

SNiFF+™

Version 3.2.1 for Unix and Windows

Tornado Integration Guide



TakeFive Software, Inc.

A Wind River Company

San Jose, CA

E-mail: info@takefive.com

TakeFive Software GmbH

A Wind River Company

5020 Salzburg, Austria

E-mail: info@takefive.co.at

Copyright

Copyright © 1992–2000 TakeFive Software Inc., A Wind River Company.
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 Introduction

Road Map	9
Using this Guide	9
Feedback and useful links	9
Overview	11
The SNIFF+ Tornado Integration.	11
Installing SNIFF+ for Tornado	13
Requirements	13
SNIFF+ Installation	13
Licensing	14

Part II Browsing Source Code

Single-User Project Setup	19
Starting SNIFF+	19
The Launch Pad	19
Creating a New Project for Browsing.	20
The Project Editor	21
Opening the Project Editor	21
Adding a subproject.	22
The Project Tree	23
Saving a Project Tree view	23
Browsing Source Code	25
The Symbol Browser	25
The Source Editor	27
The Cross Referencer	28
The Hierarchy Browser	28
The Retriever.	30
The Include Browser	31

Part III Make Support and Version Control

Opening the project	35
The Launch Pad	35

SNiFF+ MakeSupport	37
Building the executable.	37
Accessing Tornado Tools	38
Understanding SNiFF+ Makesupport	39
SNiFF+ Version Control	43
Checking whether RCS is in your path (Unix Only)	43
Setting files to read-only	43
File's history information	44
Displaying locking information	45
Tracking changes in file versions	45
Configuration Management	51
Opening the Configuration Manager	51
Looking at configurations	51
Comparing two configurations	52
Part IV Team Support	
Key Concepts	57
Shared projects.	57
Working environments	57
Working with an Existing Team Project	61
The Working Environments tool	61
Setting up a Multi-User Project	65
Multi-User Project Setup.	65
Setting Up Make Support	66
Building the Executable	68
Checking all files into the Repository	68
Creating a Shared Source Working Environment.	69
Copying the shared project into the SSWE	70
Changing Working Environment Hierarchy Structure.	70
Synchronizing Working Environments	71
Part V Advanced Issues	
Working with BSPs	75
BSP Setup for Tornado-Only Makesupport	75
Version Controlling a Released BSP in SNiFF+.	77
Enabling SNiFF+ Makesupport for a BSP	78

Working with Bootable Applications	81
Setting up a SNIFF+ Project for a Customized VxWorks Project . . .	81
Removing Absolute Path References	82
Using SNIFF+ Automatic Linking of Submodules	83
Working with WindRiver Source Products	85
Working Environment Setup	85
Setting up SNIFF+ Projects for WindRiver Sources	86
Using WindRiver Sources in Own Projects	87
Advanced Issues and Customizing	89
Changing your Version Control System	89
Enabling additional SNIFF+ Parsers	89
Optimized Compilation and Changing Build Specifications	90
Extending Custom Menus	92

Part I

Introduction



Introduction

This guide describes how to install SNIFF+ into your Tornado environment and assists you in importing your first projects into SNIFF+. This guide is designed to provide new users with an accessible introduction to setting up SNIFF+ projects and using them in the Tornado environment.

Although each consecutive part and chapter is in itself more or less modular, it is assumed that you are familiar with what has gone on before.

We assume that you have gone through the Tornado Getting Started Guide. To learn about the full power of SNIFF+ browsing tools, please refer to the SNIFF+ documentation set.

What this guide is not

This is not an exhaustive guide to the SNIFF+ or Tornado products. It merely covers the integration between the two tools and points you to the original documentation for advanced issues.

Using this Guide

While simply reading the tutorial may be edifying, we encourage you to perform the steps described in this guide so that you can experience SNIFF+ first hand.

This guide consists of the following:

- Overview
- Installing SNIFF+ for Tornado
- **Part:** Browsing Source Code
- **Part:** Make Support and Version Control
- **Part:** Team Support
- **Part:** Advanced Issues

A note on Unix/Windows

Throughout this guide we will be using Windows conventions for pathnames, as well as screenshots made on Windows. These may differ slightly on Unix.

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:

- TakeFive Support Knowledge Base

<http://www.takefive.com/support/kb.html>

- Frequently Asked Questions

<http://www.takefive.com/faq>

- SNiFF+ Users Mailing List

<http://www.takefive.com/support/sniff-list.html>

- SNiFF+ Users Mailing List Archive

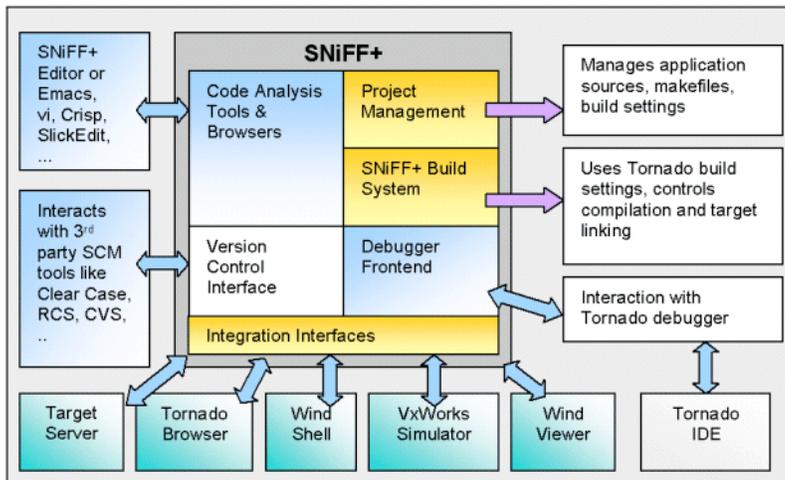
<http://www.takefive.com/sniff-list>

- TakeFive SCE Newsletter

<http://www.takefive.com/news/sce-news.html>

The SNIFF+ Tornado Integration

SNIFF+ provides an extensive collection of code comprehension, navigation, editing, build and configuration management tools. It is a highly integrated toolset that makes the daily work with your source code easier, more organized and more productive. In the illustration below, you can see how SNIFF+ fits into your Tornado Environment.



While Tornado is aimed to boost hardware start-up and to support platform (OS) and application developers without the need for team collaboration support, SNIFF+ focuses on application development with the need for advanced project management, scalable build and first class code analysis support. Both tools together offer the power of Tornado's tool chain with SNIFF+'s advanced capabilities for software development in teams.

In particular, SNIFF+ will allow you to:

- Manage all your file types in clean, hierarchically structured projects
- Use the SNIFF+ browsing and source code analysis tools to quickly navigate through, understand and develop your code
- Use clean, understandable graphical dialogs to automatically generate Makefiles for multi-language multi-platform hierarchical builds with correct dependencies across arbitrarily complex directory hierarchies
- Integrate all your Analysis & Design, Documentation and Test Tools into a single comprehensive development environment

- Use your favorite Configuration Management System to put your work into managed workspaces, share projects among your development teams, synchronize their work and get change reports through an easily understandable, powerful GUI.

Installing SNIFF+ for Tornado

This chapter covers

- [Requirements — page 13](#)
- [SNIFF+ Installation — page 13](#)
- [Licensing — page 14](#)

Requirements

You will need to have Tornado II installed before you install SNIFF+ and the integration. A 'basic' installation of SNIFF+ for Tornado needs approximately 43,5MB of disk space while a full installation needs 71MB (on Windows) up to 120MB (on Unix). The full installation contains additional components like an Ada Parser, the Visaj Java GUI Builder and Adobe Acrobat files for the printed documentation.

For details about required versions and patches, please refer to the Integration Release Notes (%SNIFF_DIR%\integrations\TornadoII\README.html).

SNIFF+ Installation

Log on using the same user name used when installing Tornado. If you downloaded SNIFF+ from the Internet, change the directory to the folder where you placed your downloaded files. If you have a SNIFF+ CD, please insert it into your CD-ROM drive and change the directory to the toplevel folder. Now, execute:

On Windows

```
Setup.exe
```

On Unix

```
./install.kit
```

In the Setup Wizard

1. Install SNIFF+ into any path that does not contain space characters: we suggest C:\sniff or C:\Tornado\sniff. Please note that C:\Program Files\sniff will not work!
A single SNIFF+ installation can handle multiple Tornado environments. We will refer to the SNIFF+ installation directory as SNIFF_DIR throughout this guide.
2. In the following Wizard pages, accept the defaults. When asked which packages to install, select “**basic**” and “**example**”, the other packages are optional.
3. Accept all defaults for the rest of the setup.

Preparing the Environment

Once SNIFF+ and Tornado are installed, you need to run the Tornado integration install script, which is located in the root of the Tornado Integration directory.

On Windows, run

```
%SNIFF_DIR%\integrations\TornadoII\install.exe
```

On Unix, run

```
$SNIFF_DIR/integrations/TornadoII/install.sh
```

You may accept all defaults during the installation process. When the installation of the Tornado integration is complete,

On Windows

A SNIFF+ for Tornado icon is put on your desktop

On Unix

The installation program will create startup scripts (`sniff.cshrc` and `sniff.profile`) in your `$WIND_BASE` directory. Depending on your login shell, you should merge the appropriate script with your shell startup script.

Licensing

SNIFF+ comes with a pre-installed personal license, which will allow you to use the product for evaluation and non-commercial projects of up to 200 files. This will suffice for working through most of this guide. For working with Board Support Packages (BSPs) in SNIFF+, we recommend that you obtain a full evaluation license. This will enable all SNIFF+ features for projects of arbitrary size, but for a limited time. You can request both evaluation and full product licenses from your local WindRiver Representative. Contact information is available from:

- Corporate Headquarters, US and Canada
inquiries@windriver.com
- France, Southern Europe, Middle-East, Africa
inquiries-fr@windriver.com
- Germany, Austria, Switzerland, Central Europe
inquiries-de@windriver.com
- UK, Western Europe
inquiries-uk@windriver.com
- Northern Europe
inquiries-se@windriver.com
- Benelux
inquiries-benelux@windriver.com
- Italy
inquiries-it@windriver.com

- Israel
inquiries-il@windriver.com
- For the Far East, South America, Australia, New Zealand and any other countries not listed here, please refer to
<http://www.windriver.com/corporate/html/contact.html>
for contact information.

Part II

Browsing Source Code



Single-User Project Setup

The Single User Project Setup, also known as Browsing-Only Setup, is the easiest way to look at your code with SNIFF+. It enables you to use all the Browsing and Editing Tools as well as some basic level of version control. SNIFF+ Makesupport cannot be used in this configuration, but you may use your own hand-written Makefiles together with SNIFF+.

This chapter is about

- starting SNIFF+ and creating a project for browsing your source code

We assume you have successfully installed SNIFF+. If not, please refer to [Installing SNIFF+ for Tornado — page 13](#).

Starting SNIFF+

On Windows

- The installation program should have created an icon, called **SNIFF+ for Tornado**, on your desktop. Double-click on this icon to start SNIFF+ (do not use the standard SNIFF+ for Windows icon).

On Unix

- Incorporate the code from `sniff.cshrc` or `sniff.profile`, which has been put in your `$WIND_BASE` directory, into your shell start-up file. Since these files extend your `PATH` setting, you may then start SNIFF+ by just typing `sniff &` on the command line.

When you first start SNIFF+, the Welcome dialog appears. From this dialog, you can check your SNIFF+ installation. If errors are reported at this stage and if you need assistance, please contact us (see [Feedback and useful links — page 9](#)).

If everything is okay, you can close this dialog. The Launch Pad will remain on the screen.

The Launch Pad

The Launch Pad is the main application window and serves to manage projects and open tools on your desktop. The **Help (?)** menu has supplementary commands that are available only in the Launch Pad.

Note

On Unix, you need the Netscape Browser to look at SNIFF+ online documentation.

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.

Creating a New Project for Browsing

We will now set up a SNIFF+ project for the `factory` C++ example, which is also part of your Tornado installation. Using a C++ example allows us to show both object-oriented and non object-oriented browsing tools. For other Languages like Ada or Java, see the respective SNIFF+ Tutorials, which are available from the Welcome dialog or the **Help (?)** menu in the Launch Pad.

To work through the `factory` example:

1. In the Launch Pad, choose **Project > New Project > with Template...**

The Project Template dialog appears.

2. In the Available Template Files list, select the `Tornado_BrowsingOnly.ptmpl` template.

3. Press the **Change Directory...** button.

4. In the Directory dialog that appears, navigate to

`SNIFF_DIR\example\Tornado\1_BrowsingOnly\factory`
and press **Select**.

5. Press **OK**.

The Attributes of New Project dialog appears. Thanks to the Project Template, you may leave all settings as they are. Don't be confused by the many options you can set, the template will be good for 99% of your own C/C++ projects, too. In fact, there are only two settings that are important at this point:

- In the **General** node, there is the **Ignore Directories** field. It allows you to specify (with GNU Regular Expressions) any directories that SNIFF+ should not take into account when creating its project descriptions.
- In the **File Types** node, you may change the list of file types that SNIFF+ should add to its project description files.

For more information, please refer to the SNIFF+ Reference Guide.

6. For now, you may just press **OK** to set up your project.

SNIFF+ will now set up the project and parse your files.

7. In the dialog that appears asking if you want to generate cross reference information, press **No** since SNIFF+ will automatically generate cross reference information as soon as it is needed.

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

The Project Editor

The Project Editor is used to edit and browse project-specific information, including project attributes, subprojects, files and version control and locking information. A project is the main structuring element in SNIFF+ for grouping together files and directories that logically belong together.

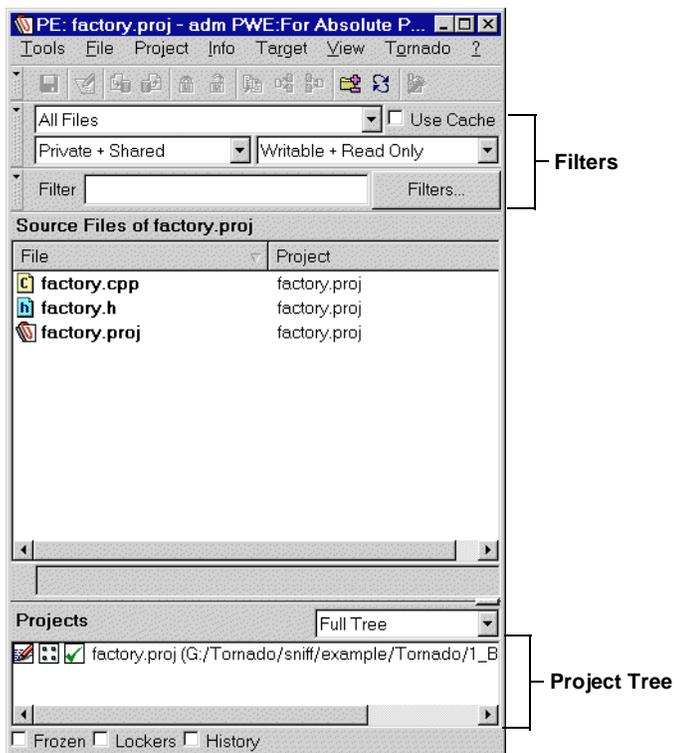
This chapter is about

- project filtering in the Project Editor
- adding a subproject to an existing project
- saving a project set

Opening the Project Editor

The Project Editor is opened automatically when you create a new project or open an existing project. Since we just set up a SNIFF+ project for the factory example, `factory.proj` is now displayed in the Project Editor.

