Pad.

The next part of the C tutorial introduces you to

- Using SNIFF+ for multi-user team projects

Please skip the next chapter and continue with [Key Concepts — page 71](#).

Debugging (Unix only)

In this introduction to the SniffGdb Debugger, you will learn how to:

- start the Debugger
- set breakpoints

On Windows

A Microsoft Developer Studio integration is available on Windows. When you debug a project's executable in SNIFF+, its symbolic debugging information is automatically loaded into Microsoft Developer Studio. To learn about the integration features, please refer to the *User's Guide*. For a description of the debug commands in Microsoft Developer Studio, please refer to the product documentation.

The Debugger command line

After starting the Debugger, you will set two breakpoints and watch what happens to a variable between the first and the second breakpoint.

- Make sure that the right debugger is selected in your **Preferences > Platform view**.

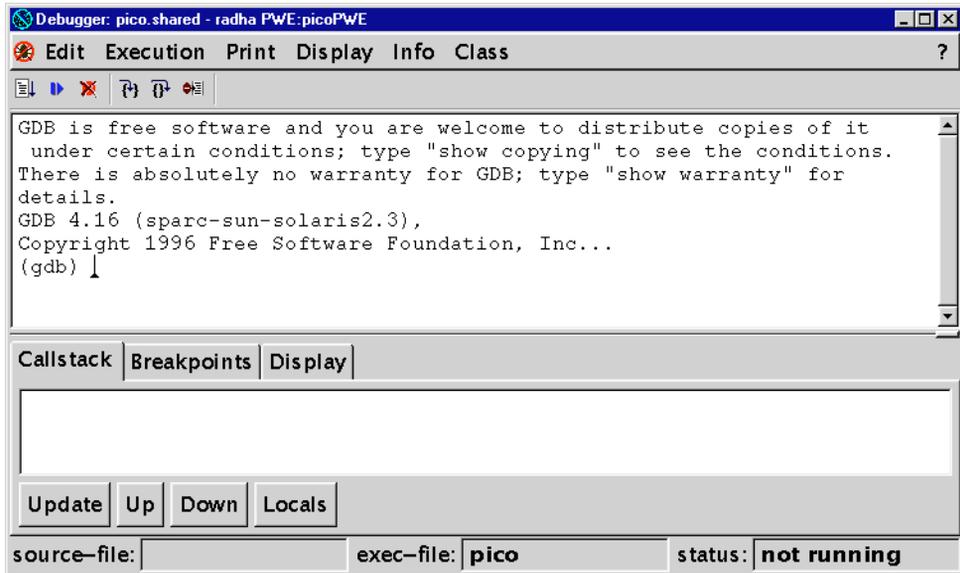
In the Project Editor

To start the Debugger:

1. In the Project Tree, make sure that `pico.shared` is highlighted. If it isn't, click on its name to highlight it.

2. Choose **Target > Debug pico**.

The Debugger opens.



In the Debugger Command Line Shell

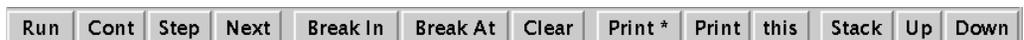
- For a summary of debug commands, type `help` at the command line prompt (`(gdb)`). Note that many of these commands can also be posted from the Source Editor.

In the Project Editor

You will be setting a breakpoint in the source file, `display.c`. To open the file:

1. In the Project Tree, make sure that `lib.shared` is highlighted.
2. Right-click anywhere in the Project Tree and choose **Context menu > Select From lib.shared Only**.
3. In the File List, double-click on `display.c` to open the file in the Source Editor.

The file is opened in the Source Editor and, because you are in debug mode, a row of buttons for the most commonly needed debug commands has been added below the menu bar.



Setting Breakpoints

Let's set a breakpoint at the start of function `clearcursor`. To do so:

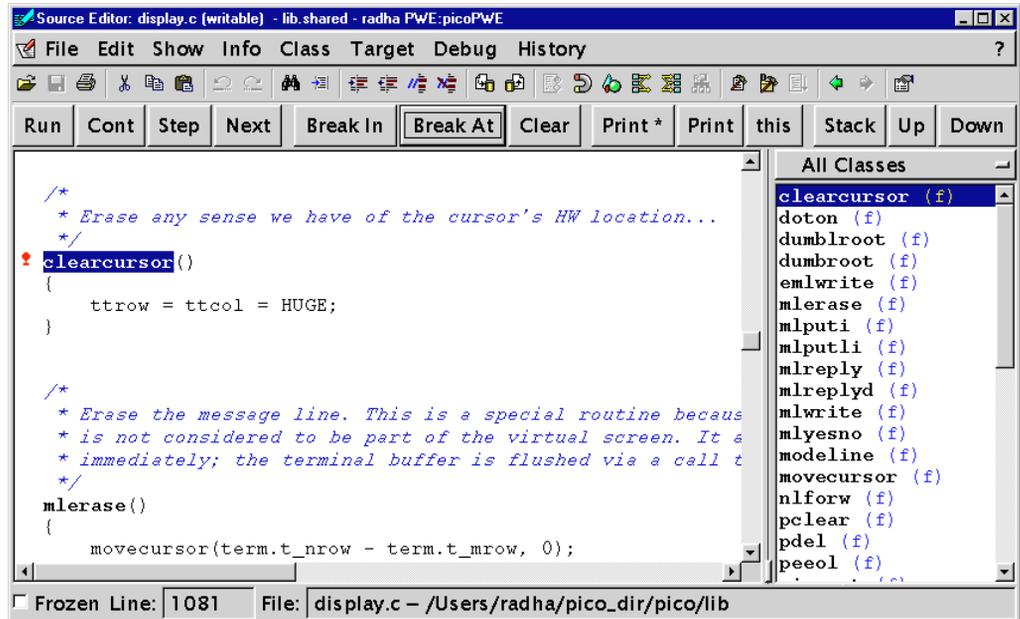
To set this breakpoint:

1. In the Source Editor's Symbol List, click on `clearcursor (f)`.

The Editor is positioned at the `clearcursor` function.

2. To set the breakpoint, press **Break At**.

A small stop sign at the beginning of the line indicates the breakpoint.



3. In the Debugger, select the **Breakpoints** tab. Notice that the breakpoint is listed in the Breakpoints list.

4. Press the **Show** button.

The Source Editor is positioned to the source code line in which the breakpoint is set.

Closing the Debugger

1. To close the Debugger, choose **Tools > Close Tool**.
2. Close all open SNIFF+ tools except for the Launch Pad.
3. In the Launch Pad, close the `pico.shared` project by highlighting it and choosing **Project > Close Project pico.shared**.

Conclusion

This concludes the Edit/Compile/Debug tutorial. In this part of the C Tutorial you were introduced to:

- Setting Make attributes for a C project
- Building a SNIFF+ project and all the subprojects that it depends on
- Using the Shell tool to jump to errors in source files
- Using the Gdb Debugger to debug executables

In the tutorial we can only give you a few hints and tips to help you on your way. To get a better understanding of how to edit, compile and debug projects in SNIFF+, we recommend experimenting. To really appreciate what SNIFF+ can do for you, you really need to work with it.

What's next

The only SNIFF+ tool that should now be open is the Launch Pad.

The next part of the C tutorial introduces you to

- Using SNIFF+ for multi-user team projects

Part IV

Team Setup



Key Concepts

Note

This tutorial assumes that you will use RCS (included in the SNIFF+ package) for configuration management and version controlling. Most other CMVC tools are also supported by SNIFF+. Please refer to the *Release Notes* for details.

This chapter introduces 2 key concepts:

- *shared projects*
- *working environments*

Although these concepts are not in themselves difficult, what follows in the hands-on tutorial chapters may tend to get a little confusing if you don't have a reasonable understanding of what these things are and how they work.

For detailed information beyond this very brief introduction, please refer to the *User's Guide*.

Shared projects

A shared project is, as the name suggests, suitable for team development. However it is equally recommended for single-user work situations.

Shared projects offer a great deal of flexibility. Because all references to files and subprojects are relative to a root directory, you can easily move a shared project to another location on a file system.

Each team member can access a shared project and make changes to its files and/or structure, regardless of what other team members are doing.

This means that the integrity of the project system as a whole needs to be maintained in some way, which is why shared projects are always used in conjunction with working environments and a configuration management and version control (CMVC) tool.

It is strongly recommended that one person be appointed to administer this "maintenance system". In SNIFF+ this person is called the *Working Environments Administrator*. This tutorial mainly covers the tasks performed by a Working Environments Administrator.

- From now on, shared projects are simply referred to as *projects*.

