

SNiFF+™

Version 3.2 for Unix and Windows

Fortran Tutorial



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

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

Copyright

Copyright © 1992–1999 TakeFive Software Inc.

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

Parts of SNIFF+:

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

Portions copyright 1991-1997 Compuware Corporation.

Trademarks

SNIFF+ is a trademark of TakeFive Software Inc.

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

Credits

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

Authors of the first version:

Walter Bischofberger (Sniff)

Erich Gamma (Sniffgdb)

Erich Gamma and André Weinand (ET++)

Table of Contents

About this Manual	1
Conventions	1
Tool elements	2
Typography	3
Feedback and useful links	3
Road Map	5
The SNIFF+ Fortran Tutorial	5
Creating a Single-User Project	7
Creating the single-user project	7
Examining the results	10
Conclusions	10
Using the Symbol Browser	11
Using the Symbol Browser	12
Using the Cross Referencer	15
Performing function body cross referencing	16
Abbreviations used in the Cross Referencer	18
Browsing Examples	21
Browsing global variables	21
Browsing label references	22
Browsing statement functions	23
Browsing includes	24
Browsing parameters	25
Parser Options	29
File include option	29
Displaying syntax errors in the SNIFF+ Log tool	32
Changing the tabulator size for fixed source form	33
Changing the line length for fixed source form	33
Changing case sensitivity mode for symbols	34
Building the Project's Executable	35
Setting up Make Support	35
Building the project target	36

Browsing a Fortran 90 Project	37
Creating a single-user project.	37
Browsing modules	39
Fortran 90 entries in the Symbol Browser's Type drop-down.	41
Browsing derived types.	42
Browsing named DO statements	42

About this Manual

What this manual is

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

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

Conventions

One basic term

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

Two conventions: menu references

For clarity and to avoid undue verbosity, the phrase:

"Choose the MenuCommand from the MenuName" is presented as follows:

- Choose **MenuName > MenuCommand**.

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

Context menu, and consequently:

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

- Choose **Context menu > MenuCommand**

A note on Unix/Windows

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

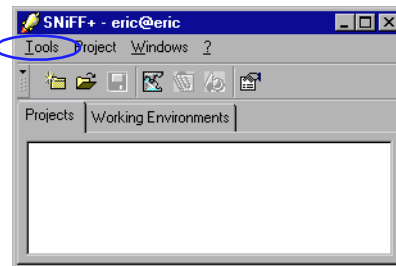
When you start SNIFF+, the first tool that appears is the Launch Pad. In this and other SNIFF+ tools, the first item in the menu bar is for launching tools.

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

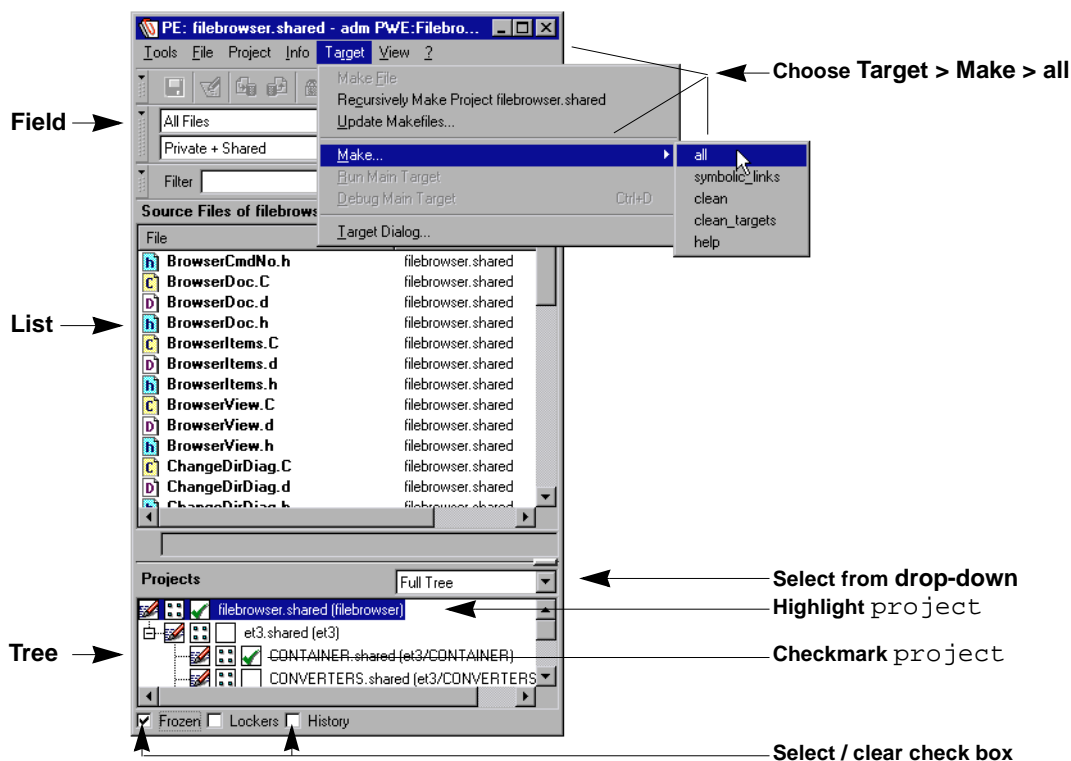
When we refer to this menu in order to launch a tool from the Launch Pad, or any other open SNIFF+ tool, we will use the notation:

Choose **Tools > ToolName**.

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



Tool elements



Typography

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

Feedback and useful links

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

Europe:

sniff-support@takefive.co.at

USA:

sniff-support@takefive.com

Useful links

SNiFF+ web pages:

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

Introduction

This manual introduces the SNIFF+ solution for Fortran development and is centered around a two-part tutorial.

What this manual is not

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

The SNIFF+ Fortran Tutorial

There are seven chapters in this tutorial. Each chapter covers a task that you will routinely perform when working with single-user Fortran projects.

In the first six chapters, you will create and work with a single-user project in SNIFF+ based on a Fortran77 example called EVCLID, which is a program for making complex geometric calculations. In the last chapter, you will learn how to use SNIFF+ to browse Fortran90 extensions made to Fortran77.

Here is a description of the seven chapters:

- **Task 1.** [Creating a Single-User Project](#)—In this chapter, you will learn how to create a single-user project for the EVCLID project.
- **Task 2.** [Using the Symbol Browser](#)—In this chapter, you will learn how to use the Symbol Browser as a starting point for browsing your project's code.
- **Task 3.** [Using the Cross Referencer](#)—In this chapter, you will learn how to use the Cross Referencer to follow references in your source code.
- **Task 4.** [Browsing Examples](#)—In this chapter, you will learn how to browse a variety of different Fortran 77 symbol types, as well as follow include statements in your source files.
- **Task 5.** [Parser Options](#)—In this chapter, you will learn how to configure a number of parser options for parsing FORTRAN77 and 90 code.
- **Task 6.** [Building the Project's Executable](#)—In this chapter, you will set up SNIFF+'s Make Support for the EVCLID project and then build its executable.
- **Task 7.** [Browsing a Fortran 90 Project](#)—As in Task 1, you will create a single-user SNIFF+ project for the Fortran 90 example code provided with your SNIFF+ installation.

Note

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

Creating a Single-User Project

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

If you didn't select the "Other Packages" option (for Fortran) during the SNIFF+ installation process, start the SNIFF+ installation again and select only this option.