- To open the Project Editor from any other SNIFF+ tool, you would choose **Tools > Project Editor**; make sure the project you want to open is highlighted in the Launch Pad.



Adding a subproject

We will now add a subproject for the VxWorks C++ headers.

1. In the Project Tree, highlight `factory.proj` by clicking on its name.
2. Choose **Project > Add Subproject to factory.proj**.
3. In the Load Subproject Named dialog that appears, navigate to `SNIFF_DIR\PDFs\VxWorks5.4`
4. Select the `VxWorks5.4_h_Cplusplus.proj` project and press **Open**.

SNIFF+ will add the VxWorks C++ Headers to your Project Tree. The projects are separated into the C++ Standard Classes (Strings, Exceptions and Iostream), the Standard Template Library (STL) and the Wind Foundation Classes (WFC). You can verify that this has been done by taking a look at the Project Tree. Notice also that the icon next to `factory.proj` has changed to warn you that the project has been modified and not yet saved.

- To save the project, choose **Project > Save factory.proj**.

The Project Tree

Having structured project descriptions for VxWorks has the advantage that you can focus your interest exactly on those portions of code that you need, and you won't be distracted by parts that you are not interested in. You may select modules for browsing simply by selecting SNIFF+ project description files (source modules) as subprojects.

Checkmarking Projects

In the Project Tree, click into the checkboxes (left of the project names) to show/hide files in the File List. In SNIFF+, a project that has a checkmark in its checkbox is called a *check-marked project*.

1. In the Project Tree, click into the checkbox next to `h_STDCPP.proj` to clear it and notice what happens in the File List.

All the files in the `h_STDCPP.proj` project are no longer shown in the File List.

2. Click into the checkbox next to `h_STDCPP.proj` again to select it.

The files in this project are now shown.

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.

1. Click on the node of `VxWorks5.4_h_Cplus.proj` to collapse it.
2. Try alternately checkmarking and clearing the checkbox next to the collapsed node.

When the project is checkmarked, all the files in `VxWorks5.4_h_Cplus.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.

Saving a Project Tree view

Rather than always resetting the Project Tree, you can save a view of the Project Tree to reuse later.

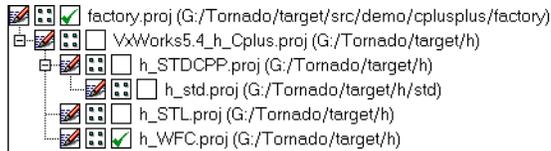
Preparing the Project Tree

- Expand the nodes of all projects and checkmark only

factory.proj

h_WFC.proj

Your Project Tree should now look like this:



Saving the Project Set

1. From the menu, choose **View > Save Project Set**.
2. In the dialog that appears, enter a name for the view of the Project Tree as it appears now, e.g., `factory_with_h_WFC` and press **Ok**.

We will use this name to refer to the project set when we next use it.

Note that you can save and reuse project sets using the **View** menu in any tool that has a Project Tree.

Browsing Source Code

SNiFF+ contains an extensive set of browsing tools to make the daily work with your source code easier and more productive. Some of these tools are briefly described in this chapter, for more information please refer to the SNiFF+ documentation.

Tools described in this chapter:

- Symbol Browser
- Source Editor
- Cross Referencer
- Hierarchy Browser
- Retriever
- Include Browser

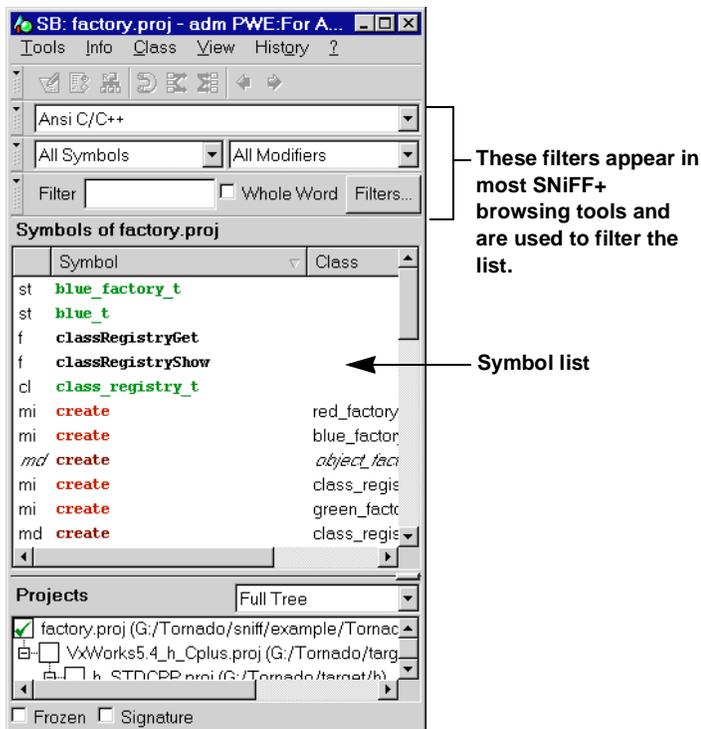
Note that you can open all SNiFF+ tools using the **Tools** menu.

The Symbol Browser

In the Symbol Browser you can easily find program elements such as functions, global variables and classes and filter symbol information from a large symbol scope.

- From any open SNIFF+ tool, choose **Tools > Symbol Browser**.

The Symbol browser displays all symbols in the `factory.proj` project as well as in all subprojects.

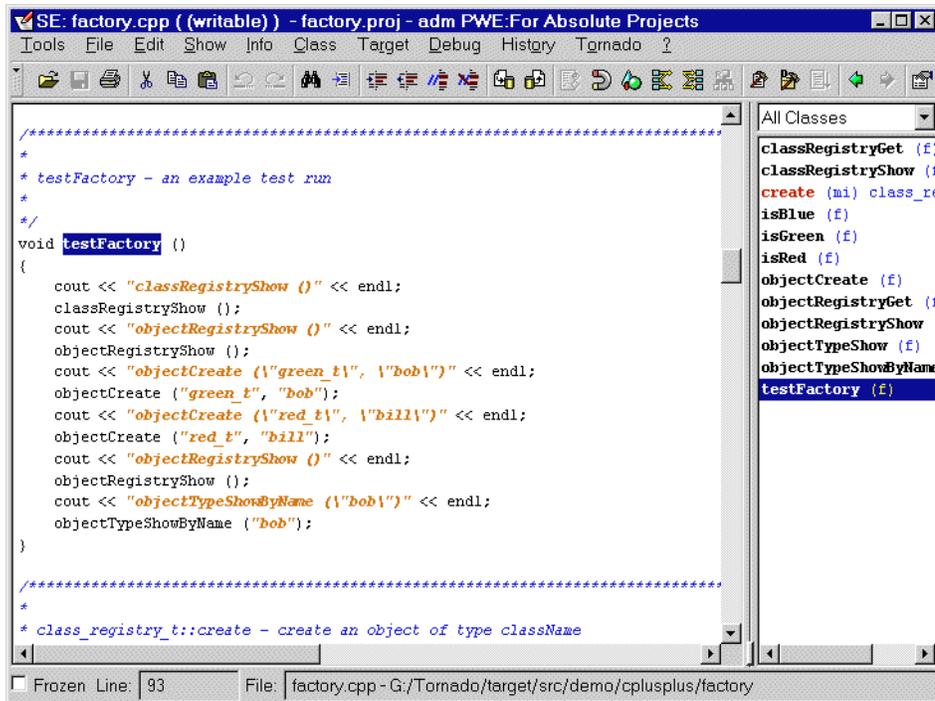


Let's now filter the Symbol list to show functions only. To do so:

- In the Symbol drop-down, choose **function**.
Only the functions are displayed.
- Click into the Symbol list and type `te` to navigate to the `testFactory` function.
- Double-click on this function to position at its definition in the Source Editor.

The Source 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 class names, method names and comments. When a source file is modified and saved, its symbol information is immediately updated.



1. In the definition of `testFactory`, highlight `objectCreate` and choose **Show > Symbol(s) objectCreate...**

The Source Editor is now positioned at the `objectCreate` function.

2. Press the **green back arrow** in the button bar to position the Source Editor at the `testFactory` function once more.

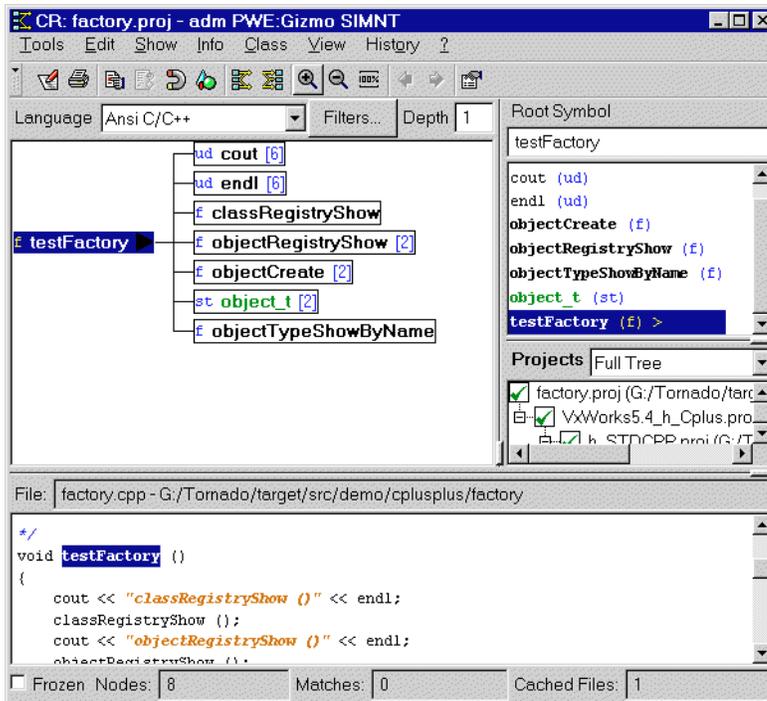
Lets now look at the call graph of the `testFactory` function in the Cross Referencer. To do so:

- Make sure that `testFactory` is highlighted and choose **Info > testFactory Refers-To**.

The Cross Referencer opens.

The Cross Referencer

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



Lets look at the backward references of struct `object_t`. To do so:

1. Highlight `object_t` in the call graph.
2. Right-click in the Graph View and choose **object_t Referred-By**.

The backward references of the struct are now displayed.

Let's now look at `object_t` in the entire class hierarchy. To do so:

- Make sure that `object_t` is highlighted and choose **Class > Show object_t in Entire Hierarchy**.

The Hierarchy Browser opens.

The Hierarchy Browser

The Hierarchy Browser shows the inheritance relationships of classes. It either displays the entire class tree or only the superclasses and subclasses of a class.

Lets now use the Project Set which we saved in [Saving a Project Tree view — page 23](#) to restrict the graph. To do so:

1. Choose **View > Select Project Set > factory_with_h_WFC**.

Now only classes in `factory.proj` and `h_WFC.proj` are shown in the Hierarchy view. Abstract classes are shown in blue typeface.

The screenshot shows the Hierarchy Browser interface for the project 'factory.proj'. The main window displays a class hierarchy tree. The root node is 'object_t (st)', which is highlighted in blue. It has three children: 'blue_t (st)', 'green_t (st)', and 'red_t (st)'. Below this, there are other classes like 'registry_t<class T>', 'VXWError', 'VXWMemPart', 'VXWModule', 'VXWMsgQ', 'VXWFingBuf', 'VXWObject', 'VXWSem', 'VXWBSem', 'VXWCSem', 'VXWMSem', 'VXWSmName', 'VXWSmMemBlock', 'VXWSmMemF', and 'VXWSmMsgQ'. The right pane shows a list of classes, with 'object_t (st)' selected. The bottom pane shows the source code for 'struct object_t'.

```

struct object_t
{
    virtual void method () {}
};

struct red_t : object_t

```

2. We will now look at the immediate relatives (subclasses and superclasses) of `object_t`. Make sure that `object_t` is highlighted in the Hierarchy view, right-click and choose **Show object_t Relatives**.

The Hierarchy view now displays only the relatives of `object_t`.

Lets now change all occurrences of the base structure `blue_t` to `skyblue_t` in all projects. To do so:

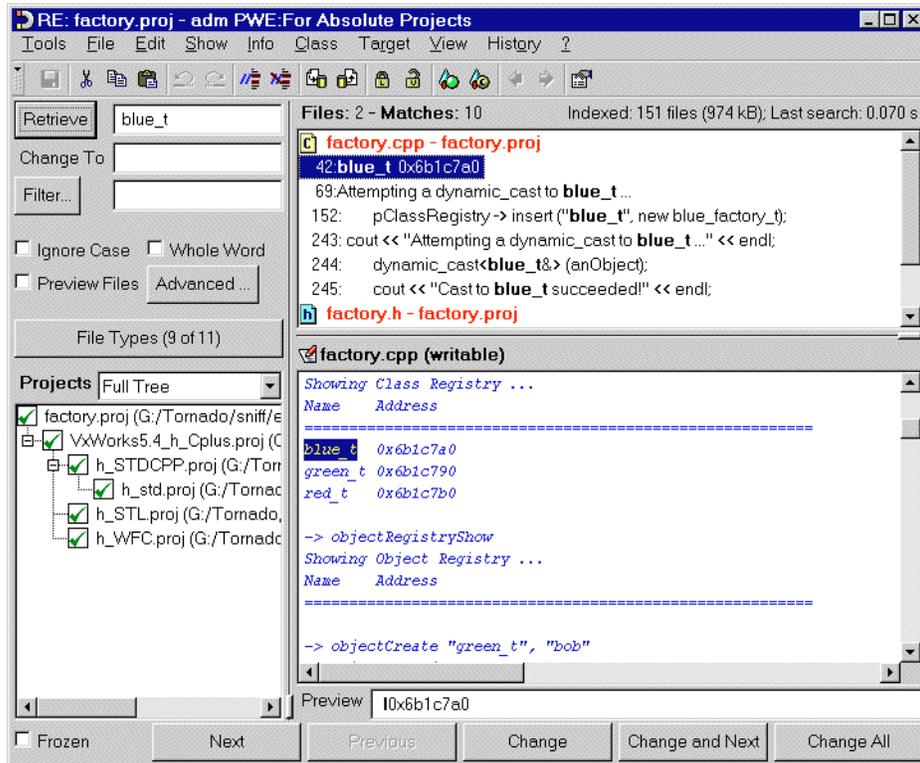
- Highlight `blue_t` in the Graph view and choose **Info > Retrieve object_t From All Projects**.