Working environments

SNIFF+ uses 4 different kinds of working environments:

- Repository Working Environment (RWE)

- Shared Source Working Environment (SSWE)
- Shared Object Working Environment (SOWE)
- Private Working Environment (PWE)

The RWE (Repository Working Environment)

Your team members access and modify a permanent shared data Repository using commands provided by your underlying configuration management and version-control (CMVC) tool.

SNiFF+ provides an interface to your CMVC tool. This interface needs to know the location of your Repository.

You provide this information by defining a *Repository Working Environment* (RWE), which specifies the root directory of your Repository.

In this tutorial, we will be using RCS, the CMVC tool provided with the SNiFF+ package.

The SSWE (Shared Source Working Environment)

SNiFF+ requires you to specify the root directory under which your team's shared source code is located. The files and directories under this root directory access your team's Repository. At regular intervals, all these files and directories are updated to reflect the most current state of your team's software system.

When creating software systems from scratch, your team's (Working Environments Administrator's) first job is to populate this root directory with source code. For existing software systems, your team will already have such a central location.

In either case, once you have such a root directory, you have to tell SNiFF+ where it is. You do this by defining a *Shared Source Working Environment* (SSWE).

All team members see, or *share*, all the source files in the SSWE. When browsing the source files, this view is read-only. When editing source files, team members work on *local* copies of the shared source files they want to modify—they never directly modify the shared source files in the SSWE. The view to all other source files remains read-only.

The SOWE (Shared Object Working Environment)

Just like with shared source code, SNiFF+ also requires you to specify a central location for your team's shared object files. In SNiFF+, you define one *Shared Object Working Environment* (SOWE), which specifies the root directory containing these files, for each target platform.

SOWEs serve as shared repositories for your team's most current and stable object code. During an update of an SOWE, source files in the SSWE are compiled and the resulting object code is stored in the SOWE.

An essential aspect of SOWEs is avoiding unnecessary builds in Private Working Environments (see below) that access them.

The PWE (Private Working Environment)

Developers must be able to work in isolation from other team members. They need their own workspaces in which they can edit, compile and debug projects without interfering with the work of other team members. Furthermore, they continually need to have access to their software system's most current source code and object code base.

SNiFF+ supports this by allowing each member of a team to work in an isolated workspace. In SNiFF+, you define a *Private Working Environment* (PWE) to specify the root directory of each team member's workspace.

You can go through the entire edit/compile/debug cycle in your PWE. In your PWE, you have a read-only view to the shared source files located in your team's SSWE. When you need to modify shared source files, you check out the necessary files from your team's Repository. When you're satisfied that the changes you've made are error-free, you check the modified files back into your team's Repository. The next time your team's SSWE is updated, these changes are incorporated, and the shared source files in the SSWE once again reflect the most current state of your software system.

Multi-User Project Setup

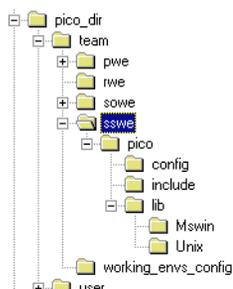
This chapter is about

- using the Project Setup Wizard for setting up a SNIFF+ multi-user / multi-platform project for development.

The Project Setup Wizard guides you through the process of setting up a multi-user / multi-platform project with version controlling.

Preparing the Environment

- In the Browsing tutorial, you copied the directory `<your_sniff_installation_dir>/example/c/pico_dir`, including subdirectories, to a place where you have write permissions. If you haven't done so, please do so now. You should have the following directory structure:



In the rest of this tutorial, we will use `<PICO_DIR>` to refer to the complete path to this directory.

Working environment information

`pwe` — This directory holds your own workspace, i.e., your **Private Working Environment**.

`rwe` — This directory holds your team's shared data repository, i.e., your **Repository Working Environment**.

`sowe` — This directory holds your team's shared object code, i.e., your **Shared Object Working Environment**.

`sswe` — This directory holds the source code your team shares, i.e., your **Shared Source Working Environment**.

`working_envs_config` — This directory will hold the working environment files generated and maintained by SNIFF+.

Setting your Preferences

To set the `working_envs_config` directory as your preferred maintenance directory:

- In the Launch Pad (or any other open SNIFF+ tool), choose **Tools > Preferences...** to open the Preferences dialog.

In the Preferences dialog

1. Under the **Tools** node, select **Working Environments**.
2. In the Working Environments view, press **Dir...** next to the **Working Environments Config. Directory** field.
3. Navigate to the `<PICO_DIR>/team/working_envs_config` directory.
4. Double-click on the directory name and press **Select**.
5. Press **Ok** to apply the changes and close the Preferences dialog.

Multi-user Project Setup Wizard

- To start the Project Setup Wizard, in the Launch Pad, choose **Project > New Project > with Wizard...**

In the Project Setup Wizard

The Wizard starts by asking you to select how you intend to use the new SNIFF+ Project.

- Accept the default selection, **Standard Setup**, and press **Next**.

The “Select developmental task” page appears.

In the remaining steps, we will refer to the names of Wizard pages. You can find a page’s name in the title bar of the Wizard.

In the “Select Developmental task” page

- Select **Create a new SNIFF+ Project from scratch** and press **Next**.

In the “Your development organization” page

This tutorial is for multi-user / multi platform development with configuration management and version control (CMVC), so:

1. Select **Yes** for both Yes/No questions.
2. Choose **RCS** as the version control tool.
3. Press **Next**.

Note

This tutorial assumes that you will use RCS for version controlling. Most other CMVC tools are also supported by SNIFF+. Please refer to the *Release Notes* for details.

In the “Select file types” page

- Select **C/C++** and press **Next**.

SNiff+ will automatically include all necessary file types needed for working with C/C++ source code in the new project. Note that, after project setup, you can add new standard file types (like the ones in the “Additional File Types Column”), or create and add your own.

In the “Specify Repository” page

You are asked to specify your *Repository Working Environment* (RWE). SNiff+ uses the RWE for version control administration. To specify the `rwe` directory:

1. Press **Browse** and, in the Directory dialog, navigate to:

```
<PICO_DIR>/team/rwe
```

 Double-click on the `rwe` directory, and then press **Select**.
2. In the **RWE name** field, type a name for the RWE, e.g., `Pico Repository`.
3. Press **Next**.

In the “Specify team source code location” page

You are asked for your *Shared Source Working Environment* (SSWE).

1. Press **Browse** and, in the Directory dialog, navigate to:

```
<PICO_DIR>/team/sswe
```

 Double-click on the `sswe` directory, and then press **Select**.
2. In the **SSWE name** field, type a name for the SSWE, e.g., `Pico SSWE`.
3. Press **Next**.

In the “Specify team object code location” page

You are asked to specify your *Shared Object Working Environment* (SOWE) root directory.

1. Press **Browse** and, in the Directory dialog, navigate to:

```
<PICO_DIR>/team/sowe
```

 Double-click on the `sowe` directory, and then press **Select**.
2. In the **SOWE name** field, type a name for the SOWE, e.g., `Pico SOWE`.
3. Press **Next**.

In the “Specify Private Working Environment” page

You are asked to specify your *Private Working Environment* (PWE) root directory.

1. Press **Browse** and, in the Directory dialog, navigate to:

```
<PICO_DIR>/team/pwe
```

 Double-click on the `pwe` directory, and then press **Select**.

2. In the **PWE name** field, type a name for the PWE, e.g., `Pico PWE`.
3. Notice that your user name is entered next to the enabled **Owner** button. SNIFF+ needs your user name to correctly handle permissions.
Being the owner of the PWE means that you are the only one who is allowed to modify the working environment's attributes.
4. Press **Next**.

In the “Additional team members?” page

You are asked whether any additional PWEs are needed. Since you are working alone through this Tutorial, you don't need to specify additional PWEs.

- Accept the default value and press **Next**.

In the “Additional target platforms?” page

You are asked whether any additional SOWEs are needed. Since the code in this Tutorial will only be compiled for one platform, you don't need to specify additional SOWEs.

- Accept the default value and press **Next**.

In the “Create new SNIFF+ Project” page

You are asked to specify the root directory of the new project. SNIFF+ automatically enters the root of your SSWE in the **Project root directory** field, since your team's shared source code is located in it. Our project root directory is `pico`, so:

1. Press **Browse** and, in the Directory dialog, navigate to:

```
<PICO_DIR>/team/sswe/pico
```

Double-click on the `pico` directory, and then press **Select**.