Goals of this chapter

In this chapter, you will learn how to use SNIFF+'s Project Setup Wizard to create a single-user project for the example code.

Creating the single-user project

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

In the Project Setup Wizard

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

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

The "Select development task" page appears.

In the remaining steps, we will refer to the names of Wizard pages. You can find a page's name in the upper left corner of the Wizard.

In the "Select development task" page

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

In the "Your development organization" page

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

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

In the "Select file types" page

- Select **Fortran77** and press **Next**.

SNIFF+ will automatically include all the necessary file types needed for working with Fortran77 in the new project. Note that, after project setup, you can add new standard file types (like the ones in the "Additional File Types Column"), or create and add your own.

In the “Specify Private Working Environment” page

You are asked to specify your *Private Working Environment* (PWE) root directory, which is the directory that contains your source code.

1. Press **Browse**, and in the Directory dialog, navigate to the root directory of the example code, which is:

```
<sniff_installation_dir>/example/fortran
```

2. Double-click on `Fortran` and then press **Select**.
3. In the **PWE name** field, type a name for the PWE, e.g., `Fortran`.

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

1. Press **Next**.

In the “Create new SNIFF+ Project” page

SNIFF+ has set your **Project root directory** to `fortran`, which is the root directory of your source code. The project has the default name `fortran`.

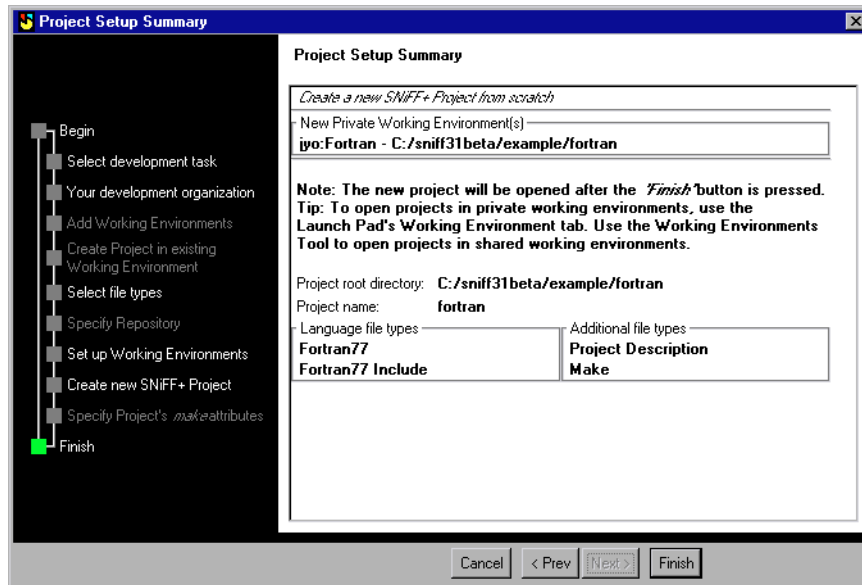
Also by default **Create Subprojects** is enabled, which is correct.

- Select the **Use SNIFF+'s Makefiles** checkbox and press **Next**.

In the “Project Setup Summary” page

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

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



- Press **Finish**.

SNIFF+ will now generate the single-user project and its associated files.

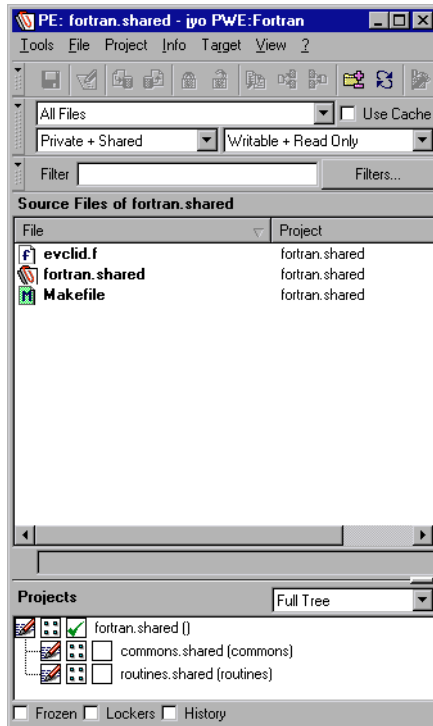
- In the dialog that appears asking if you want to generate cross reference information, press **No**.

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

When the generation process is over, SNIFF+ automatically opens the new project and displays its project structure in a Project Editor.

Examining the results

The Project Editor on your screen should look something like this:



Conclusions

You have just created a single-user project for browsing the Fortran 77 project. Starting with the next chapter, you will learn how to use the various browsing tools available in SNIFF+. In the last chapter in this tutorial, you will set up SNIFF+'s Make Support for the project and then build the project's executable.

Using the Symbol Browser

Goals of this chapter

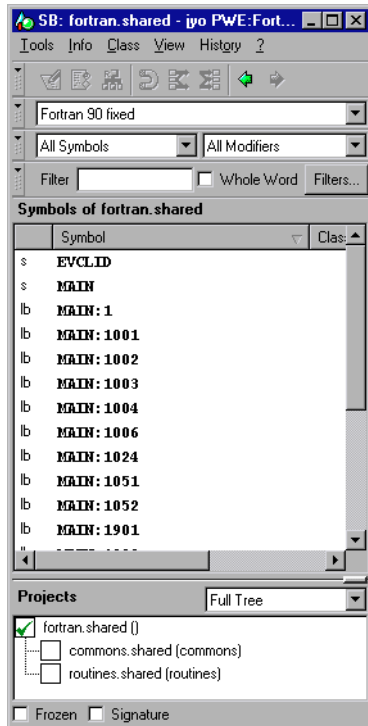
In this chapter, you will learn how to use the Symbol Browser as a starting point for browsing your project's code. The Symbol Browser allows you to browse all global symbols and symbol members of a set of projects. It offers a wide range of possibilities for filtering information.

The Symbol Browser consists of a list of symbols whose content is determined by the **Symbols** and **Modifiers** drop-down menus, the Project Tree settings, and the Filter field. The Project Tree shows the project structure and makes it possible to select the projects whose symbols are to be displayed. For detailed information about the Symbol Browser, please refer to your SNIFF+ online or documentation or the *Reference Guide*.

Using the Symbol Browser

1. In the Launch Pad, choose **Tools > Symbol Browser**.

The Symbol Browser appears.



2. Take a look at the **Language** drop-down menu. The SNIFF+ Fortran Parser is actually a Fortran 90 parser that understands the Fortran 77 subset of the language. For Fortran 77 projects, the language string displayed in the drop-down menu is **Fortran 90 Fixed**. “Fixed” refers to the fixed input format in Fortran 77 as opposed to the “free” input format supported in Fortran 90.
3. Let’s look at the symbols in all the projects in the Project Tree. To do so, right-click anywhere in the Project Tree and choose **Context menu > Select From All Projects**.
4. Choose the various entries in the **Symbols** drop-down menu and see what happens. By selecting the **Signature** check box, you can see in which files the symbols appear. Also, information about data types, modules, dummy arguments and return types will be displayed.

Note that modules and derived types are not present in this project (since they are Fortran 90 data types), resulting in an empty Symbol List when you select them.

5. From the **Symbols** drop-down, choose **subprogram**.

Now all subprograms in the project are listed in the Symbol List.

`EVCLID` is the main program unit of the project and the starting point from which all other function calls take place.