The Retriever opens.

The Retriever

The Retriever is a global textual retrieve-and-replace tool, whereby regular expression filters can be used for complex retrievals and modifications. For an introduction to regular expression syntax, please refer to the SNIFF+ Reference Guide.

Since we've just retrieved `blue_t` from all projects, all occurrences of `blue_t` are displayed.



Now, to change `blue_t` to `skyblue_t`:

1. In the **Change To** field, enter `skyblue_t`.

Take a look at the **Preview** field below the integrated Source Editor, the code line (highlighted in the Files — Matches List) is shown as it would appear after modification. You can use the **Next** button at the bottom of the tool to look at each line as it would appear after being changed.

2. To change all occurrences of `blue_t` to `skyblue_t`, press the **Change All** button.
3. In the dialog that appears, press **Yes**.

All occurrences of `blue_t` are now changed to `skyblue_t`. You can verify this by pressing the **Retrieve** button again to requery.

Although the changes you made above are harmless, knowing how to undo global changes is not a bad idea. To do so:

- Choose **Edit > Undo Change All**

When making global changes, you may be interested in knowing which files include the affected header file. Lets see which files include `factory.h`. To do so:

- In the Files Matches List, highlight **factory.h - factory.proj** and choose **Info > factory.h Is Included-By**.

The Include Browser opens.

The Include Browser

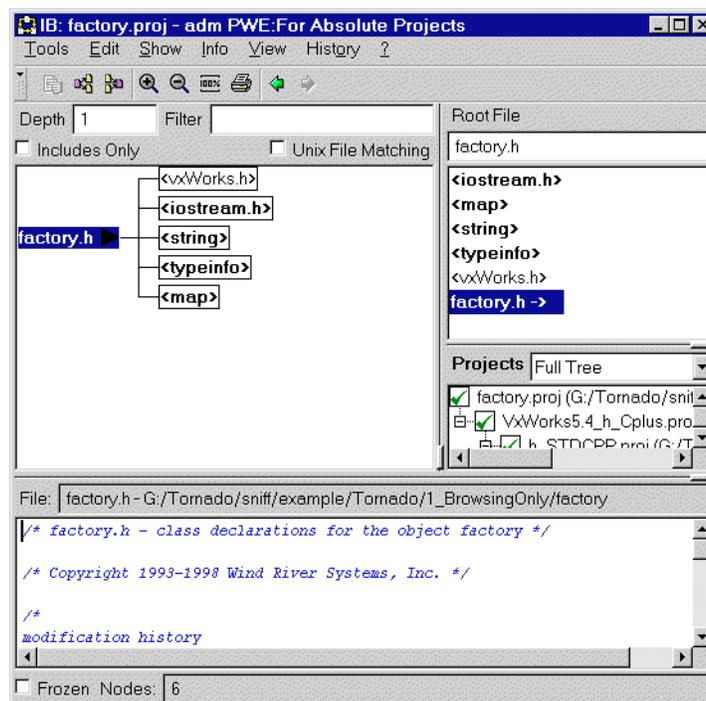
The Include Browser graphically displays include references made in the project source files. It can be used to see which files are included by a particular file and vice-versa, as well as to make sure that there are no redundant includes.

You will see that `factory.cpp` is the only file that includes `factory.h`.

Lets now look at which files `factory.h` includes. To do so:

1. Make sure that `factory.h` is highlighted, right-click and choose **Includes**.

The header files which `factory.h` includes are shown.



We will now query to the next level.

2. In the **Depth** field, enter 2.
The header files are displayed to 2 levels.
3. To see which projects these files belong to, choose **View > Show Project Name**.
Scroll to the right of the Graph view. You can now see the projects to which the files belong.

What's Next

You should now be familiar with the most common SNIFF+ browsing tools, so we can now close `factory.proj`. To do so:

- In the Launch Pad, select `factory.proj` and choose **Project > Close Project *Project***.

We recommend that you now set up a browsing project for your own source code in the same way as you did with the factory example code. You should also try to add new files to your project (Project Editor: **Project > Add New File to *project*** or **Project > Add/Remove Files to/from *project***).

If you prefer to continue with the next part, you will learn about SNIFF+ MakeSupport as well as Version Controlling and Configuration Management. Please note that you won't be using the `factory.proj` project but rather an already set up project called `gizmo.shared`.

Part III

Make Support and Version Control



Opening the project

This part shows you how SNIFF+ interacts with your Tornado Environment to *make*, run and debug your programs. Here we will use an already-setup project called Gizmo.shared to show you how easy it is for developers to work with SNIFF+ MakeSupport. By looking at the Project Attributes we have set, you will also learn how to set up your own projects.

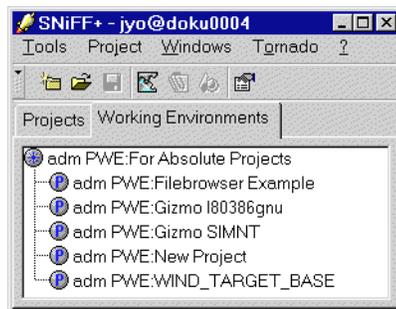
In this chapter, you will

- open an existing project

The Launch Pad

The Launch Pad automatically opens when you start SNIFF+.

- Select the **Working Environments** tab.



Working environments (WE's) are based on physical directories on your file system in which SNIFF+ projects reside. A SNIFF+ Working Environment is similar to a Tornado Workspace. We will look at Working Environments in detail in the Team Support part of this guide.

To open the example project:

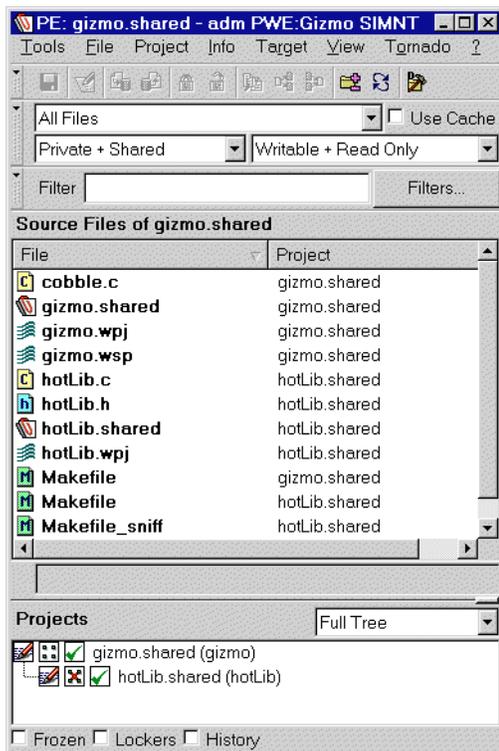
1. Double-click on the Working Environment that matches your Tornado Simulator type:
 - adm PWE: Gizmo SIMNT (on Windows NT).
 - adm PWE: Gizmo SIMSPARCSOLARIS (on Solaris)
 - adm PWE: Gizmo SIMHPPA (on HP-UX)

If there is no Simulator for your host type, you may use any other adm PWE: Gizmo (*any target*) instead. Make Support will work in the same way as in the Simulator Environments.

The Open Project dialog opens.

2. In the Open Project dialog, press the **Update List** button to display the projects in this Working Environment.
3. From the Project List, select `gizmo.shared` and press **Open**.

SNiFF+ parses all the symbol files in the project and loads the project with symbol information. After the project is opened, SNiFF+ displays the project's structure and contents in a Project Editor, which should look like the one illustrated below.



Note

The **Gizmo** application shown here is a modified version of the example shown in the original Tornado Getting Started Guide. The example is modified to contain both a Library and a main module to demonstrate SNiFF+ automatic linking of submodules.

This chapter is about:

- building the project's executable
- how SNiFF+ interacts with the Tornado Environment to *make*, run and debug programs

Building the executable

In the Project Editor

Notice in the Project Editor's File list that there's a file called `Makefile` and a file called `Makefile_sniff` in each project. `Makefile` is the one which is automatically created by Tornado; `Makefile_sniff` is automatically generated and maintained by SNiFF+, so both Make systems work in parallel.

All commands needed to compile, run and debug targets can be found in the Project Editor's **Target** menu.

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.

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

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



2. Press **Yes**.

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

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

3. In the Project Tree, highlight `gizmo.shared` by clicking on its name.

4. Choose **Target > Make > all** to recursively build both the library (which is a subproject) and the main application.

A Shell Tool appears, in which the `make all` command is executed in each project directory.

On a plain host-based system you could choose **Target > Run** to test your application. But, since we are working in a Cross Compilation Environment, we will now use the Tornado menu to access the various Tornado Tools.

Accessing Tornado Tools

As you must have already noticed there is a **Tornado** menu in the Project Editor menu bar. We will use the commands in this menu to access Tornado tools.

1. Choose **Tornado > Launch Simulator...** to start a VxWorks simulator on your machine.
2. In the dialog that appears, press **OK**.
3. Now choose **Tornado > Target Server...** to see your Target Server's properties.

Note

When you run this example on a real target, you can use this menu to connect to a running target server (in order to start a new Target Server, please use the standard Tornado IDE, which you can start with **Tornado > Launch Tornado IDE**).

4. In the Select Target Server dialog, disable the **Show before every tool start** checkbox and press **OK**. By doing so you can select a current target server for all future operations.
5. Choose **Tornado > Download [...]gizmo.out...** to download your application to the target.

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

Note

If the **Download** menu item does not start **[...]gizmo.out**, or you get an error during the download, please click into the Project Editor's File list and press the <ESC> key to remove any current selection. Now, select `gizmo.shared` in the Project Tree again and try to download again.

6. Choose **Tornado > WindSh...** to connect a WindShell to the Target.
7. In the WindSh, type `moduleShow` to verify that the module has loaded and type `progStart` to run the program.

You should get "0x0" as output in the WindSh, and soon thereafter the VxSim Window should display messages like "Warning - HOT".

8. Back in the Project Editor, select **Target > Debug** to open the SNIFF+ Debugger tool. It will automatically connect to your currently selected target server (this is only possible if an application has already been downloaded). In the SNIFF+ Debugger tool, type `attach tMonitor` to attach to the running program. Depending on where the program is currently executing, the SNIFF+ Source Editor should open up with an additional button bar for debugging.

If the Source Editor doesn't open automatically:

- Open the Project Editor and double click on `cobble.c`
- Select `monitor` from the symbol list on the right hand side of the Source Editor (as you get more familiar with SNIFF+, you may directly use the Symbol Browser for this task, so that you don't need to find the right program file first!)
- Click on the line containing the `if` statement (line #329)
- Click the **Break At** button in the Source Editor's Debugger Button Bar: an exclamation mark showing your breakpoint will appear
- Click the **Continue** button: An arrow showing your instruction pointer will appear at the breakpoint.

The SNIFF+ Debugger

The advantage of the SNIFF+ Debugger as opposed to CrossWind is that it has full browsing support, and thus it is particularly useful for object-oriented programs. You may particularly like the Cross Referencer's **Refers To** query (see [The Cross Referencer — page 28](#)) to show call graphs which allow you to quickly set breakpoints.

The disadvantage of the SNIFF+ Debugger is its limited capability of data display. Basically, data structures can just be printed or watched in the SNIFF+ Debugger Tool, which is a pure text-based interface to `gdb`. However, on UNIX it is possible to use the DDD (Data Display Debugger) as a front-end between the SNIFF+ Debugger Tool and the Tornado `gdb` backend. This will allow for very advanced data display capabilities. For more information about DDD, see <http://www.gnu.org/software/ddd/>

Understanding SNIFF+ Makesupport

- To understand how SNIFF+ Makesupport works, you should now close the Source Editor and the Debugger tool.

On Windows

The Tornado `gdb` may issue an Application Error message, which can be safely ignored since all data is saved at that point

1. Look at the File list in the Project Editor.

You will notice that there is a file called `Makefile` and a file called `Makefile_sniff` in each project. `Makefile` is the one which is automatically created by Tornado, `Makefile_sniff` is automatically generated and maintained by SNIFF+, so both Make systems work in parallel. While Tornado re-writes its entire Makefile whenever the Project is changed, SNIFF+ takes a different approach.

2. Double-click on `Makefile_sniff` to open it in the Source Editor:

You will see that this is a Template, which includes other files like `macros.incl` from the directory `.sniffdir`. `Macros.incl` is one of 4 different Make Support Files which are generated by SNIFF+ when you choose the **Target > Update Makefiles** menu. Together with a set of general Makefiles from the `$SNIFF_DIR/make_support` directory, these configure the Make Support. `Makefile_sniff` itself, however, is never modified by SNIFF+, it can be used to change any settings that are not available through the GUI.

Looking at Project Attributes

The standard way of customizing SNIFF+ Makesupport is not by editing the Makefile, but in the GUI, by editing the SNIFF+ Project Attributes.

1. To look at (or modify) the attributes of the `hotLib.shared` project, double click on `hotLib.shared` in the Project Editor's Project Tree.

The SNIFF+ Project Attributes dialog opens.

2. In the Project Attributes dialog, select the **Build Options** node.

Here make sure that the **Use SNIFF+ Make Support** checkbox is selected.

The **Make Command** field may contain the name of your Make program plus additional command line options. If this field is empty, the SNIFF+ Platform default will be used, which is `sniffmake DEBUG=1` for Tornado.

The **General Targets** field may contain a list of targets for Make that you would like to see in the Project Editor's Target menu. Entries are separated by colons; each of the entries will be shown as a menu entry and will be passed to your make program on the commandline.

3. Select the **Directives** node.

When you press the **Generate** button, SNIFF+ automatically generates include directives for the Compiler.

In the **Additional** field you may enter more "external" include directives which should not be overwritten when SNIFF+ generates include directives.

In the **Preprocessor Directive(s)** field you may enter `-DSOMETHING` directives for your Compiler/Preprocessor.

The **Directives list** above these, allows you to have separate `-D` or `-I` directives for various Build Specs or Platforms. For more information, please refer to the SNIFF+ Reference Guide or Online Documentation.

4. Select the **Project Targets** node.

Here you can enter the names of any targets that SNIFF+ Makesupport should generate.

5. For the hotLib example, we build a Library so make sure that `hotLib.a` appears in the **Library** field.
SNIFF+ allows only one master target per project, which may either be an executable or a library or a relinkable object, or a shared library (but no two of them at the same time). Additional targets may be stated in the **Other** field, separated by colons.
6. Select the **Build Structure** node.
Here you specify which of the targets (if any) should be exported to the superproject. Since we build a library in `hotLib.shared`, this library should also be automatically linked to any superprojects (in our case `gizmo.shared`), thus, **Library** is entered in the **Passed to Superproject** field.
7. Press **OK** to close the Project Attributes dialog.
You won't need the **Advanced** node so you can safely ignore it.
8. Now open the Project Attributes of `gizmo.shared` by double clicking on it.
9. In the Project Attributes dialog, select **Build Options > Project Targets**.
Notice that `gizmo.out` is entered in the **Executable** field.
10. Select the **Build Structure** node.
Notice that `../hotLib` is entered in the **Recursive Make Dir(s)** field.