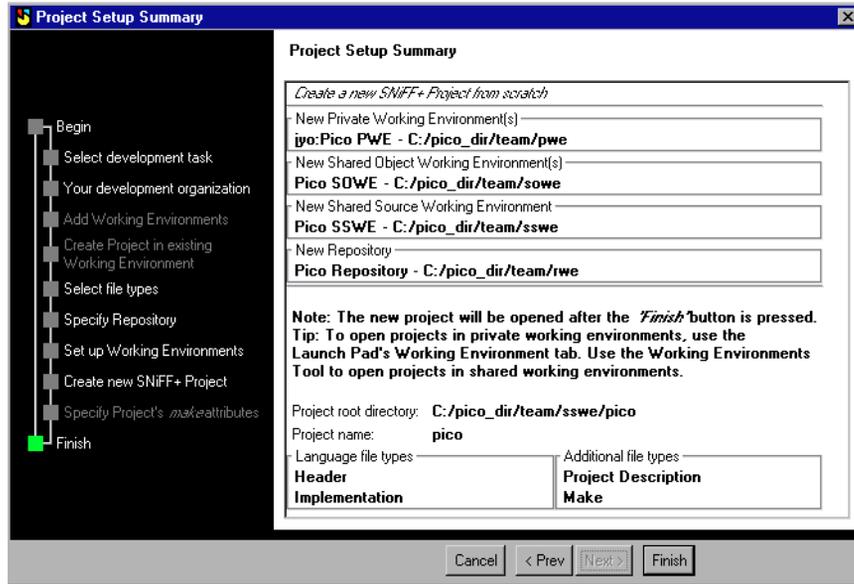
Notice that the new project's name has changed to `pico`, which is the name we will use throughout the tutorial. Also by default, **Create Subprojects** is enabled.

2. Select the **Use SNIFF+'s Makefiles** checkbox.
3. Press **Next**.

In the “Project Setup Summary” page

This page summarizes your specifications for the new SNIFF+ project and required working environments.

1. Make sure that your Project Setup Summary page is similar to the following. If it isn't, please go back to the beginning of the Wizard and start again:



2. Press **Finish**.

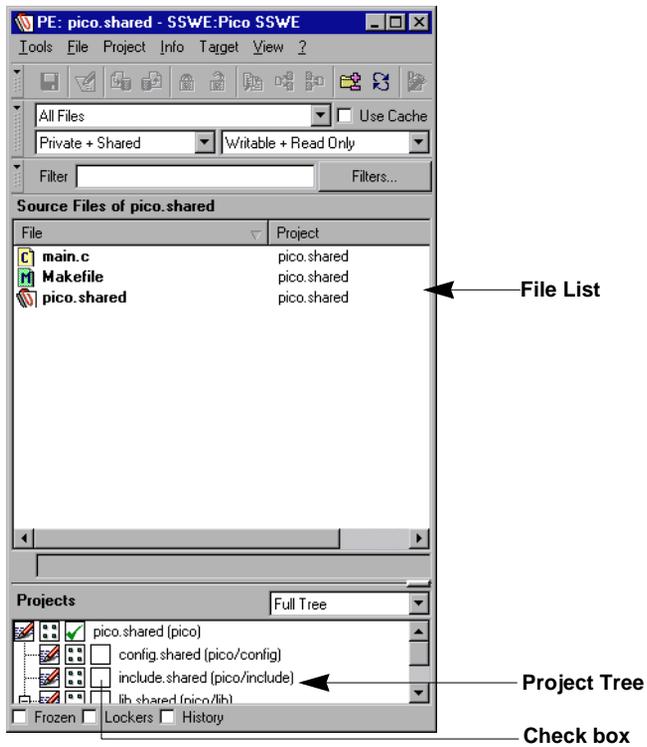
SNIFF+ will now create the new `pico` project and all its subprojects.

3. In the dialog that appears asking if you want to generate cross reference information, press **No**.

When SNIFF+ is finished, it opens the new project in the SSWE (where you set up the project) and displays its structure and contents in the Project Editor.

Viewing the results

The Project Editor on your screen should look this:



Setting Up the Build System in the SSWE

This chapter is about

- setting up the build system in the Shared Source Working Environment (SSWE).

Although you don't build your targets in the SSWE, you set the Make attributes here. Then, when you later open the project in the SOWE and PWEs, you can build targets without first having to modify the project's Make attributes.

Setting up the build system for a multi-user team project is identical to setting up the build system for a single-user project. Therefore, if you have gone through the "Browsing" tutorial, the steps in this chapter will be very familiar to you.

In both the single-user and multi-user cases, you set up the build system in the working environment that contains your project's source code. In a multi-user situation, team source code is contained in the SSWE, which is why we are working in the SSWE in this chapter.

Setting up C Make Support attributes

1. In the Project Tree of the Project Editor, checkmark `pico.shared`, `lib.shared`, `MSWIN.shared` on Windows and `UNIX.shared` on Unix.
2. Choose **Project > Attributes of Checkmarked Projects...**

The Attributes of Checkmarked Projects dialog appears. You can use this dialog to set/modify the attributes of multiple projects at the same time.

In the Attributes of Checkmarked Projects dialog

Setting up Make Support for `pico.shared`

1. Under the **Build Options** node, select **Project Targets**.
2. Make sure that `pico` is highlighted in the Project List.
3. In the **Executable** field of the Ansi C/C++ tab, enter `pico` on Unix and `pico.exe` on Windows. This will be the name of the project's executable.
4. Under the **Build Options** node, select **Directives**.
5. Select the checkbox to the right of the Generate button and press **Set for All**.
This attribute is now set for all projects.
6. Under the **Build Options** node, select **Build Structure**.
7. In **Recursive Make Dirs** field, enter `lib`.

In order to build the project's executable, SNIFF+ must first build the target of the `lib.shared` subproject (a library). SNIFF+ uses the project information in **Recursive Make Dirs** field to determine the order in which to execute Make.

Setting up Make Support for lib.shared

1. Highlight **lib** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Library** field of the of the Ansi C/C++ tab, enter `libpico.a`. This will be the name of the library built in this project.
4. Under the **Build Options** node, select **Build Structure**.
5. From the **Passed to Superproject** drop-down, choose **Library**.
The project's library is exported to `pico.shared` and is used to build the Pico executable.
6. In **Recursive Make Dirs** field, enter **UNIX/MSWIN** on Unix/Windows.
The library is built using objects built in the platform-specific subdirectory of `lib.shared`.

Setting up Make Support for MSWIN.shared (Windows only)

1. Highlight **MSWIN** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Relinkable Object** field of the of the Ansi C/C++ tab, enter `osdep.o`. This will be the name of the relinkable object built in this project.
4. Press **Ok** to apply the changes to the project attributes.
5. A dialog appears asking you if you want to update Makefiles. You will be updating Makefiles later on, so press **No**.

Setting up Make Support for UNIX.shared (Unix only)

1. Highlight **UNIX** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Relinkable Object** field of the of the Ansi C/C++ tab, enter `osdep.o`. This will be the name of the relinkable object built in this project.
4. Press **Ok** to apply the changes to the project attributes.
5. A dialog appears asking you if you want to update Makefiles. You will be updating Makefiles later on, so press **No**.

Saving changes

- In the Launch Pad, save the changes made to `pico.shared` and its subprojects:
 - Select `pico.shared` in the Project List.
 - Choose **Project > Save Project pico.shared**.
 - In the Alert dialog that appears, press the **Save all** button.

What's next

You may think that the next step is to build the project's executable in the SSWE. It isn't. In SNIFF+'s working environments concept, SSWEs contain only shared source code, and SOWEs contain the objects and targets based on this code.

During project setup, you created a SNIFF+ project in the directory that contains your team's shared source code, i.e., in the SSWE. Once the project has been created, the only time you open it in the SSWE is to update it. For any real development work, open the project in a PWE.

So, the next step is to check in the project (its Project Description File) and its source files into the Repository. When the process is over, all the files in the SSWE will be read-only.

Then, you can open the project in the SOWE and build the executable in it. This task is covered in [First Build in the SOWE — page 89](#).

Checking In the project from the SSWE

This chapter is about

- checking in project files from the SSWE

Checking in project files for the first time is the first step in version-controlling your SNIFF+ projects. We recommend that you version control at least the following types of files:

- Project Description Files (PDFs), i.e., * .shared files in our case.
- source files
- Makefiles (only if you don't use SNIFF+'s Make Support)

Once files have been checked in, you can see their history and version tree information.

In a real-world situation, it may not matter to you whether your team's shared source code is initially compilable. However, when creating new SNIFF+ team projects from scratch, we recommend that you verify that your source files are compilable before checking them in for the first time. Do not, however, perform builds in the SSWE. The SSWE should only contain source files.

Checking in the project

To check the project in, complete the following steps:

In the Project Editor

1. In the Project Tree, checkmark all projects by right-clicking anywhere in the Project Tree, and then choosing **Context menu > Select From All Projects**.

You now see all the files in all the projects.

2. Press the **Filters...** button.

The Filters dialog appears. You will now filter out SNIFF+'s Makefiles from the Project Editor's File List.