6. Double-click on `EVCLID`.

A Source Editor appears and the main program file `evclid.f` is loaded into it. As you have just found out, double-clicking a symbol in the Symbol List opens a Source Editor and loads the file in which the symbol appears. The cursor is automatically positioned to the symbol in the file.

7. Back in the Symbol Browser, in the Symbol List, double-click on `DGECOM`.

The Source Editor now shows the definition of subroutine `DGECOM` in `evclid2.f`. In the next chapter, we will use `DGECOM` to show how you can perform function body cross referencing in `SNiFF+`.

Using the Cross Referencer

Introduction

In this chapter, you will learn how to use the Cross Referencer to follow references in your source code.

Basically, there are three different ways of following references:

- **Function body cross referencing**
- **Component analyzing**
- **Interface cross referencing**

When browsing FORTRAN 77 code, only function body cross referencing is possible in Cross Referencer. This is because component analyzing and interface cross referencing are based on data structures not implemented in FORTRAN 77. However, these data structures are implemented in FORTRAN 90. As a result, all three ways of following references can be performed when browsing your FORTRAN 90 code.

Function body cross referencing

With function body cross referencing, you can find out which symbols are referred to by subprograms (subroutines and functions) and module subprograms. You can also find out which subprograms refer to a particular symbol.

To start method body cross referencing, use the Symbol Browser to select a particular subprogram and choose **Info > Subprogram Refers-To....** A Cross Referencer appears with cross referencing information for the selected subprogram.

To find out which subprograms refer to a particular symbol, use the Symbol Browser to select the symbol and then choose **Info > Symbol Referred-By....**

Component analyzing

Modules and derived types can contain components. By doing component analyzing with the Cross Referencer, you can answer questions like “What are the components (variables or parameters) of a module or the components of a derived type displayed to level 5” or “What modules or derived types have INTEGER variables/components?”.

To do component analyzing, use the Symbol Browser to select a module or a derived type and choose **Info > Symbol Refers-To Components** (Since there aren't any modules or derived types in this example, you won't get a result - try this command out with your own examples).

Interface cross referencing

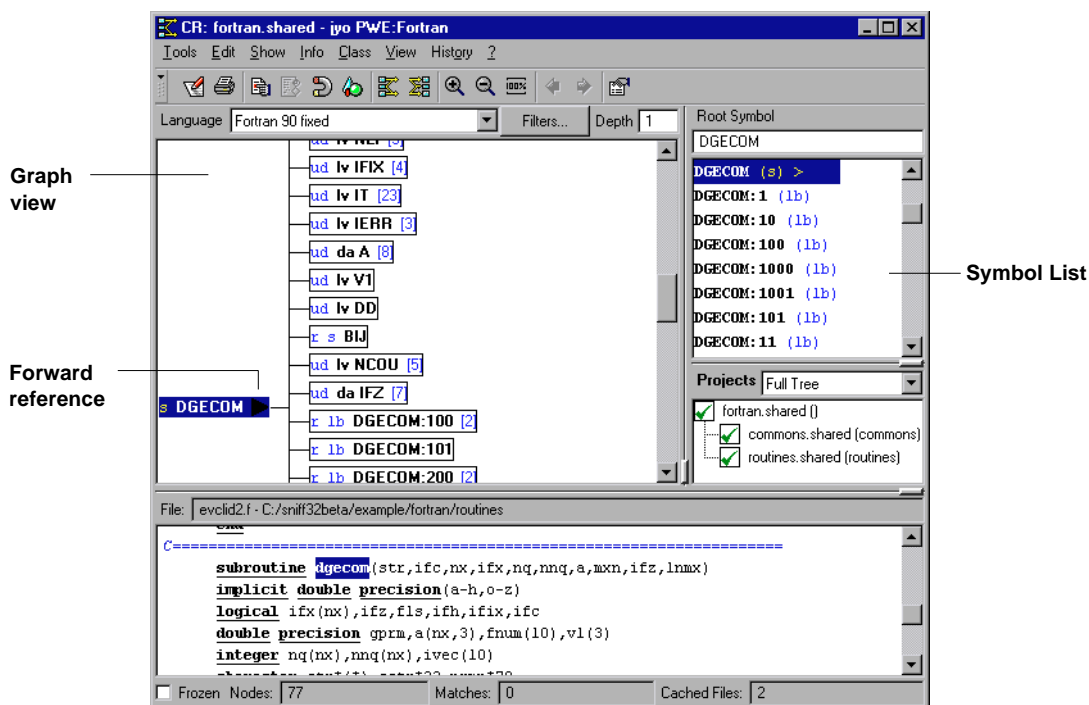
You can do interface cross referencing modules; that is, you can cross reference all symbol types that are part of a module's interface (i.e., return values or dummy argument types). You can also display all types that use another type in their interface. Questions like: "Who returns a CHARACTER type?" or "What are the return and dummy argument types of this module?" can be answered.

To do interface cross referencing, start the Cross Referencer and switch on interface cross referencing by enabling the **Interface (PR)** check-box in the Filter dialog.

Performing function body cross referencing

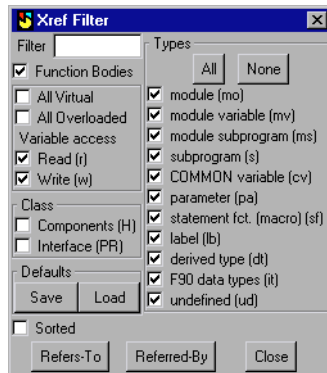
1. In the Source Editor, make sure that subroutine `DGECOM` is selected in either the Symbol List or in the Text View. Then, choose **Info > DGECOM Refers-To**.

A Cross Referencer appears. Subroutine `DGECOM` is shown in with all the symbols it refers to (77 nodes). For a description of the abbreviations used in the Cross Referencer, please refer to [Abbreviations used in the Cross Referencer — page 18](#).



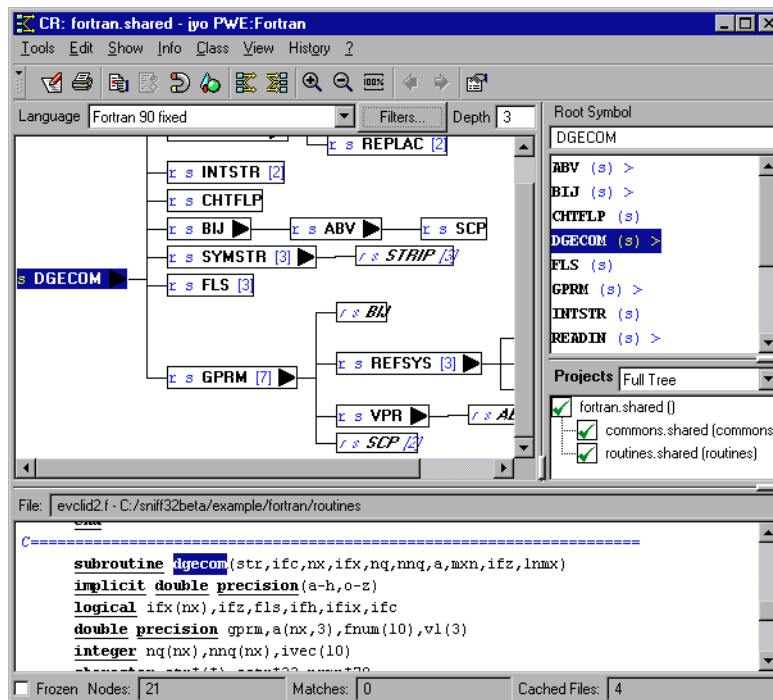
- Let's see which subroutines and functions are called by those which DGECOM calls. To do so, first enter **3** in the **Depth** field. Then, press the **Filters...** button.