Attributes of Multiple Projects

It is also possible to modify the attributes of multiple projects by doing the following:

- In the Project Tree, checkmark the projects that you are interested in and choose **Project > Attributes of Checkmarked Projects**.
The Group Project Attributes dialog opens. The Group Project Attributes dialog is an extended Project Attributes dialog that lets you set the project attributes of multiple projects listed in the Project Editor's Project Tree. For more information, please refer to the SNIFF+ User's Guide.

Using Tornado Target Tools from SNIFF+

In the Project Editor

On Unix:

- The most important Tornado target tools can be directly started from the Project Editor's **Tornado** menu. To open Tornado target tools, choose **Tornado > Launcher...**, **Cross-Wind...**, **Browser...**, **ProjectTool...**
- The tools will automatically use your currently selected target server. In some of the Tornado Tools (CrossWind in particular), even navigation back to SNIFF+ is possible by using a custom menu.
- For other tools, please use **Tornado > Launch Tornado IDE**: it will open the Tornado Launcher, from which you can start all other target tools.

On Windows:

- Since the Tornado IDE is not as open for external interaction as on UNIX, it is not possible to directly start target tools from SNIFF+.
- You may start the Tornado IDE in the context of a particular Tornado project simply by double-clicking on a `.wpj` or `.wsp` file in the Project Editor's File List.
- If you do not have a Tornado Project for a particular SNIFF+ Project, you can start the Tornado IDE by choosing **Tornado > Launch Tornado IDE**.

In Tornado

- You will be able to connect to a target server that you started from SNIFF+, and use all target tools, even if you do not have a Tornado Project open.
- In order to start a target server for a real target (not a simulator), you will always need to use the Tornado Launcher (Unix) or Tornado IDE (Windows).

SNiFF+ Version Control

In this chapter you will:

- look at a file's history information
- look at locking information#
- track changes in file versions

Checking whether RCS is in your path (Unix Only)

Note

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

The Tornado example comes with an RCS repository. This tutorial relies on RCS 5.7 which is supplied together with the SNiFF+ package and is installed from there.

To check whether the correct version of RCS is in your path:

1. Open a Unix shell.
2. Enter `% rcs -V1` at the command prompt.

You will receive **one** of the following outputs:

- `rcs error: -V1 out of range 3..5`

This shows that the **correct version** of RCS is installed.

- `rcs error: Unknown option: -V1`

This shows that an old, **unusable version** of RCS is installed. If you get this output, please install RCS 5.7 (supplied together with the SNiFF+ package) to execute the version control steps in the next chapter.

Setting files to read-only

Before proceeding with this chapter, we suggest that you set all files in the Project Editor's File List to read-only.

To set project files to read-only, do the following:

1. In the Project Tree, make sure that all projects are checkmarked.
2. Click into the File List.

3. Right-click and choose **Select All**.
4. Right-click and choose **Set Read Only**.

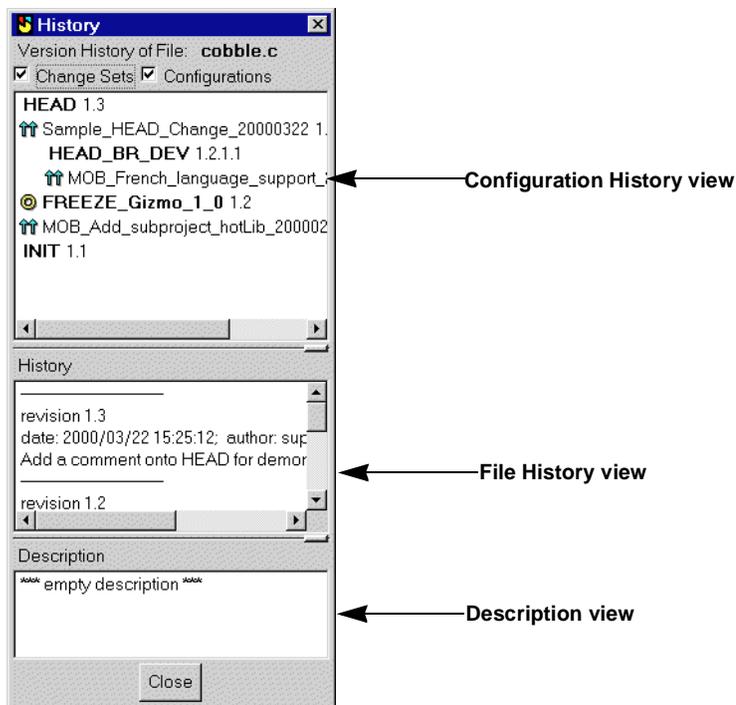
Notice that the files in the File List are no longer in bold typeface. This means that they are now read-only. The icons in the Project Tree have also changed to indicate that the projects, too, are read-only.

File's history information

Lets look at the history information of `cobble.c`. To do so:

1. Highlight `cobble.c` in the File List.
2. Select the **History** check box.

A History window appears to the right of the Project Editor. The History window contains three views that show the history information of the selected file. To see a description of the icons used, choose **Help(?) > QuickRef**.



All versions of `cobble.c` as stored and maintained by your version control tool are displayed. In the **File History** view, you can see the complete history of `cobble.c`. To see the history information of a particular version of `cobble.c`:

- Highlight the *change set* `MOB_French_language_support_20000314` in the **Configuration History** view. A change set is a set of files checked in at the same

time under the same symbolic name.

You can now only see the history record (stored and maintained by your version control tool) of `MOB_French_language_support_20000314` in the **File History** view.

- Close the History window.

Displaying locking information

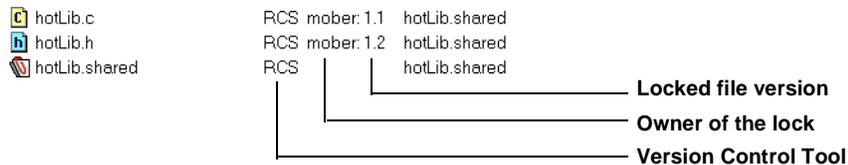
You may now want to see which files are locked. To do so:

1. Make sure that the `hotLib.shared` project is checkmarked in the Project Tree.
2. In the Project Editor, select the **Lockers** check box.

In the File List, the Lockers column appears showing locking information.

3. Look at the files `hotLib.c` and `hotLib.h` in the File List.

In the illustration below, you can see that the version control tool that is used is `RCS`, the file is locked by `mober` and the locked version number is `1.1` for `hotLib.c` and `1.2` for `hotLib.h`.



Tracking changes in file versions

Lets now take a look at which files have been modified and what changes have been made to these files.

1. In the Project Editor, choose **Modified** from the File Status drop-down (next to the Use Cache checkbox).
2. In the Files Compared to dialog that appears, press **Ok**.

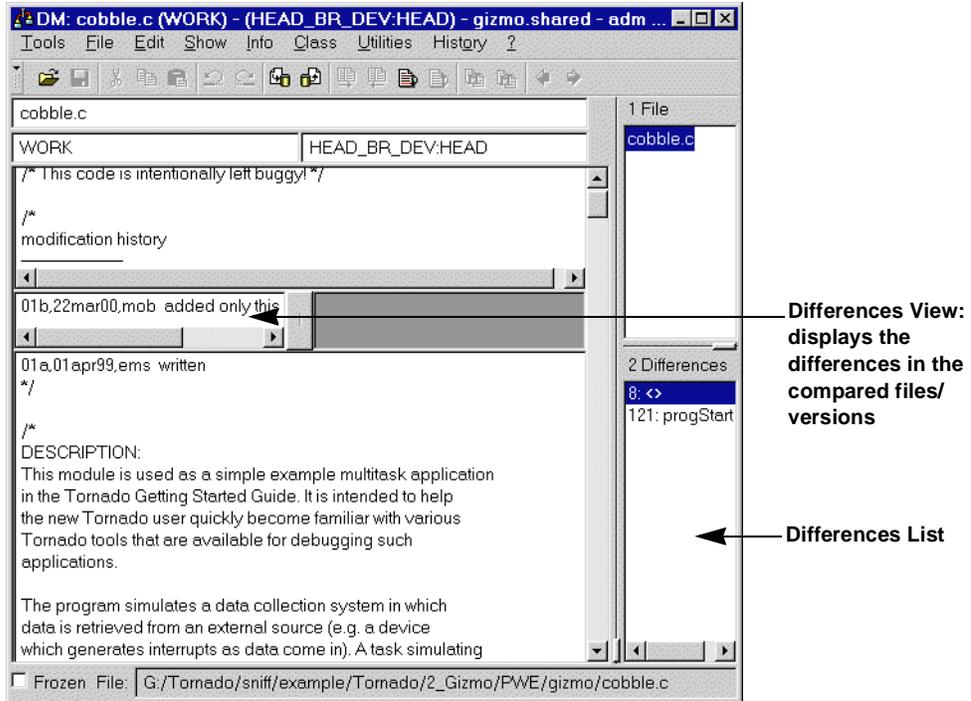
You will notice that one file is shown in the File List, this means that this is the only file that has been modified.

3. Highlight the `cobble.c` file, right-click and choose **Show Differences...**
4. In the Differences between dialog that appears, accept the defaults and press **Ok**.

The Diff/Merge tool opens and is positioned at the first difference.

The Diff/Merge tool

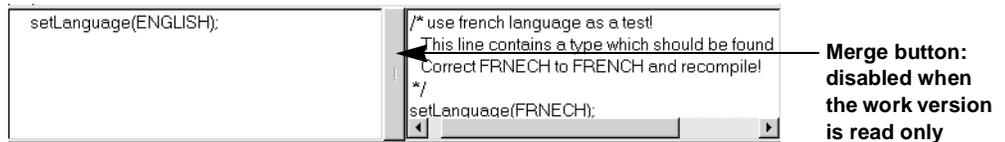
The Diff/Merge tool allows you to show and merge differences between files and versions of files. The Diff/Merge tool offers two- or three-way differences. Three-way differences are important for investigating changes made by two developers to the same file. In such a case you want to look at the two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



Now lets look at the second difference.

- Select `121:progStart (f)` in the Differences List.

121 is the line number and `progStart` is the name of the function that contains the difference. The Differences view on your screen should look like the following:



The Differences View displays the differences in the compared versions. The areas of code containing differences are split into two views and are separated by a **Merge** button. The **Merge** button between the working file and the compared version allows you to merge differences into the working file. If the working file is not writable, or if two versions of the same file are being compared (as in our case), the **Merge** button is disabled.

Notice that the language specified in the WORK version is English whereas the language specified in the HEAD_BR_DEV:HEAD version is French. Here you can see that French is incorrectly spelled, this has been done intentionally to test if the compiler reports this error. The differences show that your workspace is not up-to-date so we will now synchronize your workspace in the Project Editor.

Synchronizing your Workspace

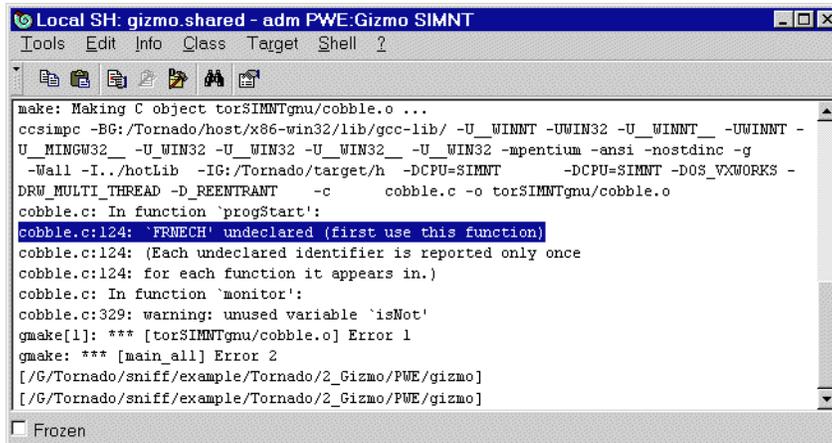
1. Make sure that all projects are checkmarked in the Project Tree.
2. From the File Status drop-down, choose **All Files**.
3. Choose **Project > Synchronize Checkmarked Projects...**
4. In the Files Compared to dialog that appears, press **Ok**.
Your workspace is now up-to-date.
5. To verify this, choose **Modified** from the File Status drop-down.
No files are shown in the File List.
6. From the File Status drop-down, choose **All Files** once more.

Building the Executable

1. Choose **Target > Update Makefiles...** to generate the Make Support Files for all projects.
2. In the dialog that appears, press **Yes**.
3. In the Project Tree, highlight `gizmo.shared`.

- Choose **Target > Make > all** to recursively build both the library (which is a subproject) and the main application.

A Shell tool appears, in which the `make all` command is executed in each project directory.



```

make: Making C object torSIMMTgnu/cobble.o ...
ccsimpc -BG:/Tornado/host/x86-win32/lib/gcc-lib/ -U_WINNT -UWIN32 -U_WINNT_ -UWINNT -
U_MINGW32_ -U_WIN32 -U_WIN32_ -U_WIN32_ -U_WIN32_ -mpentium -ansi -nostdinc -g
-Wall -I../hotLib -IG:/Tornado/target/h -DCPU=SIMNT -DCPU=SIMNT -DOS_VXWORKS -
DRW_MULTI_THREAD -D_REENTRANT -c cobble.c -o torSIMMTgnu/cobble.o
cobble.c: In function 'progStart':
cobble.c:124: 'FRNECH' undeclared (first use this function)
cobble.c:124: (Each undeclared identifier is reported only once
cobble.c:124: for each function it appears in.)
cobble.c: In function 'monitor':
cobble.c:329: warning: unused variable 'isNot'
gmake[1]: *** [torSIMMTgnu/cobble.o] Error 1
gmake: *** [main_all] Error 2
[/G/Tornado/sniff/example/Tornado/2_Gizmo/PWE/gizmo]
[/G/Tornado/sniff/example/Tornado/2_Gizmo/PWE/gizmo]

```

As you can see, errors are reported in the Shell tool. Lets now correct the error in line 124 of the `cobble.c` file.

- Highlight the line shown in the above illustration, right-click and choose **Show Error...**. The Source Editor opens and is positioned at the error.
- In the Source Editor, choose **File > Check Out...** to make the file writable.
- In the dialog that appears, press the **Exclusive Lock** button.
- In the highlighted line in the Source Editor, change `FRNECH` to `FRENCH` and save the file.
- Choose **Target > Make > all** to compile the project.

Notice that no errors are reported in the Shell tool so compilation has been successful.

Checking in your changes

Once compilation is successful, check in your changes into the Repository. To do so:

- In the Project Editor's File Status drop-down, choose **Writable**.



