Whisper Programmer Studio 3

User Guide

The powerful integrated visual development environment for server based programming.

Third Edition, June 2000

Whisper Technology Limited
25-29 High Street
Leatherhead
Surrey KT22 8AB
United Kingdom

Contents

1 Installing the Software

Programmer Studio presents a brand new way to develop enterprise-wide applications from a single workstation. It combines the ease of use and speed of a standard Windows based visual development environment with the power and flexibility of native server operating systems.

Using Programmer Studio you can construct a model of your development environment using a project hierarchy, edit source files and compile within a single environment.

Installing Programmer Studio

Programmer Studio is essentially a client/server development environment, requiring server-based software to provide access to files and a command line.

The installation program is largely self-explanatory. Simply follow the instructions presented by the installation program.

The Programmer Studio CD-ROM is AutoPlay enabled. If your system supports AutoPlay, simply insert the CD-ROM and the set-up browser will be launched automatically.

If your system does not have AutoPlay, follow these instructions:

To install Programmer Studio

1. Insert the Programmer Studio CD-ROM
2. Open the My Computer folder from the desktop
3. Click on the CD-ROM icon and select Open from the File menu
4. Launch SETUP.EXE to start the installation program

Note To install Programmer Studio from diskette, follow the instructions above, selecting the floppy drive icon instead of the CD-ROM icon in step 3.

Installing the server software

Full instructions on installing the server software can be found in the online help documentation that is installed with Programmer Studio.

2 Introducing Programmer Studio

Welcome to Programmer Studio – the powerful integrated visual development environment for server based programming. Using Programmer Studio, you can structure, edit, and compile your programs without leaving Windows.

Programmer Studio Concept

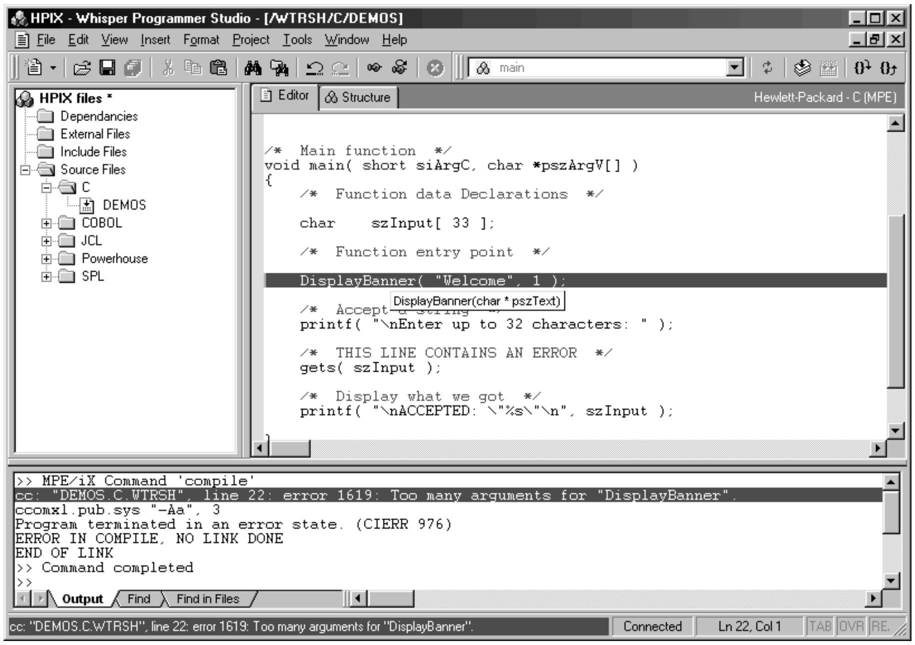
Programmer Studio introduces a new concept in developing server-based applications. Using the popular Windows environment, Programmer Studio provides the ability to edit files and compile programs from a familiar integrated development environment.

Programmer Studio uses the familiar two-tier client/server paradigm, a Windows based development environment (IDE) and server based file transfer and command execution; together providing an powerful integrated visual development environment for server based programming.

Integrated Visual Development Environment

Programmer Studio has been designed to include many of the familiar features found in common windows based development tools. The user interface uses the standard controls found in windows applications to reduce the learning curve for those programmers new to Windows based development.