3. In the **FileTypes** tab, clear the **Make** check box and press **Ok**.

SNIFF+'s Makefiles are generated and maintained by SNIFF+, so there's no reason to version control them.

4. Choose **File > Select All**.

5. Choose **File > Check In...**

SNiFF+ informs you that it cannot find the directories of the shared project in the RWE root directory (they haven't been created yet). You will now have SNiFF+ initialize your RWE by copying the SSWE project directory structure into the RWE. This dialog will reappear for each new Repository directory, unless you select the **Repeat** check box.



6. Select the **Repeat** check box and press **Yes** to create the necessary Repository directories for the project.

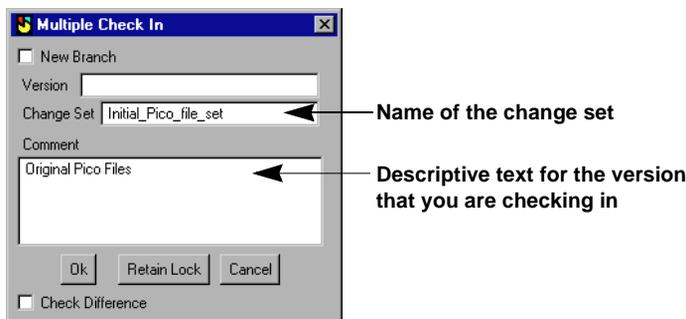
When SNiFF+ has finished initializing your RWE, the Multiple Check In dialog appears.

In the Multiple Check In dialog

You can use this dialog to check in versions of single or multiple files. When you have made changes to multiple files, you can check in all the files at the same time and associate them with a *change set*. By doing so, you can perform a variety of version-control operations on all the files in a change set at the same time.

At this point, although we haven't made any changes, we will make use of the **Change Set** field to indicate that we are checking in the initial versions of all the files in the project.

1. Leave the **Version** field blank. SNiFF+ will automatically assign a version number (1.1) and later increment it automatically.
2. In the **Change Set** field, enter a name for the change set, e.g., `Initial_Pico_file_set`.
3. In the **Comment** field, enter a descriptive text, e.g., `Original Pico Files`.



4. Press **Ok**.

In the Project Editor

When the check-in process is over, take a look at your Project Editor. You should notice the following changes:

- The files in the File List are no longer in bold typeface. This means they are now read-only.
- The icons in the Project Tree have also changed to indicate that the projects, too, are read-only.

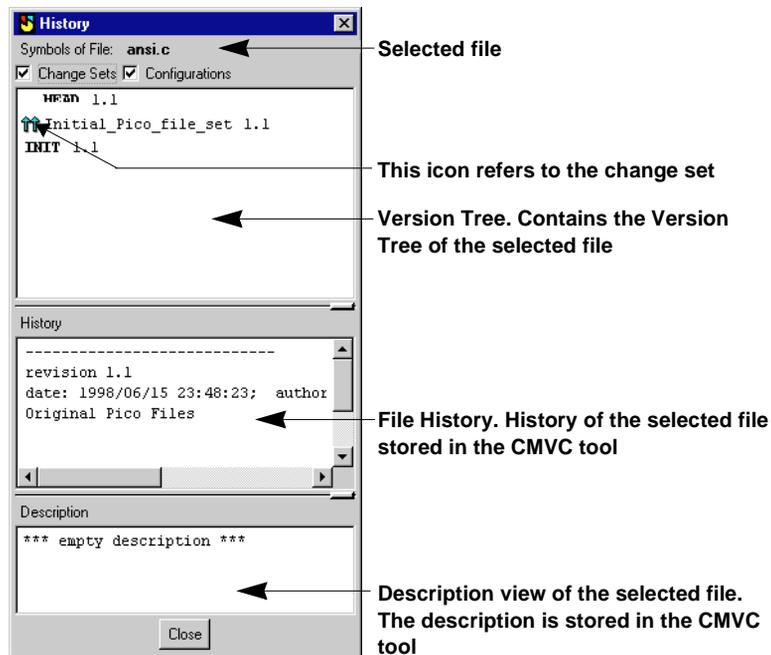
Looking at the history of a file

You can check to see whether the files were checked in properly by looking at their history. Let's look at the history of a file.

In the Project Editor

1. In the File List, highlight the file `ansi.c`.
2. Select the **History** check box.

A new History window appears:



In the Version Tree view, the version tree of the selected file is displayed. Since only one version of the project files has been checked in so far, the Version Tree only displays this version (1.1).

INIT is used by SNIFF+ to refer to the initial version of a file in the Repository. The version number of the INIT version of a file is always 1.1. The latest version on the main trunk or branch of a file's version tree is called HEAD. In this example, the HEAD and INIT versions of the file are naturally the same.

3. In the Project Editor, clear the **History** check box.

What's next

The next step is to open the project in your SOWE.

Although you can open projects in more than one working environment at a time, this tends to get confusing. We therefore suggest that you first close the project in the SSWE.

In the Launch Pad

To close the project in the SSWE:

1. Highlight `pico.shared - SSWE:Pico SSWE`.

You can see the name of the project and the working environment in which you opened it by increasing the size of the Launch Pad.

2. Choose **Project > Close Project pico.shared**.

First Build in the SOWE

This chapter is about

- **Opening the shared project in the SOWE** — When you first open the shared project in the SOWE, SNIFF+ will automatically initialize the environment by copying the directory structure found in the SSWE.
- **Building and running the Pico executable** — A successful build verifies that you have set the project's Make attributes correctly. After the initial build you will have the targets for a stable running version of the project in your SOWE.

During the edit/compile/debug cycle, each developer should only build targets in his/her own Private Working Environment (PWE). Builds in the SOWE should only take place during regular updates of your team's working environments (described in the "Team Maintenance" tutorial). The initial build in the SOWE is in fact your first update of this working environment.

Opening the shared project in the SOWE

First, you need to tell SNIFF+ that you intend to work in the SOWE. You do this in the Working Environments tool.

- In the Launch Pad, choose **Tools > Working Environments**.

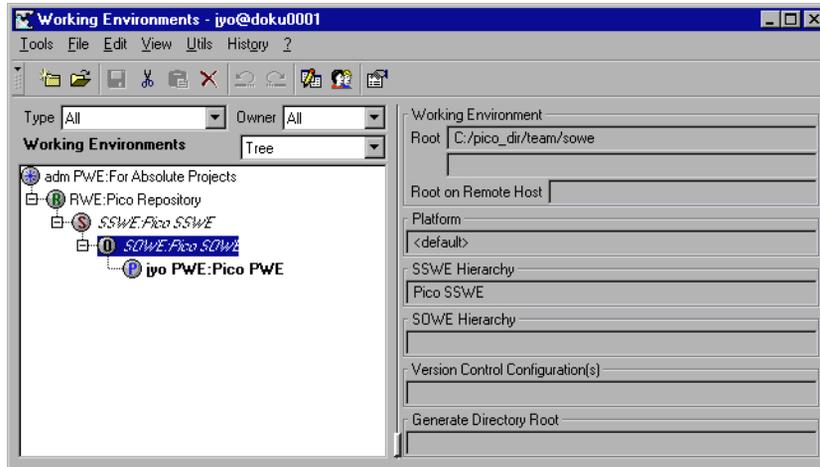
In the Working Environments tool

Note

If the main view of the Working Environments tool is initially empty, choose **File > Reload**.

1. To open a project in the SOWE, double-click on the SOWE entry in the Working Environments Tree. If you used the same names as we did, the full designation of the SOWE is:

SOWE:Pico SOWE

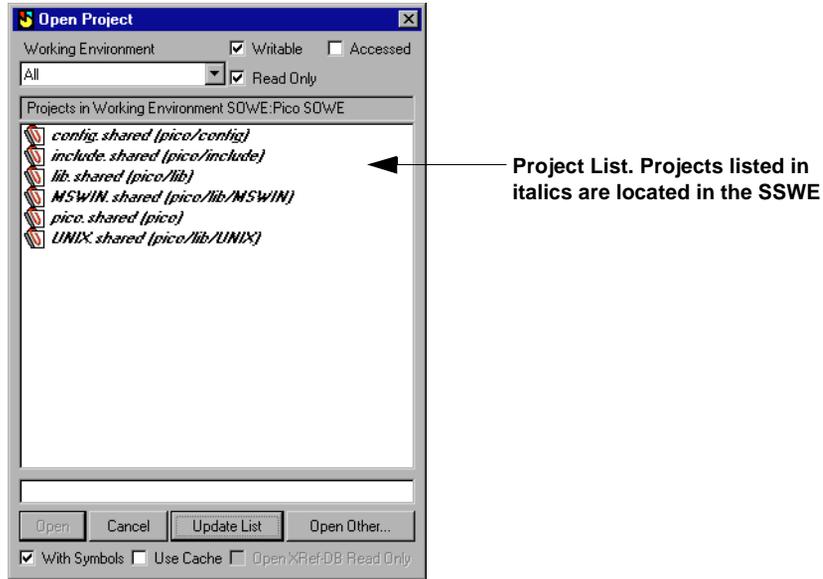


2. In the Open Project dialog that appears, press the **Update List** button to display all the projects that can be opened in the SOWE.