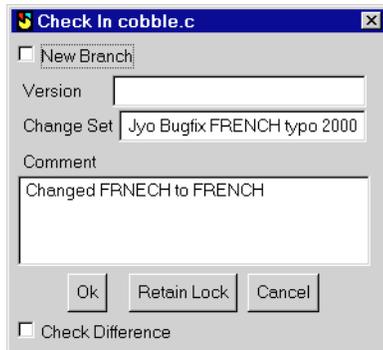
- In the Project Tree, right-click and choose **Select From All Projects**.

3. You should only see the `cobble.c` file, select it and choose **File > Check In...**
The Check In dialog appears.

In the 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*. Moreover, change sets are nicely displayed in Lists making it easier to track your changes. Note that change set names must be unique in your entire project's history; therefore, we recommend that you add your name and the current date to your changeset specifier.

1. Clear the **New Branch** checkbox.
2. Leave the **Version** field blank. SNIFF+ will automatically assign a version number (1.1) and later increment it automatically.
3. In the **Change Set** field, enter a name for the change set, e.g.,
<Your_Name> Bugfix FRENCH typo <current_date>
4. In the **Comment** field, enter a descriptive text, e.g, Changed FRNECH to FRENCH.



5. Press **Ok**.

The Configuration Manager gives a structural and file-based overview of the changes between two configurations of a software system. Configurations are selected file versions grouped together under the same symbolic name.

In this chapter you will:

- look at the configurations in `gizmo.shared`
- compare two configurations

Opening the Configuration Manager

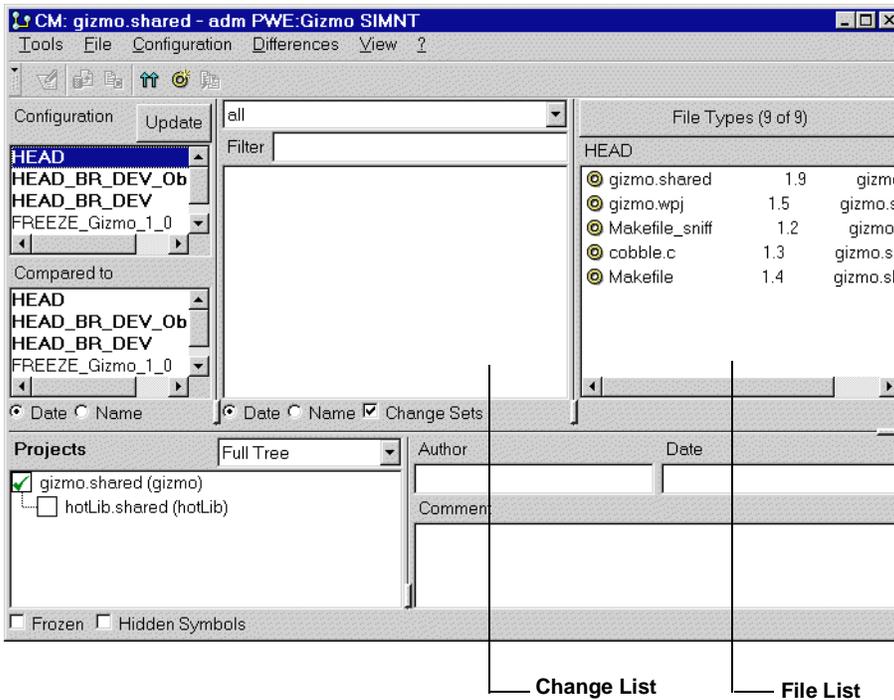
1. In the Project Editor, choose **Tools > Configuration Manager**.
The Configuration Manager opens.
2. Close the Project Editor.

Looking at configurations

You may want to look at the configurations of `gizmo.shared`. To do so:

1. Highlight `gizmo.shared` in the Project Tree of the Configuration Manager.
2. Choose **Context menu > Select from gizmo.shared Only**.
The configuration information for `gizmo.shared` is shown in the Configuration List.
3. Highlight `HEAD` in the Configuration List.

HEAD is the symbolic name of the latest version of all files in a particular configuration. The HEAD configuration thus reflects the current state of your software system. In the illustration below, you can see all files that are part of HEAD.



Comparing two configurations

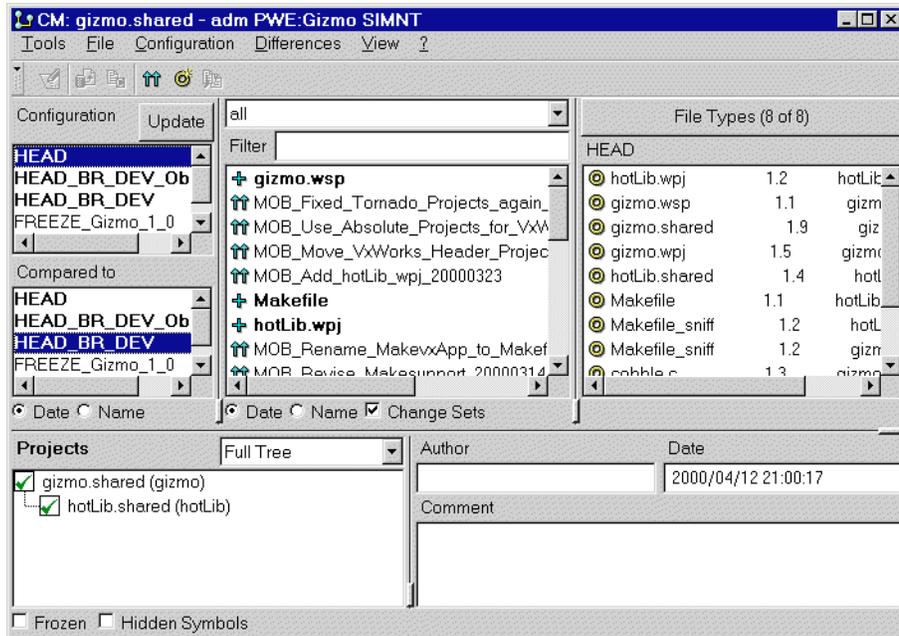
Let's view the changes between the latest configuration (HEAD) of `gizmo.shared` with its branch configuration `HEAD_BR_DEV`:

1. Make sure that HEAD is highlighted in the Configuration List.

- In the Compared to List, highlight HEAD_BR_DEV.

As you can see, the differences between the two configurations are displayed in the Change List. Icons in the Change List indicate the nature of the difference.

To see a description of the icons, choose **Help(?) > QuickRef**.



You may now want to look more closely at a change set in the Change List. To do so:

- Make sure that the **Change Sets** checkbox is selected.

When selected, change sets are displayed and the changes to the individual files that are part of the change sets are not displayed.
- Highlight MOB_Rename_MakevxApp_to_Makefile_sniff_20000323 in the Change List.

The files of MOB_Rename_MakevxApp_to_Makefile_sniff_20000323 are now displayed in the File List.
- From the drop-down above the filter field, select **different branches**.
- In the Change List, you should see the following:

 cobble.c 1.3 <-> 1.2.1.1

Here you can see that there are different versions of the `cobble.c` file in the HEAD configuration and in the branch configuration HEAD_BR_DEV.

What's Next

- Close the `gizmo.shared` project by selecting it in the Launch Pad and choosing **Project > Close Project**.

The next part of this guide introduces you to SNIFF+'s team support. You will learn how to create Working Environments and shared projects.

Part IV

Team Support



Key Concepts

This chapter introduces 2 key concepts

- *shared projects*
- *working environments*

The quick overview below will help you to understand these concepts and later apply them for your own Team Support Setup. For detailed information going beyond this brief introduction, please refer to the *SNiFF+ 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.

The root directory mentioned is defined by the SNiFF+ Working Environment. Therefore, a SNiFF+ shared project can only exist inside a Working Environment.

Working environments

SNiFF+ Working Environments serve several purposes, to mention a few:

1. They define the root directory for your projects.
2. They allow you to easily find and open projects, without having to search your file system.
3. They define the target platform and build specification that you would like to use for compiling.
4. They provide managed workspaces, where you can automatically work on a given Version Control Configuration. You can also use scripts to automatically synchronize your workspace with the given configuration.
5. They allow you to define areas of shared source code, that will be automatically visible to every developer.

Depending on the version control system that you are using, some of these issues may already be handled by the version control tool. With ClearCase, for instance, the sharing of sources is already done by the CM tool, therefore you will only need SNiFF+ Private Working Environments (PWEs) in this case. For most other CM tools, SNiFF+ provides 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.

Depending on the CMVC tool you are using, there are different ways to make the Repository location known to your tool. Some tools (like, for instance, ClearCase) know implicitly where the Repository is located; for most other tools, like the RCS system we are using in this guide, you define the Repository location by defining a *Repository Working Environment (RWE)* and setting the RWE root directory

The SSWE (Shared Source Working Environment)

SNiFF+ allows you to define a location on your file system, where you may store a copy of your source code that can be easily accessed by every team member. This shared workspace is called the *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.

Note

On Unix: File sharing is done by symbolic links from the team member's WE to the SSWE.

On Windows: Since symbolic links are not supported on Windows local copies are made.

You may use an SSWE without Version Control. For instance, you may define an SSWE for the `target` directory of your Tornado Installation (the `WIND_TARGET_BASE`). This will allow you to get read-only access to a clean copy of original Tornado Sources from every working environment that accesses the SSWE.

For full Configuration Management Support, you may add another SSWE, in which you regularly synchronize the files and directories with the RWE to reflect a well-defined state of your team's software system. For instance, many users define one SSWE to be automatically updated to the latest version every night, while you may have another SSWE to contain the latest stable version, which is only synchronized at well-defined milestones.

If you plan to use an SSWE with full Configuration Management and automatic synchronization, it is strongly recommended that one person be appointed to administer this "maintenance system". In SNiFF+ this person is called the *Working Environments Administrator*. For more information about Working Environments and the automatic synchronization mechanism, see the SNiFF+ User's Guide.

The SOWE (Shared Object Working Environment)

Just like with shared source code, SOWEs serve as shared repositories for your team's most current and stable object code.

An essential aspect of SOWEs is avoiding unnecessary builds in Private Working Environments (see below) that access them. Because this object re-use facility makes extensive use of symbolic links, it only works well on Unix systems, and it is only advisable for very large projects. Therefore, we will not cover SOWEs in this guide.

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.

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.

For your daily development, you will always work in PWEs only. You should open a project in an SSWE or SOWE only for synchronization, but never for any modifications. Therefore, the SNiFF+ Launch Pad only shows your PWEs while only the Working Environments Tool allows you to modify the entire working environment configuration. The next chapter explains how to use the Working Environments Tool.

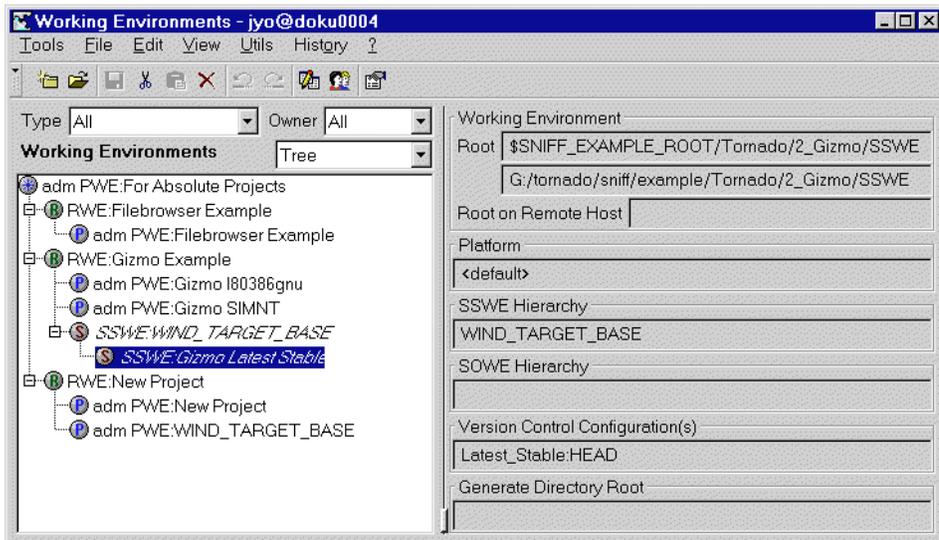
Working with an Existing Team Project

This chapter will describe how to use SNIFF+ Team Support when working with an existing SNIFF+ project. You will first add a Private Working Environment (PWE) to the Working Environment containing shared source files (SSWE) and then specify a platform, on which you want to build your projects. For a description of shared projects and Working Environments, please refer to [Key Concepts — page 57](#).

The Working Environments tool

The Working Environments tool is used for creating and maintaining Working Environments (WEs), for opening projects and for modifying Working Environments to suit your needs, for example you can specify a platform on which you want to build, you can specify your build preferences for compiling and debugging, etc.

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



Understanding the Working Environment Configuration

Our example configuration contains three different Repositories, which represent three completely different projects:

- the Filebrowser Example, which is used in the SNIFF+ C++ Tutorial

- the Gizmo Example, which is our main Project in this guide
- and an empty Repository for a New Project, that we will use in [Setting up a Multi-User Project — page 65](#).

Directly below the Gizmo Example Repository, there are two PWEs which do not access an SSWE. They both refer to the same root directory, but they are set up to use different Platforms (Build Specifications) for compiling. You have been working in Simulator PWE so far.

The `SSWE:WIND_TARGET_BASE` is set up on the target directory of your Tornado Installation, so that you can access original Tornado Source code from any Working Environment below. Finally, the `SSWE:Gizmo Latest Stable` contains a snapshot of an extended version of the Gizmo Example. In this SSWE, we have prepared a BSP and a Customized VxWorks for Gizmo. By creating a new PWE below this “snapshot” SSWE, you will get access to these projects.

Adding a new Working Environment

In order to get access to the Team Project that we have set up for you, you have to create a new PWE which accesses the SSWE. To create your new PWE:

1. Highlight **SSWE:Gizmo Latest Stable**.

This is the Working Environment containing the team's source files.

2. Choose **Edit > New Private based on SSWE**.

The Working Environment - New Private based on SSWE dialog opens.