This section describes the different elements of the programming environment. While not all immediately visible, these components are the basis for Programmer Studio.



```
/*  Main function  */
void main( short siArgC, char *pszArgV[] )
{
    /*  Function data Declarations  */

    char    szInput[ 33 ];

    /*  Function entry point  */

    DisplayBanner( "Welcome", 1 );

    /*  Accept a string  */
    printf( "\nEnter up to 32 characters: " );

    /*  THIS LINE CONTAINS AN ERROR  */
    gets( szInput );

    /*  Display what we got  */
    printf( "\nACCEPTED: \"%s\"\n", szInput );
```

```
>> MPE/iX Command 'compile'
cc: "DEMOS.C.WTRSH", line 22: error 1619: Too many arguments for "DisplayBanner".
ccomxl.pub.sys "-Aa", 3
Program terminated in an error state. (CIERR 976)
ERROR IN COMPILE, NO LINK DONE
END OF LINK
>> Command completed
>>
```

```
cc: "DEMOS.C.WTRSH", line 22: error 1619: Too many arguments for "DisplayBanner".
```

The Programmer Studio visual development environment

Project Workspace

The project workspace window provides the focal point for development using Programmer Studio. You can structure the project to reflect the organization of your existing files and directories, to identify specific components, or a combination of the two.

Code Editor

The Code Editor is a fully featured editor where you will be spending most of your development time. The Code Editor provides many powerful features such as color syntax highlighting, auto-indenting, virtually unlimited undo/redo and numbered line support for languages such as COBOL.

Output Window

The output window provides real-time output from server based compilers offering the ability to cancel at any time. Using Compiler Output Masks, the output can then be parsed allowing the user to examine the location of any errors following the compile.

Command Line

The Command Line provides limited access to the server command line. It is not a terminal emulator and is intended solely for starting programs that do not require further input and only generate simple output.

Project Settings

The Project Settings dialog is used to set the properties of each component of your project. This includes the connection details, file names, locations and compiler commands.

Visual File Compare

Visual File Compare provides a conduit to external command line comparison utilities displaying the results as a split-screen view of both files.

Information for Experienced Windows Users

If you are an experienced user of Microsoft Windows, you will find that Programmer Studio conforms to many of the text editor standards found in other Windows based development environments.

- ♦ Project oriented development model and environment
- ♦ Color syntax highlighting and ToolTips
- ♦ “What’s This?” context sensitive help in all dialog boxes
- ♦ Context menu for most visual items.
- ♦ Standard Properties command for most visual items.

3 Getting Started

The concept of project based program development is central to Programmer Studio's visual development environment. Each project determines how to connect to the remote server, the files, and commands to compile the program.

To help you get started, Programmer Studio has a project wizard that will create a new project from the answers to a few simple questions. The wizard is intended for use by new and experienced users alike.

Using the Project Wizard

Before creating a new project, check that the server software is running and that you have details of the server address (DNS name or IP address), port number, and login. Your system administrator will be able to confirm this information for you.

This chapter will describe in detail each step in the project wizard, from selecting a programming language through connecting to the server and finally adding some files to your new project.

The project wizard is simply the first step in creating a project. All the options selected using the wizard can be easily changed within the project settings.

To create a new project using the wizard

1 From the File menu, select New Project.

2 Determine the programming language(s) of the files that will be included in the project.

3 From the list, check each compiler that will be used with the project.

4 Enter the DNS name or IP address and port number or the server, and then click Next.

5 Enter your user name and password, and then click Next.

6 Enter the project root folder. This is the location in which the majority of your project files are located, and then click Next.

7 Click Add and Remove to determine which files to initially add to the project.

8 Click Finish to create the new project.

Note
MPE/iX

When connecting to MPE/iX servers, the project wizard will prompt for UDC and HPFS support. If you are unfamiliar with either of these terms please contact your system administrator.

4
Code Editor Templates

Programmer Studio supports many of the popular programming languages including Basic, C, C++, COBOL, Fortran, Pascal, etc, etc. Each language has its own specific requirements for content editing, syntax highlighting, line numbers and formatting.

To support these many variations, Programmer Studio uses Code Editor Templates to identify the options that are required on a language-by-language basis.

Introducing Code Editor Templates

Using Code Editor Templates, it is possible to define the settings to be used when editing files for each specific programming language or task. The settings provided by the templates are: -

- Custom editor settings, smart indenting, tab size, tabs to spaces, etc
- Support for languages using line numbers
- Language syntax definition and keyword lists
- Code Navigator support and context sensitive help
- File format options, removing trailing spaces, line lengths, COBOL Tokens, etc
- Specific support for MPE/iX files

As you can see from the number of different options, it is very important to determine which Code Editor Template you wish to use BEFORE you begin to edit your files.

Supported Programming Languages

Programmer Studio comes with a number of standard templates for C, COBOL, Java, Pascal, Basic, etc, etc. These templates are intended to provide the most common options for each language allowing you to get started without having to create your own personal templates.