The Filter dialog appears.



- Limit the scope of the next query to subroutines and functions by pressing the **None** button under **Types** and then clicking on the **subprogram (s)** check box.
- Now, with **s DGECOM** selected in the Cross Referencer's **Graph** view, press the **Refers To** button in the Filter dialog.

You should now see a call tree similar to the following:



5. Reset the Depth field to 1 and press Return.
6. Close the Cross Referencer and Source Editor tools.

Abbreviations used in the Cross Referencer

Symbol types that can be referred by a subprogram are listed in the following table. The table also contains abbreviations for the symbol types used in the Cross Referencer.

Symbol	Abbreviation
module	mo
module variable or module parameter	mv
module subprogram	ms
subprogram (main program units, functions and sub-routines)	s
COMMON variable	cv
parameter	pa
statement function	sf
label	lb
derived type	dt
C data type	it
r56	bo
F90 data types	it
user defined (User-defined symbols are block data, common blocks and named control statements. They cannot be referred to or referred by in the Cross Referencer.)	de

Abbreviations for symbol types are also displayed in the Filter dialog.

Cross Referencing undefined symbols

Not only SNIFF+ symbols (symbols residing in the Symbol Table and shown in the Symbol Browser) can be cross referenced, but also some local constructs and intrinsic functions. SNIFF+ treats these constructs as undefined symbols because they can't be found in the Symbol Table. Undefined symbols have the abbreviation `ud`. Further information about each construct is given by additional abbreviations that follow `ud`. The following table lists the undefined symbols:

Undefined symbol	Abbreviation
local variable	<code>ud lv</code>
dummy argument	<code>ud da</code>
result variable	<code>ud rv</code>
local parameter	<code>ud pa</code>
intrinsic function or subroutine	<code>ud if</code>

A result variable (`rv`) may be the name of a function itself or the result variable given by the `RESULT` keyword in the function definition.

Browsing Examples

Goals of this chapter

In this chapter, you will learn how to browse a variety of different Fortran 77 symbol types, as well as follow include statements in your source files.

Browsing global variables

1. In the Symbol Browser, from the **Symbols** drop-down, choose **COMMON variable**.
2. Select the **Signatures** check box.

All common variables defined in the project, their types and in which files they are defined are now displayed in the Symbol List.

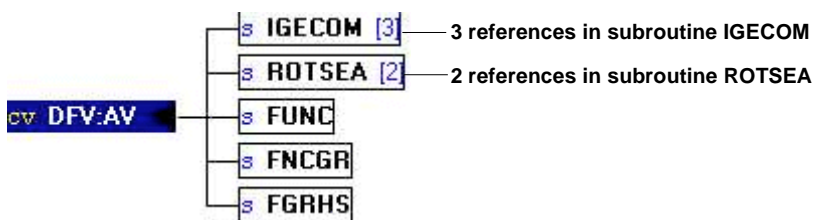
3. Notice that there are two entries in the Symbol List that begin with

```
cv DOUBLE PRECISION(:,:) DFV:AV
```

Both entries refer to the double precision variable AV, which is defined in common block DFV. In the first entry, the common block is defined in the file `comdopr.inc`. In the second entry, it is defined in the file `evclid1.f`.

4. Select the entry in which the common block is defined in `evclid1.f`.
5. Choose **Info > DFV:AV Referred-By**.

The Cross Referencer appears. In the Graph view, you should see the subroutines and functions that refer to the common variable AV.



6. Let's jump to positions in the subroutine IGECOM where AV is referred to. Select IGECOM in the Graph view and press <SHIFT>-double click.
The Source Editor appears and the file `evclid1.f` is loaded into it. The cursor is positioned to the first position in IGECOM where AV is referred to (in line 582).
7. Let's jump to the other references to AV in the subroutine. Choose **Show > Next Match**.
The cursor should now be in line 623.
8. Choose the command a second time. The cursor is now in line 774.

9. Choose the command a third time. The cursor is now in line 1058. Notice that we're no longer in subroutine IGECOM, but in ROTSEA.
10. Close the Source Editor and Cross Referencer.

Browsing label references

1. In the Symbol Browser, from the **Symbols** drop-down, choose **label**.
All labels defined in the project are now displayed in the Symbol List.
2. Select **IGECOM:1030** from the Symbol List. This entry is for label 1030 defined in the subroutine IGECOM in file `evclid1.f`.
3. Choose **Info > IGECOM:1030 Referred-By**.
The Cross Referencer appears. There are 5 references to the label 1030, all in subroutine IGECOM.



4. Let's jump to positions in the subroutine IGECOM where label 1030 is referred to. Select IGECOM:1030 in the Graph view and press <SHIFT>-double click.
The Source Editor appears and the file `evclid1.f` is loaded into it. The cursor is positioned at the definition of label 1030 (in line 1034).
5. Let's jump to the first reference of 1030 in the subroutine. Choose **Show > Next Match**. The cursor should now be in line 387.
6. Go to line 388. If you want, use the **Go To Line...** command is available in the **Edit** menu.
Line 388 is a GOTO statement to label 80. Let's go to label 80 in the file.
7. Position the cursor in 80 or select 80 by double-clicking it. Then, choose **Show > Symbol(s) 80...**.
The Choose Symbol dialog appears. The dialog appears whenever SNiFF+ finds more than one symbol of the same name that matches a symbol request, or when multiple matches are found after the **Show > Symbol(s) symbol...** command is used.
8. In the Choose Symbol dialog, select the **Show listing of files** check box. This enables you to see the file names and projects in which the symbol appears.
9. Select the **Scan only included files** check box.
Only one entry is displayed in the dialog. This entry refers to the position in file `evclid1.f` where label 80 is defined.
10. Select the entry and press the **Definition** button to jump to the label's definition.
11. Note that the label's definition is in line 317. By following the steps outlined above, you can quickly jump to a label's definition without have to scan through your source code.
12. Close all the open tools except for the Launch Pad and Symbol Browser.

Browsing statement functions

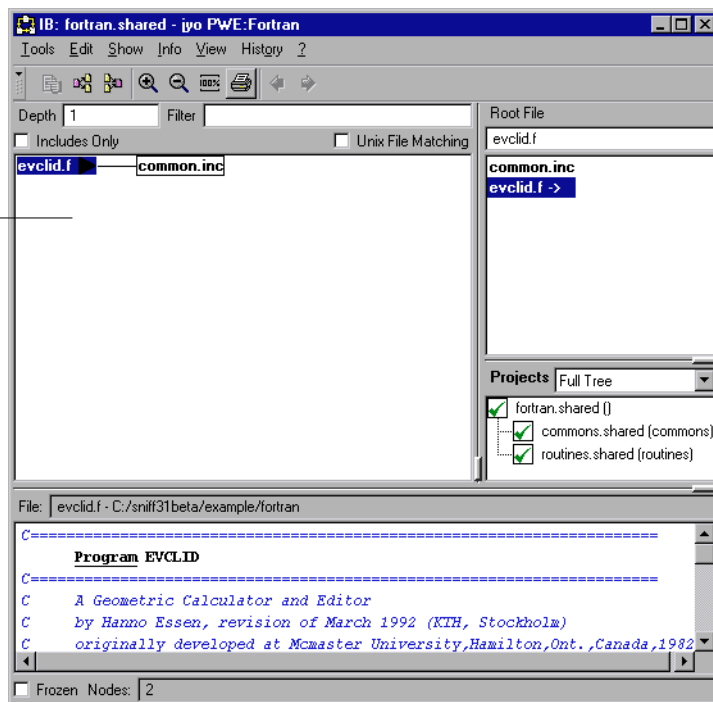
1. In the Symbol Browser, from the **Symbols** drop-down, choose **statement fct. (macro)**.
One statement function is defined in the project. The statement function `DLT` is defined in the subroutine `FGRHS`.
2. Double-click the entry in the Symbol List to jump to the statement function's definition.
A Source Editor appears and is positioned to line 1351 in the file `evclid1.f`.
3. Choose **Info > FGRHS:DLT Referred-By** to see if any other subprograms refer to this statement function.
The Cross Referencer appears. As you can see, `DLT` is referred to by subroutine `FGRHS` in four different places.
4. If you want, select `FGRHS` in the Graph view and press `<SHIFT>`–double click to jump to the first reference to the statement function in the subroutine.
5. Close all open tools except for the Launch Pad and the Symbol Browser.