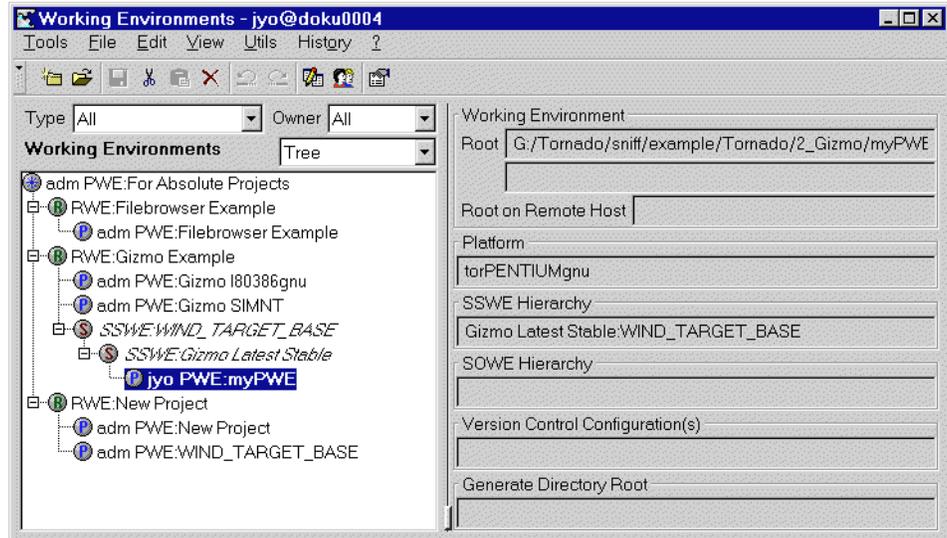
In the Working Environment - New Private based on SSWE dialog

1. In the **Working Environment** field, enter the name of the working environment, for example `myPWE`.
2. In the **Root** field, use the **Directory** button to navigate to `G:/Tornado/sniff/example/Tornado/2_Gizmo/` and add `myPWE`. The path is now `G:/Tornado/sniff/example/Tornado/2_Gizmo/myPWE`
3. In the **Platform** field, use the drop-down to select a platform on which you want to build projects in this Working Environment. Select your favorite Tornado Target platform, for instance, **torPENTIUMgnu**.
4. Delete `Latest_Stable:HEAD` from the **Version Control Configuration** field. This is necessary in order to allow checking in new versions from your PWE (you will check in to the latest version, `HEAD`, instead of the frozen `Latest_Stable`).
5. Notice that your User name is entered in the **Owner** field. Being the owner of the Working Environment means that you are the only one who is allowed to modify its attributes.

The Working Environment is now created.

- In the Working Environments tool, choose **File > Save**.

The Working Environments tool should now look similar to the illustration below.



In the same way, each team member can create a Private Working Environment for himself/herself to ensure that the entire team is working with the same project structure and to ensure that changes are made locally in each team members Private Working Environment, projects are compiled locally to test changes and then checked in to the Shared Source Working Environment to maintain a 'clean' code base.

Lets now open the shared project called `Workspace . shared` in this newly created Working Environment.

Opening the shared project in your Working Environment

- In the Working Environments tool, double-click on `<owner>PWE:myPWE`, where `<owner>` is your User name.

The Open Project dialog appears.

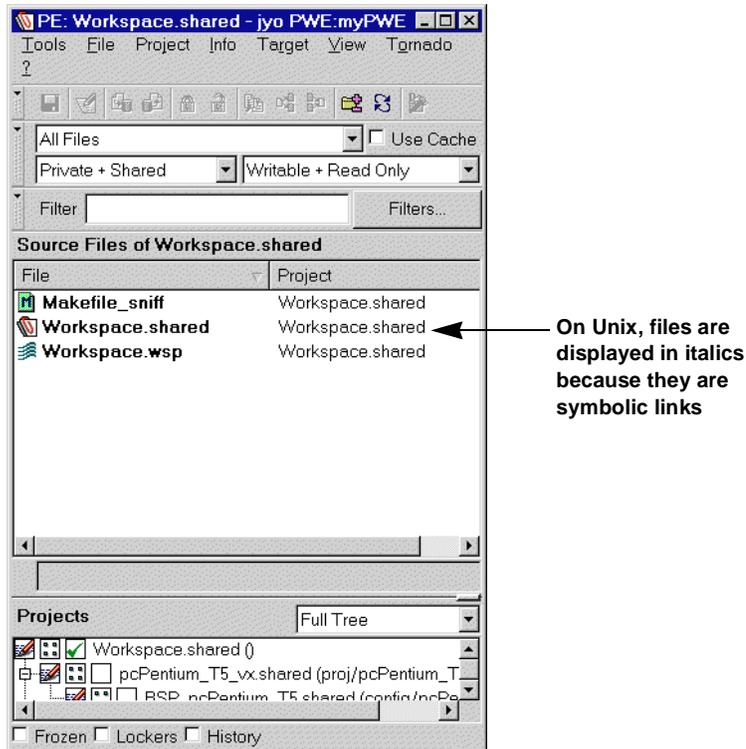
- In the Open Project dialog that appears, press the **Update** button.

The files in the Shared Source Working Environment (SSWE) are displayed. Notice that the files are shown in italics to represent that these files are in the SSWE instead of in your PWE.

- Scroll to the bottom of the list and double-click on `Workspace . shared`.

4. In the dialog that appears, select the **Repeat** checkbox and press **Create Directory**.

SNiFF+ now recreates the directory structure of the SSWE in your PWE. On Unix symbolic links are created to the source files whereas on Windows local copies are made. After the project structure has been recreated, SNiFF+ parses all the symbol files in the project and loads the project with symbol information. After the project is opened, SNiFF+ displays the project's structure and contents in a Project Editor, which should look like the one illustrated below.



You have now opened your own copy of the shared project `Workspace.shared` in your Private Working Environment and can now go ahead and compile this project but just remember to Update Makefiles before doing so.

Setting up a Multi-User Project

This chapter describes how to set up a multi-user project from scratch, including how to set the Make Attributes for your project as well as how to create Working Environments for the team.

This chapter is about

- using a predefined template for setting up a SNIFF+ shared project for development.
- setting up Build Support for a newly created project.
- creating a Shared Source Working Environment for source files and projects.

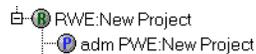
We have created Tornado-specific SNIFF+ Project Templates to make it easier to set up your projects. We explain the steps to be taken in terms of the `gizmo` and `hotLib C++ examples`, which are also part of your SNIFF+ Tornado integration (in the `SNIFF_DIR/example/Tornado/3_NewProject/PWE` directory).

Multi-User Project Setup

We will now set up a multi-user project in an existing Working Environment. We have already set up a repository `RWE:New Project` and a private Working Environment `adm PWE:New Project`, see the illustration of the Working Environments tool below.

1. In the Launch Pad, select **Tools > Working Environments**.

The Working Environments tool opens. We have already set up a repository `RWE:New Project` and a private Working Environment `adm PWE:New Project`, see below.



2. In the Working Environments tool, select `adm PWE:New Project`.

3. Choose **File > New Project... > with Template...**

The Project Template dialog appears.

4. In the Available Template Files list, select the `Tornado_Application.ptmpl` template.
5. Press the **Change Directory...** button.
6. In the Directory dialog that appears, navigate to `SNIFF_DIR/example/Tornado/3_NewProject/PWE` and press **Select**.
7. Press **Ok**.

The Attributes of New Project dialog appears. For now, there are only two settings that are important at this point:

- In the General node, there is the **Ignore Directories** field. It allows you to specify (with GNU Regular Expressions) any directories that SNIFF+ should not take into account

when creating its project descriptions.

- In the File Types node, you may add or remove any file types that SNIFF+ should add to its project descriptions.

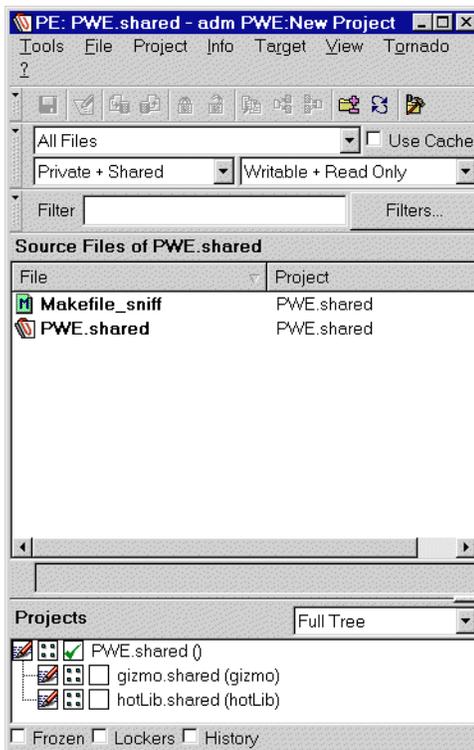
For more information, please refer to the SNIFF+ Reference Guide.

8. For now leave all settings as they are and press **OK**. In the next section, we will set the project attributes using the Group Project Attributes dialog.

SNIFF+ will now set up the project and parse your files.

9. In the dialog that appears asking if you want to generate cross reference information, press **No** since SNIFF+ will automatically generate cross reference information as soon as it is needed.

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



Setting Up Make Support

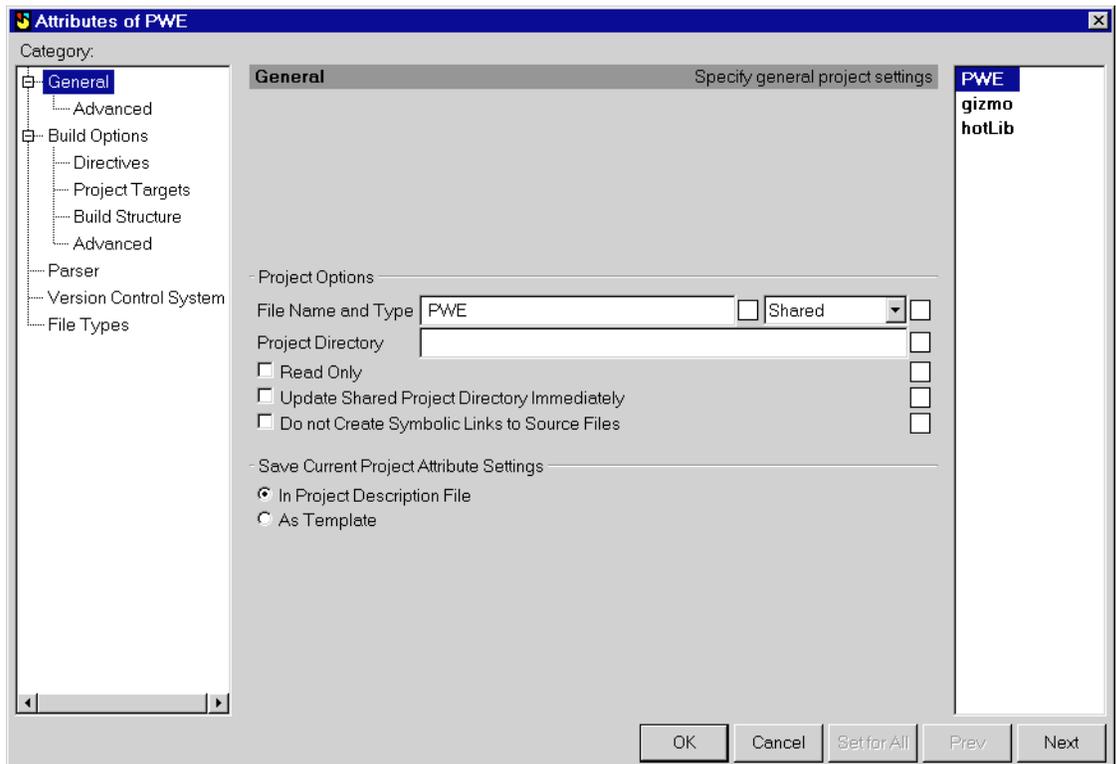
We will now set up the necessary Make attributes for the example project. For a description of the other project attributes, please refer to the SNIFF+ User's Guide.

In the Project Editor

1. In the Project Tree, highlight `PWE.shared`, right-click and choose **Select from All Projects**.
2. Choose **Projects > Attributes of Checkmarked Projects...**

The Group Project Attributes dialog opens. The Group Project Attributes dialog is an extended Project Attributes dialog that lets you set the project attributes of multiple projects listed in the Project Editor's Project Tree.

In the Group Project Attributes dialog



Setting up Make Support for hotLib.shared

1. Highlight **hotLib** in the Project List (the list on the right of the Group Project Attributes dialog).
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Library** field of the of the Ansi C/C++ tab, enter `hotLib.a`. This will be the name of the library built in this project.
4. Under the **Build Options** node, select **Build Structure**.

5. In the Build Structure view, choose **Passed to Superproject** drop-down > **Library**.
The project's library is exported to `PWE.shared` and is used to build the myApp executable.

Setting up Make Support for gizmo.shared

1. Highlight **gizmo** in the Project List.
2. Under the **Build Options** node, select **Project Targets**.
3. In the **Executable** field of the Ansi C/C++ tab, enter `gizmo.out`. This will be the name of the project's executable.
4. Under the **Build Options** node, select **Build Structure**.
5. In the Build Structure view, press the **Generate** button next to the **Recursive Make Dir(s)** field.

The executable is built using recursive Make rules. By pressing the **Generate** button, SNiFF+ generates the order of subprojects in which Make is executed.

Generating the include paths for all projects

1. Under the **Build Options** node, select **Directives**.
2. Select the checkbox to the right of the **Generate** button.
3. Press the **Set for All** button to generate the include paths for all projects in the Project List.
4. Press **OK** to apply the changes to the project attributes.

The icons in the Project Tree of the Project Editor warn you that the projects have been modified.

5. A dialog appears asking you to update Makefiles. We will do this later so press **No**.

In the Launch Pad

To save the changes made to `PWE.shared` and its subprojects:

1. Select `PWE.shared` in the Project List.
2. Choose **Project > Save Project PWE.shared**.
3. In the Alert dialog that appears, press the **Save All** button.

Building the Executable

- Once you've set the Make attributes, compile `gizmo.shared` to see if the attributes have been set correctly. To do so refer to [Building the executable — page 37](#).

Checking all files into the Repository

Once compilation is successful, check in all files into the Repository. To do so:

1. In the Project List, right-click and choose **Select All**.
2. Choose **File > Check In...**
3. In the dialog that appears asking you if you want to create the Repository directory, select **Repeat** and press **Yes**.
4. In the Check In dialog that appears, press **Ok** to start the check in.

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.

Creating a Shared Source Working Environment

Lets now import your project from the PWE into a new SSWE, so that other team members can see your projects too.

Note

Team members should **NEVER** directly work in the SSWE and these files should always be **READ-ONLY** to maintain a 'clean' code base.

In the Working Environments tool

1. Select `RWE:New Project`.
2. Choose **Edit > New Shared Source based on Repository**.
3. In the Working Environment - New Shared Source based on Repository dialog that appears, enter a name in the **Working Environment** field - for example `New Project`.
4. Press the **Directory...** button next to the **Root** field.
5. In the Directory Name dialog that appears, click on the New Folder icon and enter **Sswe** as its name.
6. Double-click on **Sswe** and press **Select**.
7. Press **Ok**.

The SSWE is now created and added to `RWE:New Project`.

We have just created a SSWE and now we will copy over the shared project and then check out the shared files into it

Copying the shared project into the SSWE

In your OS Shell (or the Windows Explorer)

- Copy `PWE . shared` from
`SNIFF_DIR/example/Tornado/3_NewProject/PWE`
to
`SNIFF_DIR/example/Tornado/3_NewProject/SSWE .`

In the Working Environments tool

1. Double-click on `SSWE:New Project`.
2. In the Open Project dialog that appears, press the **Update List** button.
3. To open the root project and all its subprojects, double-click on `PWE . shared`.
4. In the dialog that appears, press **Yes**.