A dialog appears asking you whether SNIFF+ should also look in any accessed working environments for projects that can be opened in the SOWE. Here, the SOWE accesses the SSWE, so pressing **Yes** will also display the projects in the SSWE.

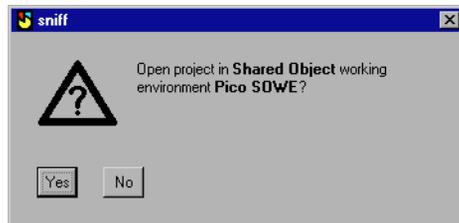
3. Press **Yes**.

The Open Project dialog on your screen should now look like this:



4. To open the root project and all its subprojects, double-click on `pico.shared`.

A dialog appears asking you to confirm that you want to open the project in a shared working environment.



5. Press **Yes**.

SNIFF+ informs you that it cannot find the directories of the shared project in the SOWE root directory (they haven't been created yet). You will now have SNIFF+ initialize your SOWE by copying the SSWE project directory structure into the SOWE.

6. Select the **Repeat** check box and then press **Create Directory**.

Selecting **Repeat** saves you from having to press **Create Directory** for each new project directory.

When SNIFF+ has finished initializing your SOWE, the project is automatically opened in it and displayed in the Project Editor.

7. Close the Working Environments tool.

Building the executable

Note

SNiFF+ doesn't have its own compiler therefore you must have a compiler installed on your computer to compile SNiFF+ projects. By default, the gnu compiler is specified on Unix and Microsoft Developer is specified on Windows. If you are using another compiler, it must be specified in your Platform Makefile. For more information, see *User's Guide — Build and Make Support*.

In the Project Editor

Before building, make sure that the projects' Make Support information is up-to-date. Makefiles should be updated whenever structural changes are made to the projects, or when projects are first opened in a new working environment.

1. Make sure that all the projects in the Project Tree are checkmarked. If they are not, right-click anywhere in the Project Tree and choose **Context menu > Select from All Projects**. This command allows you to checkmark all projects in one step.

2. Choose **Target > Update Makefiles** to generate the Make Support Files for all the projects.

A dialog appears asking you whether the dependencies information should also be updated.

3. Press **Yes**.

SNiFF+ generates the Make Support Files and stores them in the `.sniffdir` subdirectory of each project directory.

4. Highlight `pico.shared` by clicking on its name.

SNiFF+ needs to know where to start Make execution. You tell SNiFF+ this by selecting the appropriate project. In the example project, Make execution starts in `pico.shared`.

5. Choose **Target > Make > all** to recursively build the executable.

A Shell opens, in which the `make all` command is recursively executed. Upon completion, you should have an executable named `pico` on Unix and `pico.exe` on Windows in:

```
<PICO_DIR>/team/sowe/pico
```

6. Run the executable to assure yourself that it executes properly.

What's next

- Close the project in the SOWE.

In the Launch Pad

1. Select `pico.shared - SOWE:Pico SOWE`.

2. Choose **Project > Close Project** `pico.shared`.

The next tutorial introduces you to

- Developing in a team

Part V

Developing in a Team



Working in the PWE

In this chapter, you will go through the basic tasks when working in a PWE in a team context:

- opening the shared project in the PWE for the first time and letting SNIFF+ initialize it for you
- checking out a shared source file and making a minor modification to it
- checking the modified file back in
- creating a file, adding it to, and removing it from a project

This tutorial does not cover day-to-day development work or the various browsing tools. For an introduction to the tools used in daily development work, please refer to [Edit/Compile/Debug — page 49](#). Browsing tools are introduced in [Browsing — page 9](#).

Opening the shared project in the PWE

First, you need to tell SNIFF+ that you intend to work in the PWE. You do this in the Working Environments tool.

- In the Launch Pad, choose **Tools > Working Environments**.

In the Working Environments Tool

1. To open a project in the PWE, double-click on the PWE entry in the Working Environments Tree. If you used the same names as we did, the full designation of the PWE is:

Username PWE:Pico PWE

2. In the Open Project dialog that appears, press the **Update List** button to display all the projects that can be opened in the PWE.
3. In the dialog that appears, press **Yes** to confirm that shared workspace information should also be used.
4. To open the root project and all its subprojects, double-click on `pico.shared`.

SNIFF+ informs you that it cannot find the directories of the shared project in the PWE root directory (they haven't been created yet). You will now have SNIFF+ copy the SSWE project directory structure into the PWE.

5. Select the **Repeat** check box and then press **Create Directory**.

Selecting **Repeat** saves you from having to press **Create Directory** for each new project directory.

When SNIFF+ has finished initializing your PWE, the project is automatically opened in it and displayed in the Project Editor.

6. Close the Working Environments tool.

In the Project Editor

Makefiles should be updated whenever structural changes are made to the projects, or when projects are first opened in a new working environment.

1. Make sure that all the projects in the Project Tree are checkmarked. If they are not, right-click anywhere in the Project Tree and choose **Context menu > Select from All Projects**.
2. Choose **Target > Update Makefiles....**

A dialog appears asking you whether dependencies information should also be updated.

3. Press **Yes**.

Generally, your Working Environments Administrator is responsible for setting up SNIFF+'s Make Support. Therefore, you don't need to see project Makefiles in your day-to-day work. Also, SNIFF+'s Makefiles are generated and maintained by SNIFF+, so there's no reason to version control them.

1. Press the **Filters...** button.

The Filters dialog appears. You will now filter out SNIFF+'s Makefiles from the Project Editor's File List.

2. In the **FileTypes** tab, clear the **Make** check box and press **Ok**.

Check out and check in

Remember that you checked in the project to the Repository from the SSWE, so the view to the project files is read only. To modify a file, you first need to check it out.

To review how to check in a project from the SSWE, please refer to [Checking In the project from the SSWE — page 85](#).

In the Project Editor

The file which you will modify is `ansi.c`, which belongs to the `lib.shared` project. To check out `ansi.c`:

1. In the Project Tree, make sure that the `lib.shared` project is checkmarked, so that you can see its files.
2. In the File List, highlight `ansi.c` by clicking on it once.
3. Choose **File > Check Out....**



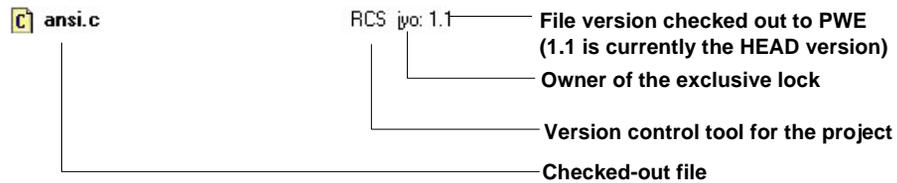
HEAD is the latest version of the file in the Repository

4. In the Check Out dialog, press **Exclusive Lock**.

In the File List, notice that `ansi.c` is now displayed in **bold** typeface, which means that it is writable.

5. Select the **Lockers** check box at the bottom of the Project Editor. This check box allows you to see which users have locked which files.

The File List entry for `ansi.c` should now look like this:



6. To load the now writable `ansi.c` file into the Source Editor, double-click on it in the File List.

In the Source Editor

All we want to do here is to make a modification, so that a newer version of the file exists.

1. Enter a comment in the first line of the file.

Notice that the Source Editor now indicates that the file has been modified.

- On **Unix**, the icon in the upper-left corner of the Source Editor indicates that the file has been modified.
- On **Windows NT/95**, the write permissions of the loaded file and its status are indicated in the title bar of the Source Editor.

2. Save `ansi.c` by choosing **File > Save**.
3. Close the Source Editor.

Checking in the file

Once you are satisfied with the changes you have made to a file, you check it back in. The rest of the team then has access to the modified file (after the shared working environments have been updated - see next chapter).

Note that “being satisfied with changes”, above, means that, at the very least, your code is compilable. You should NOT check in untested, possibly uncompileable, code!

Since you only added a comment to the checked out file, it is safe to check it back in.

In the Project Editor

You can check in files either from the Project Editor or the Source Editor. Here, you will use the Project Editor (the menu command is the same in both tools).

1. In the File List, make sure `ansi.c` is highlighted. This is the file you checked out, as you can see by the **bold** typeface.
2. Choose **File > Check In...**

3. In the Check In dialog, enter a comment in the **Comment** field.

You can leave the **Version** field blank, because SNIFF+ will automatically increment the version number to 1.2. Also leave the **Change Set** field blank; this is usually only used for multiple files.