Browsing includes

1. Use the Launch Pad to open the Project Editor. In the Project Editor, make sure that all projects in the Project Tree are checkmarked and select file `evclid.f` in the File List.
2. Then, choose **Info > evclid.f Includes**.

The Include Browser appears. The Include Browser graphically displays include references between files in your projects.

Graph
view



3. Let's see which other files include `common.inc`.

In the Graph view, select `common.inc` and choose **Info > Included-By**.
The Graph view now also displays the files that include `common.inc`.



4. Close the Include Browser.

Browsing parameters

1. In the Symbol Browser, from the **Symbols** drop-down, choose **parameter**.

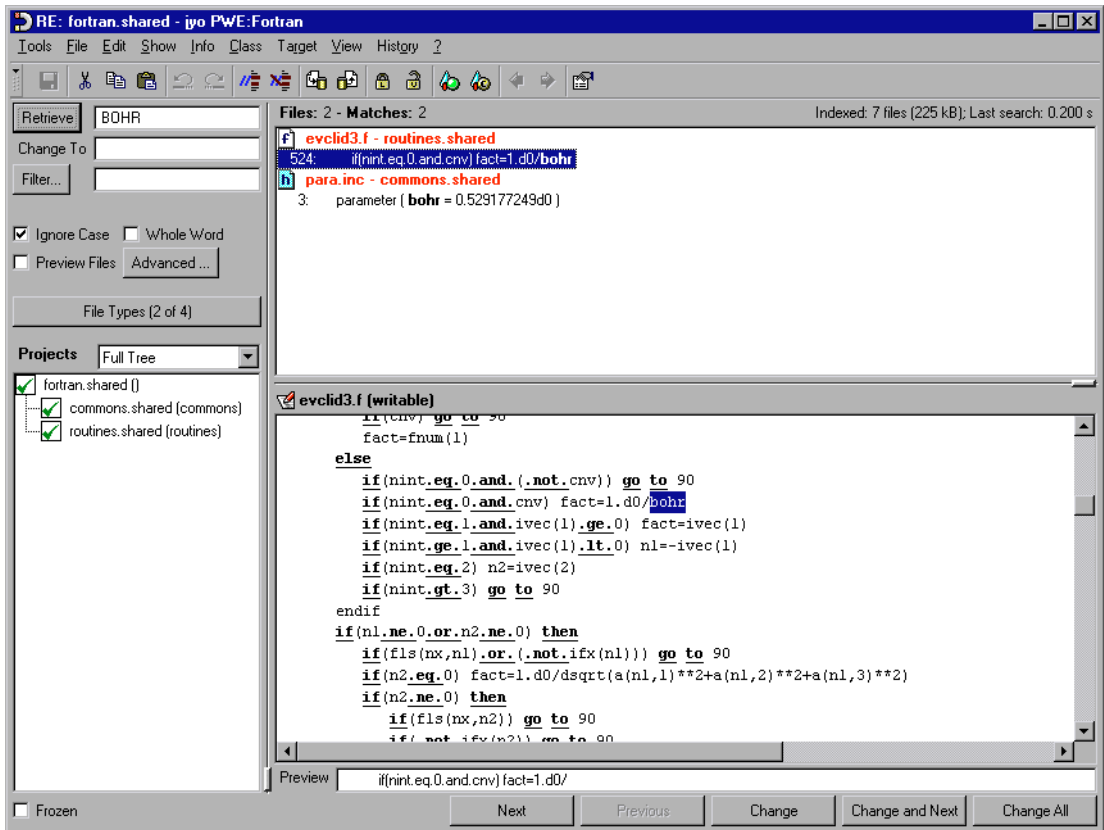
One parameter is defined in the project. The parameter `BOHR` is defined in the include file `para.inc`.

2. Let's see how often the parameter `BOHR` appears in the project. Select the entry in the Symbol List and choose **Info > Retrieve BOHR From All Projects**.

The Retriever appears. The Retriever (like a super-grep in Unix) starts a full text search over all files in projects checkmarked in the Project Tree. Information provided with a list of matches consist of the corresponding source file, the string that was matched, and the source line containing the match.

According to it, the string "BOHR" does not appear in any of the projects. However, we know that this cannot be the case.

3. In the Retriever, select the **Ignore Case** check box and press the **Retrieve** button.
Two matches have been found, and the Retriever now displays them properly.



By default, SNIFF+ displays all Fortran symbol names in uppercase, regardless of how the symbols are written in your source code. As a result, when searching for a particular symbol in the Retriever, SNIFF+ may not be able to find matches to symbols written in lowercase in your code, as was just demonstrated above. You have two options for dealing with this problem:

- If case-sensitivity only matters to you when using the Retriever, select the **Ignore Case** check box before starting a query.
 - If you want all of SNIFF+'s browsing tools to display symbol names in lowercase, or in the same way as written in the source code, you can set a special parser option to do this. See also [Parser Options — page 29](#).
4. In the Retriever, double-click the first entry displayed in the File Matches list to jump to the line in the source code where SNIFF+ found a match (in file `evclid3.f`).
A Source Editor appears and is positioned to line 524 in `evclid3.f`.

5. Leave the Source Editor open and close all other tools except for the Launch Pad and Project Editor. Do not change the position of the cursor in the Source Editor. In the next chapter, we will continue from this point.

Parser Options

Goals of this chapter

In this chapter, you will learn how to configure a number of parser options for parsing FORTRAN 77 and 90 code.

The Fortran Parser allows you configure certain parser options. These options are described below. In the general, to configure these options for a given project:

- Open the Project Attributes dialog, or the Group Project Attributes dialog for configuring parser options for multiple projects at the same time.
- Select the **Build Options - Directives** node.

The **Preprocessor Directive(s)** field is available in this view. This field was originally used for entering preprocessing directives for C/C++ projects. Since such directives are not needed for Fortran projects, we can use the field to specify parser options.

File include option

When the file include option is configured, SNiFF+ also resolves Fortran include statements found in a project's source files. As a result, parsing will be slower (since additional files must be parsed). On the other hand, more exact cross referencing information is delivered by the parser to SNiFF+'s Symbol Table and consequently to the various browsing tools.

Browsing with file include option switched off

To demonstrate how the file include option works, let's first see what happens when this option is NOT configured.