In many cases the standard templates are configured to work with the most common compilers given the platform and language. For example, the ANSI – COBOL template has the option to "Insert tabs as spaces" turned on, because many of the COBOL compilers in use today do not support the use of the tab character for formatting.

Code Navigator

Code Templates also determine which Code Navigator to use to provide a virtual 'map' of the file being edited. The Code Navigator is discussed in greater detail in Chapter 7.

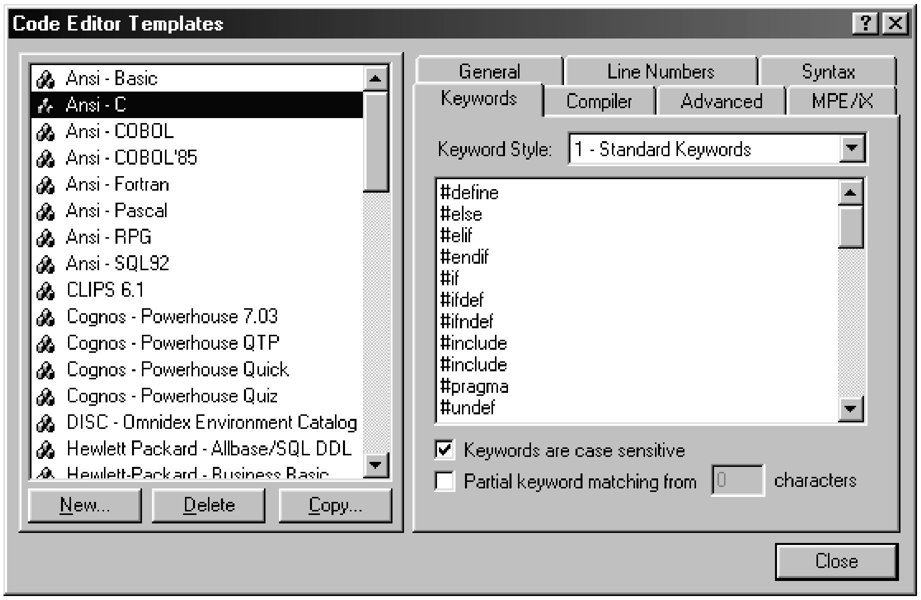
Template Manager

Programmer Studio provides a Template Manager to allow viewing and editing of the many different templates that are currently installed. It is recommended that you examine the settings for each Code Editor Template you intend to use **before** you begin editing.

To view the Code Editor settings for a specific template

1 From the Tools menu, select Code Editor Templates

2 From the Available Templates list, select the specific template


Code Editor Templates

- Ansi - Basic
- Ansi - C
- Ansi - COBOL
- Ansi - COBOL'85
- Ansi - Fortran
- Ansi - Pascal
- Ansi - RPG
- Ansi - SQL92
- CLIPS 6.1
- Cognos - Powerhouse 7.03
- Cognos - Powerhouse QTP
- Cognos - Powerhouse Quick
- Cognos - Powerhouse Quiz
- DISC - Omnidex Environment Catalog
- Hewlett Packard - Allbase/SQL DDL
- Hewlett-Packard - Business Basic

Keyword Style: 1 - Standard Keywords

```
#define
#else
#elif
#endif
#if
#ifdef
#ifndef
#include
#include
#pragma
#undef
```

[✓] Keywords are case sensitive
[ ] Partial keyword matching from 0 characters

The Template Manager displaying the Keyword properties for the ANSI C template

The Template Manager is divided into two parts, a list of available templates to the left, and the settings for the currently selected template to the right. As the selection in the Available Templates list changes, the contents of the tabs to the right will be updated.

The template settings are grouped into 6 sections: -

General

The Editor tab provides options to override the standard Code Editor options found in the Options dialog box. These include tab size and formatting, smart indenting etc.

Line Numbers

The Line Numbers tab provides support for files containing line numbers. These features include automatic and intelligent renumbering, sequence, alignment, etc.

Syntax

The Syntax tab defines the syntax of the programming language, non-keyword characters, quoted string support, and comment styles.

Keywords

The Keywords tab lists the programming language keywords. The list is divided into four styles, which, using the Display options, are painted in different colors. Also included, is the option to determine if the language is case sensitive or not.

Compiler

The Compiler tab defines how third-party language compilers are supported within the Programmer Studio development environment. Options include compiler error detection and pre-compile headings.

Advanced

The Advanced tab defines Code Navigator support, case sensitive help, and advanced file content formatting options.

MPE/iX

The MPE/iX tab provides details for the format of new MPE/iX files, including Robelle Qedit files.