4. Press **OK**.

In the File List, notice that `ansi.c` is now displayed in regular typeface, which means that it is now read-only and located in your PWE.

5. To look at the history of `ansi.c`, highlight the file in the File List and select the **History** check box at the bottom of the Project Editor.

Notice that the HEAD version of the file is now version 1.2.

Take a look at the history of some of the other project files. Since you have not modified those files since checking them in, their HEAD version is still 1.1.

6. Clear both the **History** and **Lockers** check boxes.

Adding a new file to a project

In the course of your day-to-day development work you will often create new source files. These files must be included in the project they logically belong to. SNIFF+ will do this for you by modifying the Project Description File (PDF) accordingly. But first you have to check out the PDF to make it writable.

In the Project Editor

You will add a new file to the `pico.shared` project. To first get an uncluttered view of the project, and then check out its PDF:

1. In the Project tree, highlight the `pico.shared` project, right-click and choose **Context menu > Select From pico.shared Only**.
2. In the File List, highlight the `pico.shared` file (the PDF).
3. Choose **File > Check Out...**
4. In the Check Out dialog, press **Exclusive Lock**.
5. In the dialog that appears to warn you about project structure changes (you are checking out a PDF), press **Yes**.

In the File List, notice that `pico.shared` is now displayed in **bold** typeface, which means that it is writable.

Creating and adding the new file

Once you have checked out the PDF:

1. In the Project Tree, make sure that the `pico.shared` project is highlighted.
You highlight the project so that SNIFF+ knows which project you intend to modify.
2. Choose **Project > Add New File to pico.shared...**

3. In the New File dialog that appears, enter the name of the file you want to create, e.g. `test.c`.
4. Press **Ok**.

The new file is added to the File List, and the icon next to `pico.shared` in the Project Tree changes to warn you that the project structure has changed.

Note

SNiFF+ will only allow you to add file types that you have specified as being part of the project. During project setup, you specified the **C/C++** file types. To find out how to add new file types to a project, please refer to the *User's Guide*.

5. Choose **Project > Save pico.shared...**

The icon in the Project Tree has changed again; it indicates that the project is writable (the PDF is still checked out). The project information has now been saved, and will be used in your PWE only. As soon as you want to make the file you added available to the rest of the team, check in the PDF again (don't check it in yet).

Removing files from a project

You may also have to remove source files from a SNiFF+ project in the course of your day-to-day development work. When you remove a file from a project, the file is not physically deleted; SNiFF+ simply edits the PDF, which means it must be writable (checked out). To make the PDF available to the rest of the team, you have to check in the PDF again. To show how the process of removing files works, let's now remove `test.c`.

In the Project Editor

The `pico.shared` PDF should still be checked out (**bold** typeface).

1. In the Project Tree, make sure the `pico.shared` project is highlighted so that SNiFF+ knows which project you intend to modify.
2. Choose **Project > Add/Remove Files to/from pico.shared...**

In the Add/Remove Files dialog

- To remove `test.c` from the project, double-click it in the Files In Project list and press **Ok**.

The file is removed from the project, but is still physically stored in the directory. Files in the project directory can later be easily added to the project again using this dialog.

In the Project Editor

The `test.c` file no longer appears in the File List, and the icon in the Project Tree warns you that the project has been modified.

1. To save the changes you made, choose **Project > Save pico.shared**.

The icon in the Project Tree and the **bold** typeface in the File List indicate that the PDF is writable - you haven't checked it in yet.

2. In the File List, make sure the `pico.shared` PDF is highlighted, and choose **File > Check In...**

3. In the Check In dialog that appears, press **Ok**.

The latest version of the PDF is now in the Repository. After your next working environments update (described in the following chapter), any changes to the project structure will be visible to all team members.

- In the Launch Pad, close `pico.shared - Username PWE:Pico PWE`.

Part VI

Team Maintenance



Updating Working Environments

When a developer checks out a file, the checked-out version is locked in the Repository, and a local copy is made in the developer's PWE. When a developer is satisfied with changes he/she has made to a checked-out file (compilable!), he/she checks it back in. This means that the new (checked-in) version replaces the older (checked-out) version in the Repository. However, the SSWE still has the older version of the file, and the objects in the SOWE are also based on this version.

Clearly, the working environments are no longer consistent with each other, and they need to be updated so that all PWEs (i.e., their owners) can access the most current state of the project.

Updates should be done on a regular (daily) basis, especially if you have a large development team. The shared working environments (SSWE and SOWE) should only be updated by the Working Environments Administrator. Although it is relatively natural for individual developers to update their PWEs when they start work, this can also be done by the Working Environments Administrator.

You update your working environments in the following order:

- First, update the SSWE - the latest information is taken from the Repository.
- Then, update the SOWE (which accesses the SSWE) and build the targets with the latest file versions. The PWEs access the SOWE, so you can use the up-to-date object code in the SOWE for builds in PWEs.
- Finally, update your PWE so that you have a view to the latest configuration.

Here, only the most basic update requirements are described. For information on more advanced options and unattended updates, please refer to the *User's Guide*.

Updating the SSWE

`pico.shared` is the root project of all the other projects. When you update a root project in a particular working environment, SNIFF+ will automatically update all its subprojects.

First, you need to tell SNIFF+ that you intend to work in the SSWE.

- In the Launch Pad, choose **Tools > Working Environments**.

In the Working Environments tool

1. To open a project in the SSWE, double-click on the SSWE entry in the Working Environments Tree. If you used the same names as we did, the full designation of the SSWE is:
`SSWE:Pico SSWE`
2. In the Open Project dialog that appears, press the **Update List** button to display all the projects that can be opened in the SSWE.
3. To open the root project and all its subprojects, double-click on `pico.shared`.

4. In the dialog that appears, press **Yes**.
The project is opened in the SSWE and displayed in the Project Editor.
5. Close the Working Environments tool.

In the Project Editor

1. Checkmark all projects by right-clicking anywhere in the Project Tree, and then choosing **Context menu > Select From All Projects**.
2. Choose **Project > Synchronize Checkmarked Projects....**
The Files Compared To dialog appears. All files in the SSWE will be updated to the version that appears in the dialog's **Version** field (HEAD by default).
3. Press **Ok**.
A progress bar appears, and SNIFF+ updates all the files in the SSWE. A dialog then appears asking you to reload the project structure.
4. Press **Yes**.

In the Launch Pad

- Close `pico.shared` - SSWE:Pico SSWE.

Updating the SOWE

After you have updated the SOWE files, you should compile them. Subsequent builds in PWEs are then quicker, because the compiler uses the up-to-date object code in the SOWE. First, you need to tell SNIFF+ that you intend to work in the SOWE.

- Use the Working Environments tool to open `pico.shared` in your SOWE (how to do so was described under [First Build in the SOWE — page 89](#)).

In the Project Editor

1. Checkmark all projects by right-clicking anywhere in the Project Tree, and then choosing **Context menu > Select From All Projects**.
2. Choose **Project > Synchronize Checkmarked Projects....**
The Files Compared To dialog appears. All the files in the SOWE will be updated to the version that appears in the dialog's **Version** field (HEAD by default).
3. Press **Ok**.
SNIFF+ now updates all the files in the SOWE.
4. Choose **Target > Update Makefiles...** and press **Yes** in the dialog that appears.
Make Support Files are regenerated for all projects in the working environment.
5. In the Project Tree, highlight `pico.shared`.

6. Choose **Target > Make > all** to build the project's targets.

A Shell tool opens. The project's Make command is recursively executed in each of the projects in the Project Editor's Project Tree. Upon completion, you should have an executable named `pico` on Unix and `pico.exe` on Windows in:

```
<PICO_DIR>/team/sowe/pico
```

In the Launch Pad

- Close `pico.shared` - `SOWE:Pico SOWE`.

Updating the PWE

First, you need to tell SNIFF+ that you intend to work in the PWE.

- Use the Working Environments tool to open `pico.shared` in your PWE (how to do so was described under [Working in the PWE — page 97](#)).

In the Project Editor

1. Checkmark all projects by right-clicking anywhere in the Project Tree, and then choosing **Context menu > Select From All Projects**.
2. Choose **Project > Synchronize Checkmarked Projects...**
3. In the Files Compared To dialog that appears, press **Ok**.
SNIFF+ now updates all the files in the PWE.
4. Choose **Target > Update Makefiles...** and press **Yes** in the dialog that appears.
Make Support Files are regenerated for all projects in the working environment.
5. In the Project Tree, select `pico.shared`.
6. Choose **Target > Make > symbolic_links** to build the `symbolic_links` help target.
A Shell tool opens. Symbolic links are made in the PWE to all the objects and targets in the SOWE. On Windows NT/95, local copies are made instead of symbolic links.
This completes the update of the PWE. When you next build targets in your PWE, the results will reflect the latest status of the team project.