1. In the Source Editor's Symbol List, click GSCCOM to position the cursor to subroutine's definition.
2. Choose **Info > GSCCOM Refers-To** to see which symbols are referred to by the subroutine.

The Cross Referencer appears.

3. Scroll down the Cross Referencer's Graph view or the Symbol List until you come to a reference to the symbol BOHR.

As we know from last chapter, BOHR is a parameter defined in `para.inc` and referred to by subroutine GSCCOM in the file `evclid3.f`. However, both the Graph view and the Symbol List tell us that BOHR is an undefined local variable. See also [Abbreviations used in the Cross Referencer — page 18](#):

ud 1v BOHR

Explanation

Since the file include option is not configured, no persistent cross reference information about BOHR was stored when the parser parsed `evclid3.f` and `para.inc`. As a result, SNIFF+ treats the two occurrences of BOHR in the project as independent of each other.

In `evclid3.f`, BOHR was implicitly declared a local variable during parsing since no definition of it was found. In `para.inc`, BOHR was correctly declared as a parameter.

Browsing with file include option switched on

To switch on the file include option and browse the project:

1. In the Project Editor, make sure that all three projects in the Project Tree are checkmarked and choose **Project > Attributes of Checkmarked Projects....**

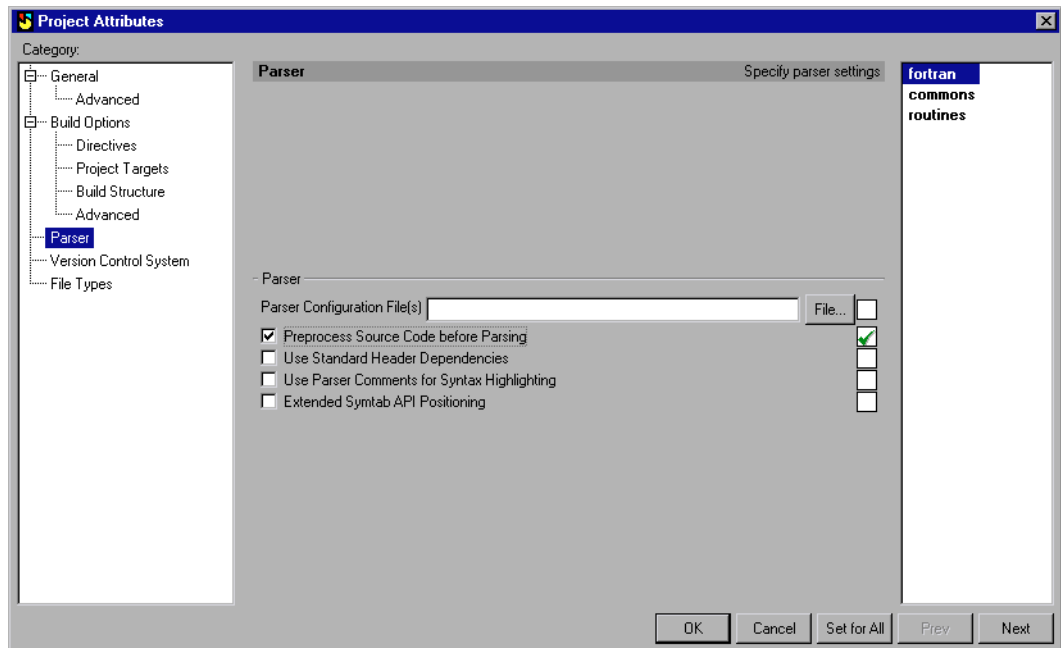
The Group Project Attributes dialog appears. You can use this dialog to modify the attributes of all the projects checkmarked in the Project Tree at one time.

2. Select the **Parser** node.

3. In the **Parser** view, select the **Preprocess Source Code before Parsing** check box. Also select the check box directly to its right.

Clicking on the **Preprocess Source Code before Parsing** check box activates the file include option. Clicking on the check box to its right activates this option for all the checkmarked projects.

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



4. Press the **Set for All** button to apply the settings to the checkmarked projects.
5. From the **Build Options** node, select **Directives**.
6. Click on the check box to the right of the **Generate** button and press the **Set for All** button to apply the setting to the checkmarked projects.

This generates the include paths for the checkmarked projects. The include path will be entered into the **Include Directive(s)** field next to the **Generate** button.

7. Press the **Ok** button to apply the settings.

A dialog appears, in which you are asked whether Makefiles should be updated.

8. Press **No**, as you will be updating Makefiles at a later time in this tutorial.

In the Project Tree of the Project Editor, all the projects are now indicated as being modified.

9. Use the Launch Pad to save `Fortran.shared` and its subprojects.

10. In the Project Editor, choose **Project > Force Reparse** to reparse all project files. In the dialog that appears, press **Yes** to confirm the force reparse.

In general, it is unnecessary to execute this command to reparse your project files, since SNIFF+ itself reparses your project files when changes have been made to them.

11. Now, perform steps 1 through 3 from the section [Browsing with file include option switched off — page 29](#). When you are done, BOHR should be listed as a parameter in the Cross Referencer:

 pa BOHR

We recommend that you switch on the parser include option when you create new SNIFF+ projects for your Fortran source code. This will guarantee that cross referencing information is exact. However, please be aware that parsing will take longer as a result.

Note

The file include option does not affect the functionality of the Include Browser. Includes are always shown in the Include Browser regardless of the state of the **Preprocess Source Code before Parsing** check box.

Displaying syntax errors in the SNIFF+ Log tool

The Fortran Parser exactly parses correct programs. Note that the Parser can also provide symbol information to SNIFF+ for incomplete or non-compilable programs. However, symbol information obtained from such programs may lead to inconsistencies in the way symbols are displayed and highlighted in SNIFF+.

You can have SNIFF+ report any syntax errors it comes across when parsing Fortran source files. These errors are displayed in the Log tool. When you parse new files or reparse already-parsed files, the Log tool displays messages about possible syntax errors, warnings and information about which files were parsed and how the syntax errors were handled. These messages include a position field which indicates the line and column in the parsed source file which resulted in the error. By correcting any syntax errors that the Parser comes across, and then reparsing the files in question, you can guarantee that SNIFF+ receives accurate symbol information.

To see how the syntax error option works, let's first set it:

1. In the Project Editor, choose **Project > Attributes of Checkmarked Projects...**

The Group Project Attributes dialog appears.

2. Select the **Build Options - Directives** node.
3. In the **Directives** view, enter `-e` in the **Preprocessor Directive(s)** field. Also click on the check box directly to the right of the field.

This switches on the syntax error option for all the projects.

4. Press the **Set for All** button to apply the settings.
5. Press **Ok** to apply the changes to all the projects.

Now let's introduce a syntax error into an error-free file and see what happens:

1. Load the file `evclid3.f` into a Source Editor and position the cursor to line 1621 of the file. This line contains the following code:

```
double precision function abv(v1)
```

2. Introduce the syntax error by changing line 1621 to the following:

```
double precision functio abv(v1)
```

3. Open the Log tool by choosing **Tools > Log** in any open tool.
4. Save `evclid3.f`. The file will now be reparsed.
5. Take a look at the Log tool. Look for the following message:

```
Error (sniff_fortran_fixed): C:/sniff31beta/example/fortran/
routines/./commons/common.inc(1621)c24: [RECURSIVE]FUNCTION name
expected
```

As you can see from this message, the Parser found the error which you just introduced.

6. Take a look at the Source Editor's Symbol List. You should notice the following entry in it:

```
FUNCTIONOABV (s)
```

This entry is obviously a result of the syntax error in line 1621.