Note MPE/iX	When opening files using the '*(use rules wizard)*' Code Editor Template, the MPE/iX properties can be used to identify a template when no file extension is present.

The Template Manager will only allow one template to be modified at a time. If a change has been made to the properties of the selected template, you will be prompted to save any changes made when selecting another template, or on clicking the Close button.

If any changes have been made to the standard templates, (those installed with Programmer Studio), the user will be prompted to save the changes as a new template.

The remainder of this chapter discusses the areas of consideration when choosing the right template. It does not offer instructions on how to modify the many options available, as these are evident when using the Template Manager.

Custom Editor Settings

When switching between programming languages, or more correctly language compilers, specific limitations can be imposed on the content of files and how they are edited.

Code Editor templates provide the option to override the standard Editor preferences found in the options dialog. This is particularly useful for many of the legacy compilers, which do not support Tab characters or languages imposing a fixed form to code structure, such as COBOL.

Substituting Tabs with Spaces

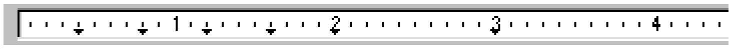
An alternative to standard tab stops is to insert a variable number of spaces in order to simulate tab position formatting. This has a number of disadvantages, including increased file size. However it is necessary in certain applications, notably when using compilers that do not support the tab character.

Defining Tab Stop Positions

Once the option to replace tabs with spaces has been selected, the option to define specific tab positions becomes available. Specific tab positions are very useful in fixed form programming languages like COBOL, imposing rules on the position of specific elements of a program.

The tab positions are specified in order from left to right, in actual character positions separated by a comma. The character positions are relative to the Code Editor ruler, so they are always in ascending order. Once the last defined tab position is reached, the standard tab size comes into effect.

For example, to define a series of tab stops every 4 characters and at 20 and 30, the tab stop positions would be 4,8,12,16,20,30. The Code Editor ruler displays set tab positions as inverted arrows, providing a guide to the current settings.



The Code Editor ruler displaying tab stop positions

Files Containing Line Numbers

For those programming languages that include a number on each line, Programmer Studio provides increased productivity. Using Code Editor Templates, you can remove line numbers while editing, restoring them when the file is saved.

Code Editor Templates provide two options for line number support; automatic renumbering for instances where the existing line numbers are unimportant, and intelligent numbering for instances where the existing line numbers have to be preserved.

Automatic Renumbering

Automatic renumbering examines the first few lines of a file when it is opened. The line numbers appearing on these lines are then used to determine the starting line number and the sequence for the remaining lines. When the file is saved, the existing line numbers are replaced with the automatic line number sequence.

Using the Hide Line Numbers option in the Code Editor Template, it is possible to remove any indication that a file has line numbers, perfect for programmers new to line numbers and those wanting to forget the limitations line numbers impose.

Intelligent Renumbering

Intelligent renumbering saves the numbers assigned to each line as the file is loaded. When a new line is inserted, the Code Editor will assign a number to the new line as long as there is sufficient space in the existing sequence of lines. When no more lines can be added, the user is prompted to renumber the current section or entire file to allow new lines to be inserted.

This option is intended for those users familiar with the line numbers appearing in a given file, and who wish to maintain the line numbers while editing.

Line Number Position

Programmer Studio supports line numbering to both the left and right. For files with line numbers appearing to the right of each line, the character position of the line number is required. For files with line numbers to the left, the position should be set to 1.

Note	Remember when specifying a line number position greater than 1, that the file will have trailing spaces when the line numbers have been removed. Remember to set the 'Remove Trailing Spaces' option.

COBOL Line Tagging

In addition to lines numbers, the COBOL'85 language specification includes a tag field at character positions 73-80 (8 characters). The tag field is free format text, commonly used to store modification dates, programmer initials, version numbers, etc.

Each Code Editor Template has an option to remove these 'tags' while editing, and like intelligent line renumbering, to restore them when saving. If the tokens are removed during editing, you can specify your own token that will be applied to modified lines. These are added to the file when saving.

Programmer Studio provides two options for defining the tag used in COBOL files, either as a standard tag string applied to all files, or, using an advanced property found in project settings for individual files.

Defining a Standard Tag String

For files using a Code Editor Template that has line tagging enabled, the standard tag string is used. This can be overridden using a project files property. Remember the tag string is limited to a maximum of 8 characters. If the tag string is longer than 8 characters it will be truncated.

To define the default tag string

1 From the Tools menu, select Options.

2 In the Line Tag Format box, enter the default tag string.

3 Click OK.

Note	Changing the default tag string will replace unsaved tags in all open files.