In the Launch Pad

- Close `pico.shared` - `username PWE:Pico PWE`.

Freezing the Project in the SSWE

Goals of this chapter

All your working environments are now up-to-date, your source files are compilable, and the project's executable functions properly. In this chapter, you will learn how to create a “virtual snapshot” of the project (or, to be exact, of its source files). 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”.

You can freeze configurations in the Configuration Manager. You can also use this tool to view the lists of configurations of your projects and to compare configurations. To learn more about the Configuration Manager, please refer to the *User's Guide* and the *Reference Guide*.

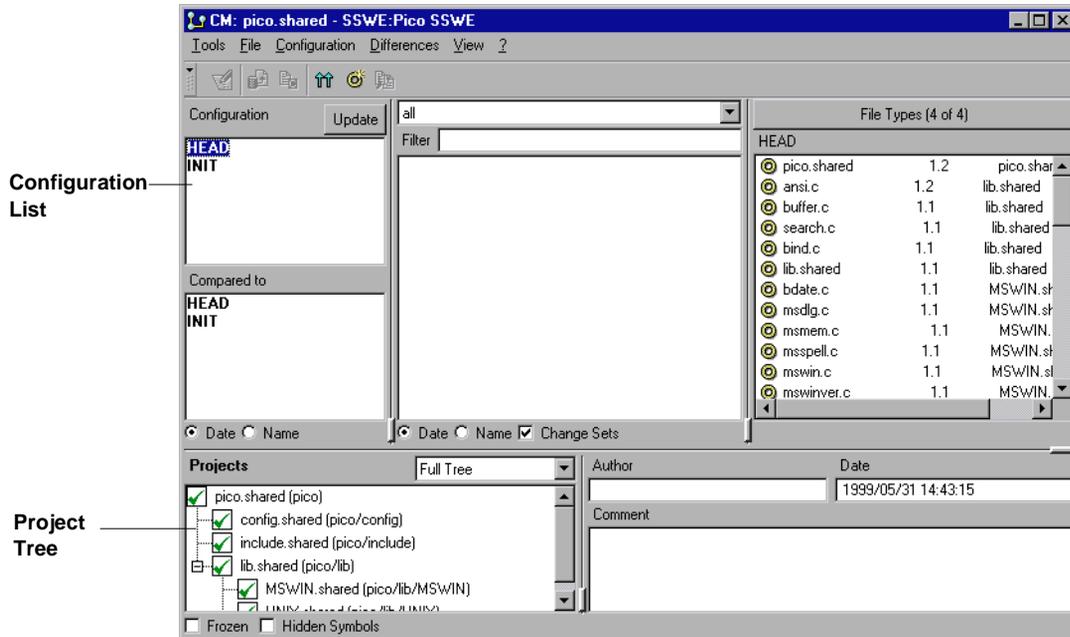
Freezing the project

To freeze the project:

1. Open the `pico.shared` project in the SSWE.
2. In any open SNIFF+ tool, choose **Tools > Configuration Manager**.

In the Configuration Manager

1. Select the **HEAD** configuration in the Configuration List (see the following screen shot).
The project's configuration information is loaded into the Configuration Manager.
Your Configuration Manager should look similar to the following:



2. Choose **Configuration > Freeze Head...**
The Freeze Head dialog appears.
3. Enter a name for the new configuration in the **Configuration** field of the dialog (e.g. Pico_configuration) and press **Ok**.
The Configuration List is now updated to include the newly created configuration.
4. In the Project Editor, take a look at the history of any of the project files. A circle next to one of the file's versions in the Version Tree indicates that the version is part of a configuration. The configuration name comes after the circle, followed by the version number.
5. Close all open tools except for the Launch Pad.

Concluding remarks

This concludes this tutorial on working with C multi-user projects.
In this tutorial, you:

- set up working environments for a team project
- created the project in the Shared Source Working Environment

- set up the build system for the project in the Shared Source Working Environment
- checked in all project files to the Repository from the Shared Source Working Environment
- built the project's executable in the Shared Object Working Environment
- worked with files (check out/in, create, add, remove) in your Private Working Environment
- updated the working environments of the project
- froze a stable version of the project in the Shared Source Working Environment

What's next

The next tutorial introduces you to version controlling in SNIFF+. In the tutorial, we will use the following three SNIFF+ tools to look at the version control information of the multi-user project you worked on in the "Team Maintenance" tutorial:

- Project Editor
- Configuration Manager
- Diff/Merge tool

Part VII

Version Controlling



File history and locking information

This tutorial assumes that you have worked through the “Team Maintenance” tutorial. During different parts of the “Team Maintenance” tutorial, you had the opportunity to work on a multi-user SNIFF+ project in three different working environments:

- Shared Source Working Environment (SSWE)
- Shared Object Working Environment (SOWE)
- Private Working Environment (PWE)

In this tutorial, starting with this chapter, you will “reconstruct” the version-control actions you performed on the multi-user project. You will first start by looking at the history and locking information of the files in the project.

Throughout this tutorial, you will be working as a Working Environments Administrator. Then, the most appropriate working environment for you to work in is the SSWE.

Looking at file history information

- In the Launch Pad, choose **Tools > Project Editor**.

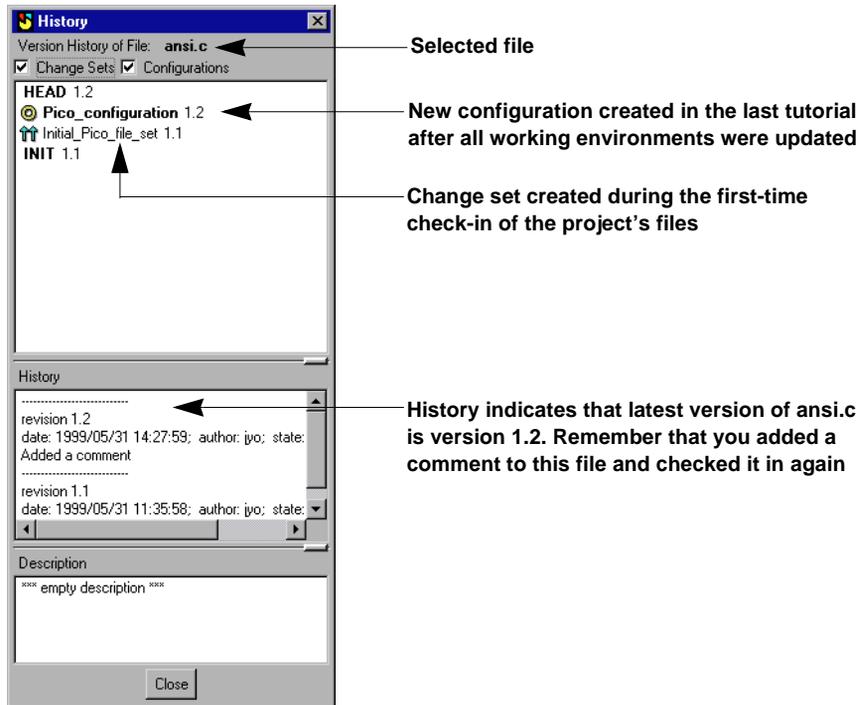
In the Project Editor

Let's look at the file history information of some of the files in the Project Editor. To do so:

1. Checkmark all projects by right-clicking anywhere in the Project Tree, and then choosing **Context menu > Select From All Projects**.
2. In the File List, highlight `ansi.c` by clicking on it once.

3. Select the **History** check box.

A new History window appears:



The history information of file `ansi.c` indicates that its latest version is 1.2. Later on, we will use Diff/Merge tool to find the differences between the latest version and the initial version (1.1) of the file.

1. Look at the history information of some of the other files in the File List. Notice that their latest versions are all 1.1. Also notice that they are all part of the configuration **Pico_configuration**, which you created at the end of the last tutorial when you froze the project.
2. In the Project Editor, clear the **History** check box.

Displaying locking information

In the last tutorial, you locked the file `ansi.c` by checking it out in the PWE. After modifying it and saving the file, you checked it in again, thus removing the lock. Now, none of the files of the multi-user project should be locked. Let's verify this by displaying the project's locking information.

1. In the Project Editor, select the **Lockers** check box.

The File List is expanded to show file locking information.

2. Scroll through the File List to look at the locking information of the files. You should notice only the name of the version control tool (RCS).

Clearly, none of the files are locked, which is what we expected.

In a more complicated multi-user team development situation, many files may still be locked after an update of team working environments. SNIFF+ correctly handles updates even when files are locked and haven't been checked in again. For details about how SNIFF+ updates working environments, please refer to the User's Guide.