7. Take a look at the **Info** menu. You should be able to notice other inconsistencies that are due to line 1621.
8. Fix the syntax error in line 1621 and resave (reparse) file `evclid3.f`. Notice that no errors are reported in the Log tool. Furthermore, the Source Editor's Symbol List no longer contains the entry `FUNCTIONOABV (f)`.

Changing the tabulator size for fixed source form

Using tab characters in your fixed source form code may result in parsing errors, depending on the size of the tab. The default tab length is 4 blanks. If you want to change this default, enter `-t nn` in the **Preprocessor Directive(s)** field. nn stands for the new tab length.

For example, if you enter `-t5` in **Preprocessor Directive(s)** field, the parser will treat tab characters in your fixed source code file like 5 blanks spaces.

Changing the line length for fixed source form

You might also want to use a different line length than the default for your fixed source code files. The default line length is 160 characters a line. Characters beginning on column 73 to the end of a line are treated as comments.

To change the default line length, enter `-l nnn` in **Preprocessor Directive(s)** field. nnn stands for the new line length (e.g., `-l132` sets the default line length to 132 characters).

Changing case sensitivity mode for symbols

By default, SNIFF+ displays all Fortran symbol names in uppercase, regardless of how the symbols appear in your source code. SNIFF+ can also display symbol names in lowercase, or in the same way as written in the source code.

You can set SNIFF+'s case sensitivity mode by entering `-cx` in the **Preprocessor Directive(s)** field. `x` stands for one of three modi:

- `u` uppercase letters (default)
- `l` lowercase letters
- `s` use original typing from source code

Note

- If you enter `-cx` in the **Preprocessor Directive(s)** field, you will have to use the name of a symbol as it appears in your source code to refer to the symbol in SNIFF+ (e.g., when searching for a particular symbol from the Retriever).
- The options entered in the **Preprocessor Directive(s)** field must be separated by blanks. Each option begins with a `-` character. The options can be entered in any order.

Building the Project's Executable

Goals of this chapter

In this chapter, you will set up SNIFF+'s Make Support for the project and then build its executable.

Note

In order to complete the last section in this chapter, [Building the project target](#), you must have a Fortran compiler installed on your machine!

Setting up Make Support

Setting up Make Support for routines.shared

1. In the Project Tree of the Project Editor, double-click `routines.shared` to open its Project Attributes dialog. In this dialog, you can look at and modify all the attributes of a particular project.
2. Under the **Build Options** node, select **Project Targets**.
3. Press the **Generate** button next to the **Include Directive(s)** field to generate the include paths for the project.
4. Under the **Build Options** node, select **Build Structure**.
5. From the **Passed to Superproject** drop-down, choose **Object Files + Received**.
You can use the **Passed to Superproject** drop-down menu to export the object files built in the project to its superproject. These object files are then used to build the targets of the superproject.
6. Press **Ok** to save the changes to the project attributes.
A dialog appears, in which you are asked whether Makefiles should be updated.
7. Press **No**. You will update Makefiles in the next sections.

Setting up Make Support for Fortran.shared

1. In the Project Tree of the Project Editor, double-click `Fortran.shared` to open its Project Attributes dialog.
2. From the **Build Options** node, select **Project Targets**.
3. In the **Executable**, enter a name for the project's executable (e.g., `evclid`).
4. Press the **Generate** button next to the **Include Directive(s)** field to generate the include paths for the project.

5. Select the **Build Structure** node.
6. Press the **Generate** button.

SNiFF+ enters the names of the project's subprojects in 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.
7. Press **Ok** to save the changes to the project attributes.

A dialog appears, in which you are asked whether Makefiles should be updated.
8. Press **No**. You will update Makefiles in the next sections.
9. In the Launch Pad, save the changes made to the Project Description Files of `Fortran.shared` and its subprojects.

Note

No object files are created in subproject `commons.shared`. As a result, there is no need to set up Make Support for the project. Note that SNiFF+ automatically sets up most of the relevant Make Support attributes of a project during its creation. These default values (and all other project-specific default values) can be set in your Preferences.

Building the project target

You are now ready to build the executable. The steps outlined below are to be executed in the Project Editor.

1. Make sure that all the projects in the Project Tree are checkmarked. If they are not, right-click anywhere in the Project Tree and choose **Context menu > Select from All Projects**.
2. Choose **Target > Update Makefiles** to generate the Make Support Files for all the projects.
3. Choose the **Target > Make > Make all** to build all the object files and targets in the shared project.

A Shell opens, in which the `make all` command is executed. Upon completion, you should have an executable named `evclid` in:

```
$SNiFF_DIR/example/fortran
```
4. Run the executable if you want. To do so, enter `evclid` in the Shell, or choose **Target > Run evclid**.
5. Close the `Fortran.shared` project.

This concludes the tutorial on browsing Fortran 77 code. The next and last tutorial in this Guide covers SNiFF+'s Fortran 90 browsing features.

Browsing a Fortran 90 Project

Goals of this chapter

In this chapter you will learn how to browse Fortran 90 extensions made to the Fortran 77 standard.

Creating a single-user project

(For an explanation of the steps below, please refer to [Creating a Single-User Project — page 7.](#))

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

In the Project Setup Wizard

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

In the “Select development task” page

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

In the “Your development organization” page

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

In the “Select file types” page

- Select **Fortran 90** and press **Next**.

In the “Specify Private Working Environment” page

You are asked to specify your *Private Working Environment* (PWE) root directory, which is the directory that contains your source code.

1. Press **Browse**, and in the Directory dialog, navigate to the root directory of the example code, which is:
`<sniff_installation_dir>/example/fortran90`
2. Double-click on `fortran90` and then press **Select**.
3. In the **PWE name** field, type a name for the PWE, e.g., `Fortran90`.
4. Press **Next**.

In the “Create New SNIFF+ Project” page

- Select the **Use SNIFF+’s Makefiles** checkbox and press **Next**.

In the “Project Setup Summary” page

- Press **Finish**.

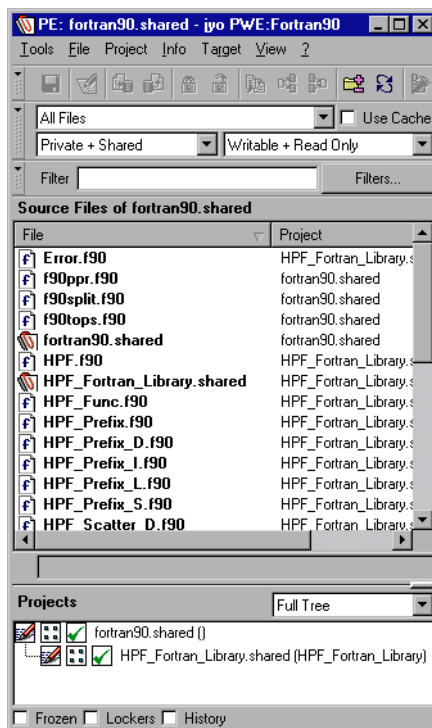
SNIFF+ will now generate the single-user project and its associated files.

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

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

When the generation process is over, SNIFF+ automatically opens the new project and displays its project structure in a Project Editor.

- Make sure that `fortran90.shared` and its subproject `HPF_Fortran_Library.shared` are checkmarked in the Project Tree. If not, checkmark both of them. The Project Editor on your screen should look like this:



Browsing modules

1. In the Launch Pad, choose **Tools > Symbol Browser**.

The Symbol Browser appears.