SNiFF+ informs you that it cannot find the directories of the shared projects in the SSWE root directory (they haven't been created yet). You will now have SNiFF+ initialize your SSWE by copying the PWE project directory structure into the SSWE.
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 SSWE, the project is automatically opened in it and displayed in the Project Editor.
6. Select **Project > Close Project**.

Changing Working Environment Hierarchy Structure

The SSWE now contains the shared project as well as the source files. This code should never be directly modified in the SSWE. If team members make modifications to files, these files should be checked out and changes should be made in their PWEs. In order for this to be possible, the PWEs should access the SSWE, i.e., they should lie under the SSWE in the Working Environment hierarchy.

You must have noticed that `adm PWE:New Project` lies directly under the `RWE:New Project` and not under `SSWE:New Project`. If we want this PWE to access the shared code base in the SSWE, we need to change the hierarchy structure. To do so:

- In the Working Environment hierarchy, drag and drop `adm PWE:New Project` into `SSWE:New Project`. the hierarchy structure should now look like the following:



- Click **File > Save** to save the modifications made to the Working Environment structure.

Synchronizing Working Environments

Lets now synchronize the PWE to get it up-to-date.

1. Double-click on `adm PWE: New Project`, select `PWE.shared` and press **Open** to open your project in the PWE.
2. Make sure that all projects in the Project Tree are checkmarked and choose **Project > Synchronize Checkmarked Projects...**

The Files Compared To dialog appears. All files in the PWE will be updated to the version that appears in the dialog's **Version** field (HEAD by default).

3. Press **Ok**.

A Log tool appears, and SNIFF+ updates all the files in the PWE.

4. If the project structure has changed, a dialog appears asking you to reload the project structure. If so, press **Yes**.

SNIFF+ can perform the synchronization procedure automatically driven by scripts. For more information on synchronizing Working Environments, please refer to the SNIFF+ User's Guide or refer to the automatic update scripts in your `SNIFF_DIR/ws_support` directory.

What's Next

The next part of this guide describes advanced issues such as creating BSPs and bootable applications, WindRiver Source products as well as customizing issues.

Part V

Advanced Issues



Working with BSPs

SNiFF+ can help you to create or configure your Board Support Packages (BSPs) for Tornado. Depending on your team size and configuration management setup, the environment can be set up at different levels of sophistication. We will explain everything in terms of a Pentium BSP here, but the steps can be followed in the same manner for any other BSP you may use.

This chapter is about:

- how to set up BSPs for Tornado-only Make Support.
- version controlling a Released BSP in SNiFF+.
- enabling SNiFF+ Make Support for a BSP.

BSP Setup for Tornado-Only Makesupport

This very simple setup will suffice for most applications, and because only original Tornado Makesupport is used it is more than likely that your BSP will compile correctly. To create a modified version of the Pentium BSP under SNiFF+ control, we will do the following:

- First, we will copy the original `WIND_BASE/target/config/pcPentium` directory into a new directory, for instance `WIND_BASE/target/config/pcPentium_T5`. `WIND_BASE` is your Tornado installation directory. This is necessary because Tornado Makesupport will write into the `WIND_BASE` tree during the `Make Release` process, and we do not want to overwrite original Tornado files.

Note

If you do not have write access to the `WIND_BASE` tree, you may also copy the BSP into your PWE (for instance `$HOME/Sniff_WS/PWE/config/pcPentium_T5`) and work there.

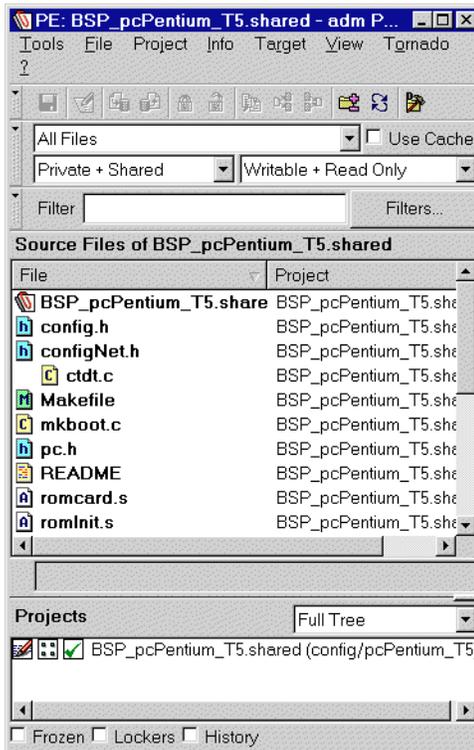
Setting up a SNiFF+ Project onto a BSP

Next, we will set up a SNiFF+ Project onto the newly copied BSP.

1. In the Working Environments tab of the Launch Pad, select
`adm PWE:WIND_TARGET_BASE`
If you copied the BSP to a different place, select this as your PWE.
2. Choose **Project > New Project... > With Template...**
3. In the dialog that appears, select the `Tornado_BSP.ptmpl` Template

4. Press the **Change Directory...** button next to the Project Directory field and navigate to the `WIND_BASE/target/config/pcPentium_T5` directory.
5. Press the **OK** button
6. In the Project Attributes dialog that appears, you may want to change the project description file's name from `pcPentium_T5.shared` to `BSP_pcPentium_T5.shared` or `config_pcPentium_T5.shared` to clearly document that this is a BSP (we will refer to `BSP_pcPentium_T5.shared` in the remainder of this document). Press the **OK** button again (all settings from the Template should be correct).

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



7. In the Project Editor, double click on the Makefile and change all occurrences of `pcPentium` to `pcPentium_T5` to make sure Tornado will create correct output files. Save the file.
8. Choose **Target > Make > bootrom** to verify that the original BSP compiles correctly.
9. In the Project Editor, choose **Project > Add Subproject** to add any SNIFF+ Projects you would like to see for browsing. If you just need to do some simple customizing, you probably don't even need to do this; for advanced customizing or writing additional drivers, you may want to add some or all of the following project description files:
 - `config/VxWorks5.4_config.shared` to see user-configurable hooks for

old-style BSPs

- `src/VxWorks5.4_src_comps.shared` to see user-configurable hooks for new-style BSPs
- `src/VxWorks5.4_src_drv.shared` to see sources for hardware drivers
- `h/VxWorks5.4_h_drv.shared` to see headers for hardware drivers
- `h/VxWorks5.4_h_SYSTEM.shared` to see the VxWorks system headers.
- `h/make/VxWorks5.4_h_make.shared` to see the VxWorks General Make-files.

Note

You may need a full SNIFF+ product license or a time-limited full evaluation license to be able to add these projects, because some of them may contain more than 200 files. See [Licensing — page 14](#) for information on how to get an evaluation or product license.

10. Check in all files of the BSP project, so that you can track the modifications that you make later.
11. When you are done with any modifications you need to do, choose **Target > Make > release** to build the entire BSP and create a Tornado Template project in `$WIND_BASE/target/proj/pcPentium_T5_vx`.

From that point on, your BSP is released and other users can build their bootable applications based on it. In a multi-user environment on Windows, where every developer has his private copy of Tornado, the released BSP needs to be copied into the other Tornado Installations. This can be simplified by putting the released Tornado template project under SNIFF+ Control as well.

Version Controlling a Released BSP in SNIFF+

Having your released BSP and template projects under SNIFF+ Version Control allows every developer to quickly get new versions by simply synchronizing his workspace. In a multi-team environment with two SSWEs, this can even be done automatically at night with the update scripts. All you need to do is create a SNIFF+ "Master Project" for your Tornado installation, which will contain all your BSPs and template projects as subprojects. Again, we will explain everything in terms of our `pcPentium_T5_vx` project, but the instructions apply equally to other BSPs.

- First, we create a `Workspace.shared` project which will be the common root of all released BSPs. If you already have a such a Workspace Project, you may skip this step, open your Workspace Project and continue with step 8 below. If not, this is what you should do

Creating the root project

1. In Working Environments tab of the Launch Pad, select the adm PWE: WIND_TARGET_BASE working environment.
2. If you have no write access to the \$WIND_BASE tree, first copy the \$WIND_BASE/target/proj/pcPentium_T5_vx directory into your PWE (for instance \$HOME/Sniff_WS/PWE/proj/pcPentium_T5_vx) and select the PWE instead.
3. Select **Project > New Project... > With Template...**
4. Select the **Wrapper.ptmpl** Project Template.
5. In the **Project Directory** field, enter a dot (.) to tell SNIFF+ that we would like to set up the project in the root of the Working Environment and press **Ok**.
6. In the Project Attributes dialog that appears, enter a name for your Workspace Project (for instance `Workspace.shared`) and press the **OK** button.

Adding Subprojects to the Root Project

1. In the Project Editor that opens, select your `Workspace.shared` project. If it is read only, check it out.
2. Select **Project > Add Subproject to Workspace.shared...** and add your BSP project `config/pcPentium_T5/BSP_pcPentium_T5.shared`.
3. Select **Project > Add New Subproject... > With Template...**
4. Choose the `Tornado_Customized_VxWorks.ptmpl` template.
5. Press the **Change Directory...** button and navigate to the `$WIND_BASE/target/proj/pcPentium_T5_vx` directory and press **Ok**.
6. In the Project Attributes Dialog, click **OK** to accept all defaults.
7. Save the root project and check-in all files from the root project as well as all subprojects.

Now, any user who opens the `Workspace.shared` project and synchronizes his workspace, will automatically get the latest version of the released BSP. For full SNIFF+ Workspace Support without absolute directory references, it may be necessary to replace absolute paths created by Tornado in the `pcPentium_T5_vx.wpj` project with relative paths. For more information, see [Working with Bootable Applications — page 81](#).

For removing a subproject from your SNIFF+ project, select a subproject in the Project Tree and choose **Project > Remove Subproject**.

Similarly, you can also add/remove files to/from a project. To do so, select the project in the Project Tree and choose **Project > Add New File to *project*** or **Project > Add/Remove Files to/from *project***.

Enabling SNIFF+ Makesupport for a BSP

One disadvantage of using the Tornado Makesupport for your BSP is, that you cannot use SNIFF+ Recursive Make for your entire workspace, to build a complete product. The other disadvantage is that you cannot switch Build Specifications (for instance, Release and Debug Build) for your BSP and that object files are not redirected into a Platform-specific

subdirectory. All these drawbacks can be amended easily by enabling SNIFF+ Makesupport for your BSP. Again, we show the steps necessary in terms of the `pcPentium_T5`, but they can be done in just the same way for any other BSP.

1. Open your project `BSP_pcPentium_T5.shared` in the Working Environment where you have created it.
2. The Project Editor appears. Select `BSP_pcPentium_T5.shared` in the Project Editor's Project Tree.
3. If the project is read-only, check it out.
4. Choose **Project > Attributes of BSP_pcPentium_T5.shared...**
5. In the Build Options view, enable the **Use SNIFF+ Makesupport** checkbox and clear the **Make Command** field. Having this field empty will allow you to use the general make command from your Platform Description. Press the **OK** button.
6. SNIFF+ will ask you to copy a Makefile from Template, since SNIFF+ Makesupport has been enabled. Press **Yes**, and you will get a new file called `Makefile_sniff`.
7. In the dialog that appears prompting you to update Makefiles, press **Yes**.
8. Choose **Target > Make > bootrom** to verify that you can correctly build your BSP.

Working with Bootable Applications

Bootable Application Projects, also known as “Customized VxWorks”, are very tightly coupled to the Tornado Environment. In fact, you customize your VxWorks Kernel in the Tornado GUI, and the GUI dumps a Makefile plus some C Source files to the disk, which describe your configuration.

In general, you will continue to use your Tornado IDE to configure your bootable VxWorks Kernels. But SNIFF+ can help you to keep this configuration together with the BSP and the application code in a single managed workspace; it gives you more flexibility to move your project by getting rid of absolute path references; it allows for full Version Control, and it allows you to build your entire application in a single step.

This chapter is about:

- Setting up a SNIFF+ Project for a Bootable Application
- Modifying the Project to get rid of absolute path references.

Setting up a SNIFF+ Project for a Customized VxWorks Project

We will describe the setup in terms of the `pcPentium_T5_vx` project that we created from the `pcPentium_T5` BSP in the previous chapter. All descriptions equally apply to any other VxWorks project. If you want to set up a SNIFF+ project onto an existing Customized VxWorks project, just copy the corresponding directory into your PWE instead of making a new setup in Tornado.

To create your Customized VxWorks Project:

1. Launch the Tornado IDE. You can do this from the SNIFF+ Launch Pad or Project Editor by selecting **Tornado > Launch Tornado IDE**.
2. In Tornado, Choose **New Project > Bootable Application (Customized VxWorks)** and select a directory inside your PWE as the target directory. In our example, we choose to create a new project based on the `pcPentium_T5_vx` project, call it `pcPentium_T5_vx`, and place it into `$HOME/Sniff_WS/example/Tornado/3_NewProject/myPWE/proj/pcPentium_T5_vx`.
Tornado will create the project and show its Project Facility. You may now customize any options you like. Build your project once to make sure that the Makefile is dumped, then close the Project in Tornado.
3. In the SNIFF+ Launch Pad, select the Working Environments Tab and choose your PWE. Select **Project > New Project > With Template**.
4. In the New Project with Template Dialog, select **Tornado_Bootable_VxWorks.ptmpl**. Press the **Change Directory** button and navigate to the directory where you created the Tornado Project. Back in the dialog, press **OK**.

5. In the Project Attributes for New Project Dialog, leave all settings as they are and press **OK**.

SNiFF+ opens the Project Editor.

6. In the Project Editor, select all files and choose **File > Check In**.
7. In the Check In Dialog, press the **Retain Lock** button to check the files in but keep a writable copy in your PWE.

It is important to know, that Tornado will re-generate and modify all files of your “Customized VxWorks” Project whenever you make any modification to the Kernel configuration. Along with this re-generation, Tornado will always produce absolute paths in your project, which are not suitable for Teamwork.

Therefore, we will now manually modify the Tornado Project to get rid of all absolute path references. Since this is an operation that can easily destroy your project, it was important to check in all original files, so that we are sure we can always revert to a good version. Moreover, it will be instructive for you to compare subsequent versions with the original ones created by Tornado.

Note that later, when you plan to modify the project in Tornado again, all files must be writable or Tornado will issue an error message and refuse to save your modifications.

Removing Absolute Path References

In a regular Tornado Project, there may be three types of absolute path references:

- References into your Tornado installation (`$WIND_BASE`)
- References to an external BSP from which you created the Project
- References to external libraries that should be linked.

Each of these absolute references limits re-using your project in other environments. To remove these absolute references:

1. In the Project Editor’s File List, select `pcPentium_T5_vx.wpj`, then right-click and choose **Edit File**.

2. Choose **Edit > Find** to search for absolute references.

On Windows, you may want to search for the colon (“:”) that is found in all pathnames with a drive letter.

On Unix, you may want to search for a prefix like `/opt` or `/Users`.