3. Clear the **Lockers** check box.

Configuration and File Differences

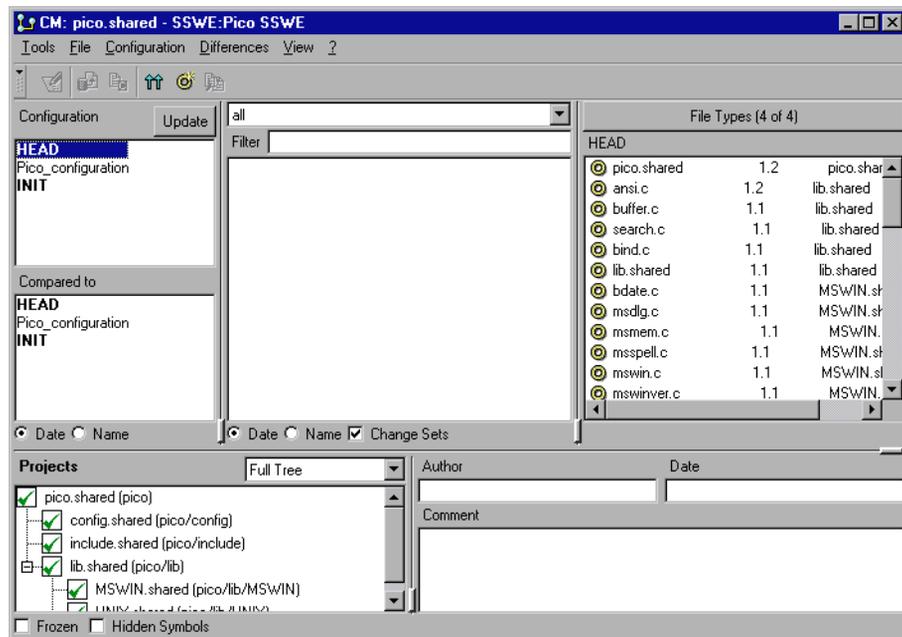
In the “Team Maintenance” tutorial, you used the Configuration Manager to create a new configuration of the project, called `Pico_configuration`. In this chapter, let’s take a look at the Configuration Manager again and use it as a starting point for comparing two file versions of the `ansi.c` file.

- In any open SNIFF+ tool, choose **Tools > Configuration Manager**.

In the Configuration Manager

1. Select the **HEAD** configuration in the Configuration List (see the following screen shot).

The project’s configuration information is loaded into the Configuration Manager. Your Configuration Manager should look similar to the following:

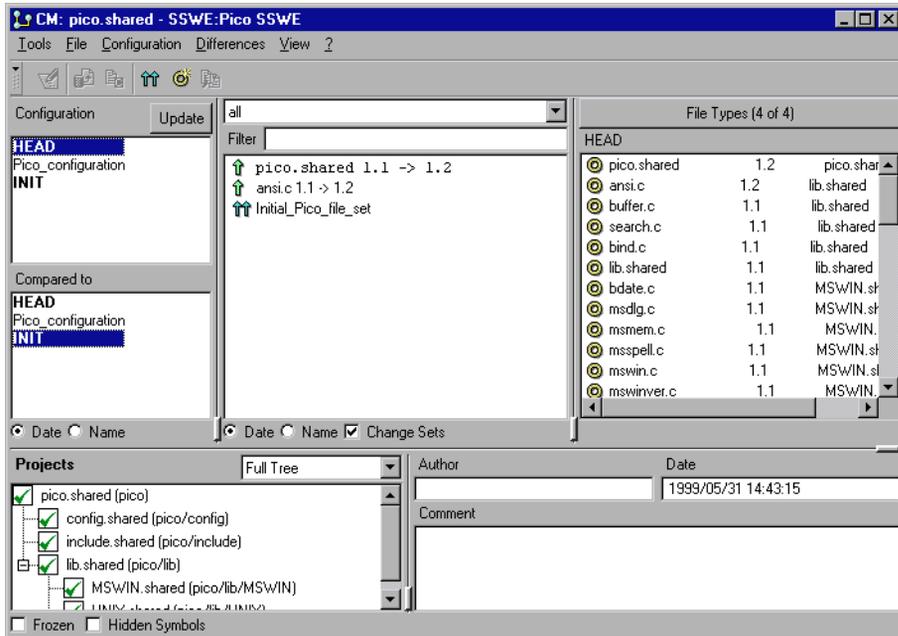


2. Select the **Pico_configuration** configuration in the Compared To List.

You should notice no difference, which means that the `HEAD` and `Pico_configuration` configurations contain the same set of files and file versions. Differences between two configurations are displayed in the Change List.

- Now, select the **INIT** configuration in the Compared To List.

The Change List should now display differences between the **HEAD** and the **INIT** configurations.



- In the Change List, notice the following line:

↑ ansi.c 1.1 -> 1.2

The line indicates that different versions of file `ansi.c` exist in the two configurations. The icon tells us that the **HEAD** configuration contains the newer version (1.2), and that the **INIT** configuration contains the older version (1.1).

Looking at file differences with the Diff/Merge tool

As the last part of this tutorial, let's look at the differences between the two versions of `ansi.c`. To do so:

- In the Change List, highlight the line:

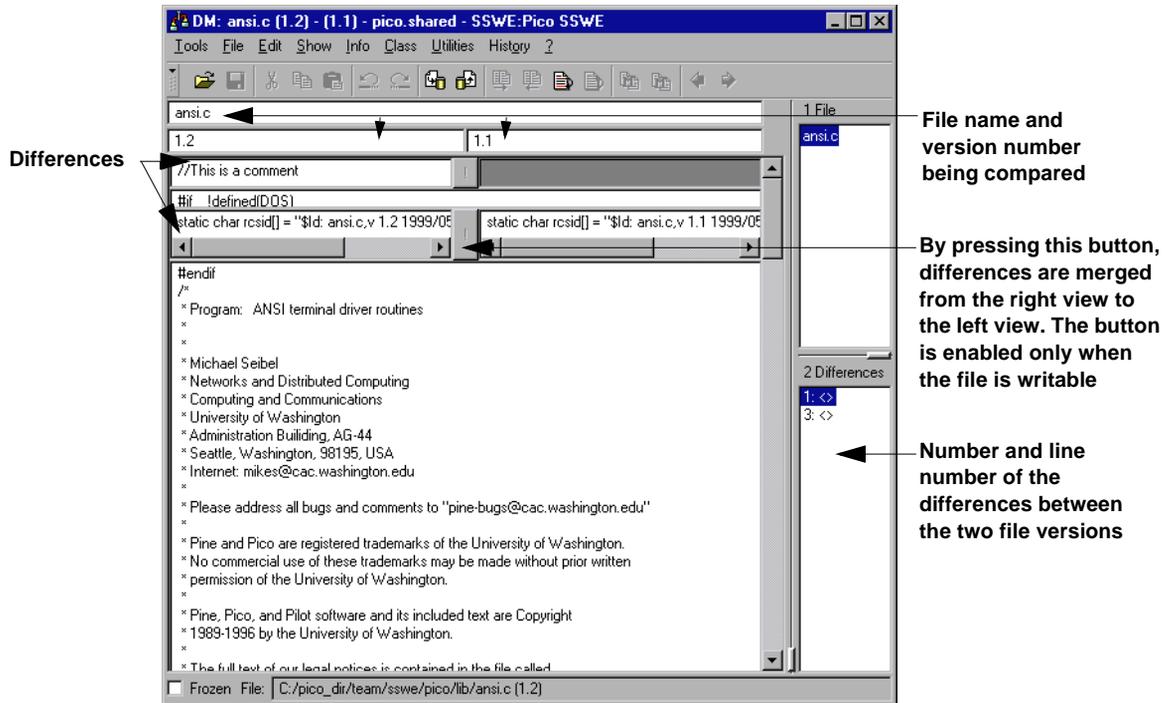
↑ ansi.c 1.1 -> 1.2

- Choose **Differences > Show Differences...**

A dialog appears, in which you are asked whether you are interested in 2-way or 3-way differences.

3. Press the **2-Way** button.

The Diff/Merge tool appears. It should be similar to the following:



Remember that you added a comment line to the first line of `ansi.c` in the "Developing in a Team" tutorial. The two differences in the screen shot are related to these changes.

4. Close the `pico.shared` project in the SSWE.
5. Quit SNIFF+.

Concluding remarks

This concludes this tutorial on looking at the version control information of a SNIFF+ project. This also concludes the C tutorial. For detailed explanations of any of the concepts in this tutorial, please refer to the *User's Guide*. To learn more about any of the SNIFF+ tools, see the *Reference Guide*. To learn how you can use SNIFF+ for your C++, Java, or Fortran projects, please refer to the respective tutorials.

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