2. Checkmark both projects in the Symbol Browser's Project Tree.
3. Choose the various entries in the **Symbols** drop-down menu. By selecting the **Signature** check box, you can see in which files the symbols appear. Also, information about data types, modules, dummy arguments (called parameters in other languages) and return types will be displayed.

For a description of these (and additional) Fortran 90 language constructs that the Fortran Parser can parse, refer to [Fortran 90 entries in the Symbol Browser's Type drop-down — page 41](#).

4. Notice that there are no common variables in the `Fortran90.shared` project. However, the project contains modules, derived types and named DO statements.

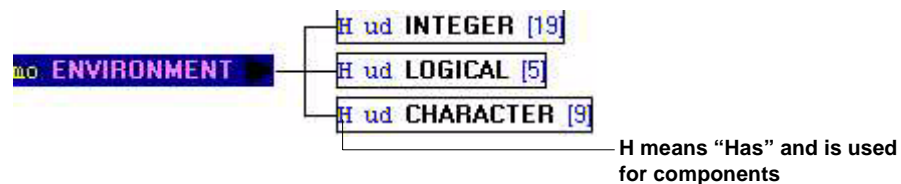
(To differentiate between modules and derived types, select the **Signature** check box and from the **Symbols** drop-down menu, choose **module**.)

5. Select the module `ENVIRONMENT` and double-click it to jump to its definition in the source code.

The Source Editor appears and is positioned to the definition of `ENVIRONMENT` (in line 48 of file `pp.f90`).

6. Let's look at the components that `ENVIRONMENT` refers to. Choose **Info > ENVIRONMENT Refers-To Components**.

A Cross Referencer appears. In the Graph view, you should see the different types of components that are referred to in `ENVIRONMENT`.

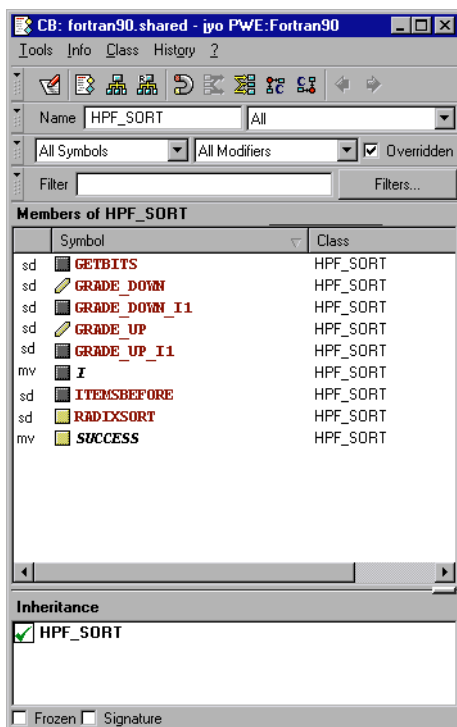


7. Let's now browse another module in the project. In the Symbol Browser, select the module `HPF_SORT` and double-click it to jump to its definition in the source code.

The Source Editor appears and is positioned to the definition of `HPF_SORT` (in line 1 of file `Sort.f90`). This module contains variables, subprograms and generic functions. Some of the symbols in the module have restricted read/write access, as you can see from the keyword `private`.

8. Let's take a closer look at the members of module HPF_SORT. In the Source Editor, choose **Class > Browse HPF_SORT**.

The Class Browser appears and displays the members of the module. The Class Browser lets you browse modules and derived types. It provides a wide range of filtering possibilities based on the inheritance, visibility and type of components.



9. Try out the various drop-downs and buttons. If you want, select and double-click a member to look at its definition in module HPF_SORT.

Fortran 90 entries in the Symbol Browser's Type drop-down

module

When you select this symbol type, you can see all modules and derived types defined in your project. If the **Signature** button is enabled, you can differ between modules and derived types.

Modules and derived types can be analyzed more closely in the Class Browser. You can also do component analyzing with these two symbol types. You can also perform interface cross referencing with modules.

module variable

When you select this symbol type, a list of all variables and parameters defined in modules and of all components of derived types is displayed. To see their types, enable the **Signature** button. The name of the enclosing module or derived type is then prefixed to the variable, parameter or component name, separated by a double colon ("::"). When the **Signature** button is disabled, the name of the module or derived type is postfixed, separated by whitespaces.

module subprogram

When you select this symbol type, all subprograms defined at top level inside of modules are listed. This includes not only functions and subroutines, but also generic functions. Whenever an interface defined in a module is given a name, this name is handled as a module subprogram symbol. As a result, it is possible to resolve the correct references in the Cross Referencer when a generic function is invoked.

Just like module variables, module subprograms can also be prefixed with their module scope. Furthermore, the same conventions that apply to subprograms (with respect to return values and dummy argument types) also apply to module subprograms.

named control statements

Lists all named control statements (IF, CASE, DO) defined in your project.

Browsing derived types

1. In the Symbol Browser, from the **Symbols** drop-down, choose **derived type**.
2. In the Symbol List, select the derived type `ROMAN` and then double-click it to jump to its definition.

The Source Editor appears and is positioned to the definition of `ROMAN` (in line 7 of file `roman_numerals.f90`).
3. Notice that `ROMAN` is a derived type of the module `roman_numerals`.
4. If you want, browse this derived type further in the Class Browser and Cross Referencer.
 - To see which subprograms refer to `ROMAN`, choose **Info > ROMAN Referred-By**.
 - To see which components are referred to by `ROMAN`, choose **Info > ROMAN Refers-To Components**.
 - To look at the members of `ROMAN`, choose **Class > Browse ROMAN**.

Browsing named DO statements

1. In the Symbol Browser, from the **Symbols** drop-down, choose **named DO statement**.

All named DO statement defined in the project are now displayed in the Symbol List.
2. In the Symbol List, select `BSEARCH` and double-click it to jump to its definition in the source code.

The Source Editor appears and is positioned to the definition of `BSEARCH` (in line 3072 of file `iso1539-2_mod.f90`).
3. Look at line 3074 of the current file. Notice that this line contains the exit statement `EXIT BSEARCH`. Let's jump from this statement to the definition of `BSEARCH`.
4. Position the cursor in line 3074 in `BSEARCH` or select `BSEARCH` by double-clicking it. Then, choose **Show > Symbol(s) BSEARCH....**

The cursor is once again positioned to definition of `BSEARCH` in line 3072. As you can see, using the **Show > Symbol(s)** command is a quick way to jump to a named DO statement's definition.

Conclusion

This concludes the tutorial on browsing Fortran 90 code. This also concludes the manual. For a detailed explanation of the various features in SNIFF+, please refer to the *User's Guide* and *Reference Guide*.

- In the Launch Pad, close the `Fortran90.shared` project. Then, quit SNIFF+.

Colophon

This manual was produced with FrameMaker.

We at TakeFive have tried to make the information contained in this manual as accurate as possible. We cannot, however, guarantee that it is error-free.

© 1992-1999 TakeFive Software GmbH.
All rights reserved.



sniff \ˈsnɪf\ *vb* -ED/-ING/-S

[ME *sniffen*; prob. akin to ME *snivelen* to snivel]

vt (14c)

3: to recognize or detect by or as if by smelling
<German shepherd dogs are parachuted in the
Austrian Alps to *sniff* out survivors of avalanches
— P.T.White>

Webster's Unabridged Third New International Dictionary