3. When you find an absolute reference, we will use the SNiFF+ Retriever to replace it with a reference relative to `$(PRJ_DIR)`. The `PRJ_DIR` is the only variable that can be used in Tornado to create references relative to the location of your project. For example, let us assume that

- Your PWE is located at `C:/Projects`
- Your Customized VxWorks is located at `C:/Projects/myVxWorks`
- Your BSP is located at `C:/Projects/myBSP` and has been released to `$(WIND_BASE)/target/config/myBSP`

- An external library is located in `C:/Projects/myLib`.

Then you would do the following:

4. Select `C:/Projects` at the first location where you find it in the Editor, then choose **Info -> Retrieve C:/Projects from this file**.
5. In the Retriever's **Change to** box, enter `$(PRJ_DIR)/..` since relative to the project directory this is the equivalent to `C:/Projects`. Then, press the **Change All** button.
6. If you want to link with your BSP from the PWE rather than from the WIND_BASE, go back to the Source Editor and select `$(WIND_BASE)/target/config` at any place where you find it. Choose **Info -> Retrieve... from this file**.
7. In the Retriever's **Change to** box, enter `$(PRJ_DIR)/..` again, since this is the correct relative reference to your BSP. Press the **Change All** button again.
8. Choose File > Check In in any tool to save your project's .wpj file. In the Check In Dialog, press the **retain lock** button to keep the file writable.

You should now open your project in Tornado again to verify that your changes were correct, and to re-generate the Make file. To do so:

On Windows: Double click on `pcPentium_T5_vx.wpj` in the Project Editor's File List

On Unix: In the Project Editor, choose **Tornado > Tornado Project Tool...**

In the Tornado Project Tool

Build your Project in Tornado to re-generate the Makefile and to make sure that your Modifications were OK. If the Build works, close Tornado and check in all files of your project (this time, you should press the OK button to make the files read-only after check in).

Using SNIFF+ Automatic Linking of Submodules

You can now apply version control to your project from SNIFF+, and you can build from SNIFF+ using the original Tornado Makefile. But you can not yet use full SNIFF+ Recursive Make. To enable this feature:

1. Check out the Project Description `pcPentium_T5_vx.shared`.
2. Add any subprojects of libraries or downloadable modules that you would like to link with your application. If you like to manage your BSP in the same workspace, you should also add the BSP project as a subproject.
3. In the Project Editor's Project Tree, Double Click on `pcPentium_T5_vx.shared`. The Project Attributes Dialog will open.
4. In the Project Attributes Dialog, choose the Build Options node and
 - enable the **Use SNIFF+ Makesupport** Checkbox.
 - delete the contents of the **Make Command** field.
5. Under the Build Options Node, choose the Build Structure node. Press the **Generate** button beside the **Recursive Make Directories** field. Verify that the entries for **Recursive Make Directories** and **Received from Subprojects** match your needs.

6. In the Project Targets Node, verify that the Executable Target is set to `vxWorks`. Below it, you will find a field for **+Libraries linked**. Here, you may specify any external libraries that should be linked to your project, and for which you have not set up SNIFF+ projects.

Note

You should specify these libraries with make macros and relative paths. You may use the macros `$(SNIFF_RELATIVE_ROOT_DIR)`, which points to your working environment root directory, `$(OBJECT_DIR)` which is set to your build specification, and any environment variables such as `$(WIND_BASE)`.

7. Press the **OK** button. SNIFF+ will ask you whether a Makefile Template should be copied to your project directory: answer **yes**. Rebuild your project to verify that everything is linked correctly.

Note:

SNIFF+ Makesupport will override the `$(EXTRA_MODULES)` Macro of your Tornado Makefile, so any modules that you configured in VxWorks Makesupport to be linked will not be taken into account. You may add external libraries in the SNIFF+ Project Attributes instead, as described above.

Beta Notice

Automatic Linking of submodules will only work with the Tornado Integration released after May 1, 2000 (t2-1.0_rc8 or later).

A good deal of Wind River Software is shipped as source code: source code that you are meant to understand, modify and re-use in your own products. SNIFF+ Browsing facilities can help you to understand such external source code more quickly, and to better handle it in managed workspaces.

This chapter is about:

- Working Environment Setup
- Setting up projects for WindRiver Source Products

Working Environment Setup

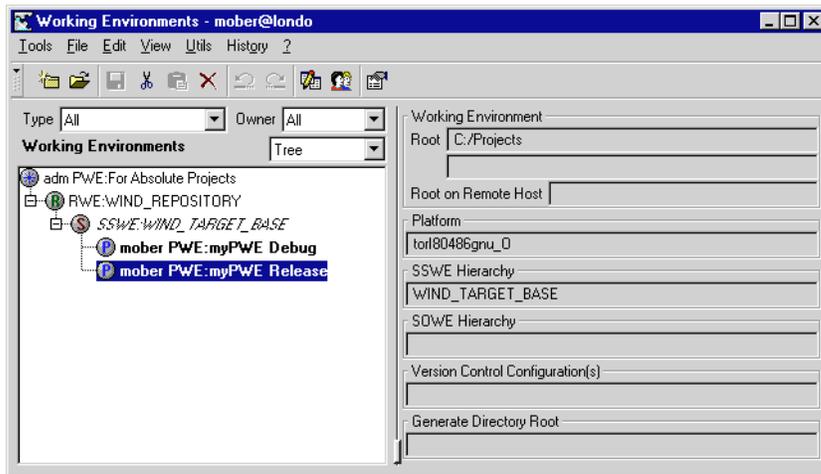
Wind River Source Code will always be installed into the `WIND_BASE/target/src` tree. You may either set up your own source code in the `WIND_BASE/target` tree as well (then you will get along with only one SNIFF+ Working Environment).

The other option is to set up a SNIFF+ SSWE onto the `WIND_BASE/target` tree, and leave original sources untouched. All modifications and local additions can then be made in a PWE.

To create this setup, open the SNIFF+ Working Environments Tool and:

1. Create an RWE called `WIND_REPOSITORY` with root directory `$WIND_REPOSITORY`
2. Below, create an SSWE called `WIND_TARGET_BASE` with root directory `$WIND_TARGET_BASE`
3. Below, create a PWE called `myPWE` with root directory `C:/Projects` (or wherever you would like to place your own projects).

Your Working Environments Tools should now look more or less like this:



Note that this time, we have set two separate PWEs, one for the Debug Build and one for the Release build. The SNIFF+ Platform for the Release Build is torl80486gnu_0 whereas the Debug build is bound to the Platform torl80486gnu. Both PWEs may reference the same Root Directory, C:/Projects, since SNIFF+ Build Specifications for Tornado automatically redirect objects into subdirectories.

Setting up SNIFF+ Projects for WindRiver Sources

To set up SNIFF+ Projects for the WindRiver Source Products you have obtained:

1. Select the SSWE:WIND_TARGET_BASE and choose File > New Project > With Template.
2. In the New Project with Template dialog, select `Tornado_BrowsingOnly.ptmpl`

Note

Although we are going to set up a shared SNIFF+ project, we take the BrowsingOnly Template because we can not expect that the unknown source code will correctly build with SNIFF+ Makesupport. The BrowsingOnly project is set up to support original Tornado Makesupport. You may switch to SNIFF+ Makesupport at any time later by selecting **Use SNIFF+ Makesupport** in the Project Attributes and deleting the contents of the **Make Command** field.

3. When the Project Editor opens, you should choose Project > Check Obsolete Files to make sure that all files are visible in the SNIFF+ project. Files may be missing if they use filename patterns that are not present in the Project Creation Template.

If the dialog shows any obsolete files, this is the best to do:

- In the Launch Pad, choose Project > Delete Project to remove the SNIFF+ project descriptions again
- Select Project > New Project > With Template again
- In the Project Attributes Dialog, show the **File Types node**. Press the **Show All** and **Add** buttons to add any file types that were missing, or edit the **Signature** field of any existing filetype.
- When you think the filetypes are correct, press the OK button to set up the project again.

Note

If you expect that you will have to set up projects with the special file type(s) more often, you may create your own project template. Just show the General node of the Project Attributes Dialog, and select the **Save as Template** radiobutton. SNIFF+ will create a new project template rather than setting up a project.

4. When all files are visible in the Project Editor, you may want to check them all in (this step is optional).

Using WindRiver Sources in Own Projects

You should create your own projects in your PWE just as usual; when you need to access WindRiver Sources, just add the corresponding project as a subproject.

Note that the special BrowsingOnly Template has set up the projects in a manner where source files are not copied to your PWE, they are merely for browsing. To copy Sources from the SSWE to the PWE, just check them out.

This chapter is a compilation of Hints and Information that you may find useful. You will find many more Tips on the SNIFF+ online FAQ and Knowledge Base at

<http://www.takefive.com/support/kb.html>

If there are still any questions or issues, please do not hesitate to contact us at any of the addresses shown in [Feedback and useful links — page 9](#).

Changing your Version Control System

To change the Version Control System for existing projects, you have to first check out all SNIFF+ Project Description Files. Then, you best use the Group Project Attributes Dialog (see [page 67](#)) to change the setting in the Version Control System node. It will depend on the CM systems used, whether migration of the Repository is possible or not.

You can also change the default RCS system in the Project Templates to your favorite CM System. To do so:

1. In the Launch Pad, select **Project > New Project > With Template**
2. In the New Project with Template Dialog, select the Template you'd like to change.
3. In the **Directory** field, enter a single dot (.) - The contents doesn't matter at this stage, since we just want to change the template. Press **OK**.
4. In the Project Attributes General Node, select the **Save as Template** Radiobutton - we recommend to do this first so as not to forget it later.
5. In the Version Control System Node, select your favorite CM system and press OK.
6. In the Save Project Template, either choose the same name as before to overwrite the template, or save it with a new name.

For more information about specifics of a particular Version Control System with SNIFF+, please refer to the corresponding documentation. Note that some Version Control Systems like Perforce and MKS SourceIntegrity need to be installed from the SNIFF+ integrations directory before they can be used. Some others, like Sablime and TeamConnection are not part of the standard SNIFF+ installation, they are rather available for download at the SNIFF+ Web site.

Enabling additional SNIFF+ Parsers

For regular Tornado projects, only the C/C++ Parser and the GNU Assembler Parser are enabled by default.

SNIFF+ contains additional Parsers for Shell Scripts, Makefiles and TCL scripts. These parsers are provided on an as-is basis, they are not supported and by default turned off.

To enable these parsers:

1. Open the Project Attributes Dialog of a project where you would like to use one of the Parsers
2. Select the File Types node and press the **Show All** Button
3. For the Makefile Parser, for instance, add the filetype **Makefile with Parser**
4. Make sure that the **Signature** field of Makefile with Parser is the same as for the Make filetype; then, remove the Filetype **Make**.
5. If you want to use the parser on a regular basis, you may modify the project template as explained above.

Note

On Windows, the SNIFF+ TCL Parser is not installed by default. It will only work if you selected **Additional Packages** and TCL/ITCL Parser during installation. If you didn't do this during first-time installation you can add it later by running the setup program again. Setup will not overwrite existing packages installed.

Other Languages Supported

SNIFF+ also comes with parsers for Java, Fortran, Ada, Python and Perl. Each of these parsers is activated by simply using the corresponding Filetype. Note that for Java, Fortran and Ada the corresponding packages must be installed during the setup procedure.

For Java in particular, SNIFF+ also provides a high-end GUI builder called Visaj. Please refer to the SNIFF+ documentation for details.

More parsers for various Assembler dialects and languages like Delphi, Cobol, Chill, VHDL and many others are available from TakeFive on request.

Optimized Compilation and Changing Build Specifications

In SNIFF+, build specifications are called *Platforms*. The SNIFF+ Tornado Integration comes with pre-set Platforms for the standard Debug and OPTIM_NORMAL settings as defined in original Tornado Makefiles. You can modify compiler settings for a particular platform on global (working environment) level, on project level and on file level.

Creating a new Build Specification

To create a new build specification:

1. select Tools > Preferences from any SNIFF+ Tool.
2. In the Preferences Dialog, select the Platforms node.
3. In the Platforms display, choose any platform that most closely resembles what you want. Press the **Copy** button and copy it to a new name.
4. In the lower part of the Preferences Dialog, select the **Make Support** tab. You may now specify directory names where your objects and targets should be stored; and, most important, you must specify a name by which this Platform will be internally recognized.

Enter this name in the **Platform Makefile** field. We usually recommend to have this name match the name of your Platform: for example, choose `torI80386_OPTIM_DRIVER`.

5. Actual make settings are then stored in a file called `SNIFF_DIR/make_support/torI80386_OPTIM_DRIVER.mk`. Create this file by copying any of the existing `*.mk` files into it.
6. In your Platform Makefile, you may override and redefine any of the make macros you know from Tornado Makefiles.
7. To assign your new build specification to a working environment, open the Working Environments tool, select a PWE or an SOWE and choose **Edit > Modify**.

Changing Build Specifications on Project Level

In the Project Attributes Dialog, select the Directives node (below Build Options). You may use the **Additional** field to enter any compiler options that should be valid for all build specifications.

If you want to use particular compiler options only for particular build specifications, press the **new** button and select any platform that should receive special handling. You may now enter directives per Platform.

If the GUI is not versatile enough for the settings you need to take, you can always edit the local `Makefile_sniff` to accommodate your changes: it contains a load of comments that shows you what you can customize and where.

In SNIFF+, the `Makefile_sniff` is treated like a source file: it is never overwritten, and it should be handled with regular Version Control. So any changes you make to the `Makefile_sniff` will be persistent.

Changing Build Specifications on File Level

In order to enable file-level compiler options, you have to modify the file `SNIFF_DIR/make_support/general.c.mk` and add / modify the following lines:

```
CC_OPTION_FILE=`if [ -f $*.cop ]; then \  
    cat $*.cop; else echo ""; fi`  
  
CXX_CMD = $(CXX) $(ALL_CXXFLAGS) $(CC_OPTION_FILE) \  
    $(OVERALL_OPTION_CXX) "$<" $(CXX_OUTPUT_FLAG)  
  
CC_CMD = $(CC) $(ALL_CFLAGS) $(CC_OPTION_FILE) \  
    $(OVERALL_OPTION_CC) "$<" $(CC_OUTPUT_FLAG)
```

Now, for every file `toto.c` where you need special compiler options, you just add a file called `toto.cop` that contains your compiler options (the filetype for `*.cop` files is already defined in the Tornado Project Templates). Future versions of the SNIFF+ Tornado integration will have the two lines mentioned already pre-set in the makefiles.

Filtering Make Messages

The default setting for your Tornado Platforms is to have `make` output all commands as it executes. You can filter these messages to the most important ones by using the switch `DEBUG=0` either in your Platform Makefile, or on the Make commandline.

Extending Custom Menus

The Tornado menu in your Project Editor and Source Editor can easily be extended to accommodate additional tools that you need more often. You can also easily add additional menus. Just add the file `SNIFF_DIR/config/SiteMenus.sniff`. For more information, see the `SNIFF+` User's Guide.

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.

© 1992-2000 TakeFive Software GmbH, a Wind River Company.
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