Defining a Project File Tag String

Using the tag string property in the Project settings allows more control over the line tags that are applied to the files being edited. When using line tags to record version numbers, being able to specify tags on a file-by-file basis can be very useful.

To define a tag string for a project file

1. Select a file from the project in the tree view.
2. From the View menu, select Properties.
3. Select the Advanced tab.
4. Select the Override standard COBOL line tag button.
5. In the COBOL tag format box enter the desired tag string.
6. Click OK.

Tip Using the Shift and Ctrl keys you can extend the selection in the tree to include more than one file. Any Tag Format will then be applied to all those files highlighted.

Using Date Variables in Tag Strings

A common use for the line tag is to store the date a line was added or last modified. In order to make maintenance of these types of tags as painless as possible, Programmer Studio provides special variables which can be used in tag strings to represent day, month, year and century.

The following variables can be used to represent the date 4th July 1999.

For	Variable	Result
Day	%d	4
Day (two digits space padded)	%+d	<space>4
Day (two digits zero padded)	%dd	04
Month (two digits zero padded)	%mm	07
Short month text (sentence case)	%mmm	Jul
Short month text (upper case)	%MMM	JUL
Year excluding century	%yy	99
Year including century	%yyyy	1999
Robelle Qedit™ formatted date	%qedit_date	4 JUL99
Filename	%filename	sample

Note The length of the tokens cannot exceed 8 characters. Any characters in the token past this point are truncated.

Creating New Code Editor Templates

As you become more familiar with Code Editor templates you will no doubt wish to make changes to the standard templates to, for example, include new keywords, alter compiler output masks, or simply tweak the editor settings.

It is strongly advised that no changes are made to the standard templates, as later releases of Programmer Studio are likely to replace many of these files with more up-to-date versions.

For those users wishing to create or alter the standard templates, Programmer Studio provides User Code Editor Templates. These are identical to the standard template, with the exception that they are stored in an alternative location so there is no chance of losing the files.

Note All Code Editor Template names must be unique. Programmer Studio will generate an error when attempting to use the same template name more than once.

To create a new template

1. From the Tools menu, select Templates
2. Click New

   — Or —

   From the Available Templates list, select the existing template that most closely resembles the new template, click Copy
3. In the Name box, enter the name of the new template
4. Click OK

Note By default, the filename of the new template will be created using the existing template's name. This can be overridden by editing the contents of the filename box.

Using Code Editor Templates

It is important to remember that Code Editor Templates provide options for formatting, in addition to the editor settings like color syntax highlighting. These formatting options, like intelligent line numbering and strip trailing spaces, are used when the file is loaded. This makes it important to select the correct template before loading a file.

The chapters "Working with Files" and "Developing Projects" discuss how to specify the Code Editor Templates used to edit particular files.

5

Working with Files

Programmer Studio attempts to hide many of the complexities of remote, server based development by presenting a familiar interface to remote, server based file systems. By providing dialog boxes that appear and operate in the same way as those found in all Windows applications, any learning curve should be reduced to a minimum.

This chapter describes how to open existing files and create new files. It is assumed that a project has been created and is currently connected to the server.

Opening Files

Using the File Open dialog box you can either open one, or many files from a single directory. Once selected, the files are then transferred in order from the server. The transfer of files can be cancelled at any time by selecting Stop from the View menu, pressing Ctrl+Break, or pressing the Stop button on the toolbar.

Once you have successfully opened a file, the file name will appear at the top of the most recently used (MRU) file list. Using the MRU, you can quickly open the file again by simply selecting the file name from an item in the File menu.

To open a file

1 From the File menu, select Open (CTRL+O).

2 Navigate to and open the folder in which the file is located.

3 In the File name box, enter the name of the file you want to open.

— Or —

From the file list, select the file you want to open.

4 Click Open as read only to stop any changes being made to the file.

5 Click Open to load the file.

Using the file name box, you can quickly change the open folder or limit the files listed by entering and path or wildcard in the File name box and clicking Open (RETURN).

To	Enter
Change the open folder to ‘/usr/dev’	/usr/dev
List only those files beginning ‘de’	de*
List all the files in the open folder	*
Change the open folder and list a subset of files	/usr/dev/de*

Note MPE/iX — MPE/iX users can also use FILENAME.GROUP.ACCOUNT to identify file names, and use the @ symbol for wildcard matches.

Tip — When displaying the open file dialog box, Programmer Studio uses the directory property of the currently selected project folder as the starting location. This can provide a quick way to open files from a directory represented by a project folder.

To open a file with a specific Code Editor template

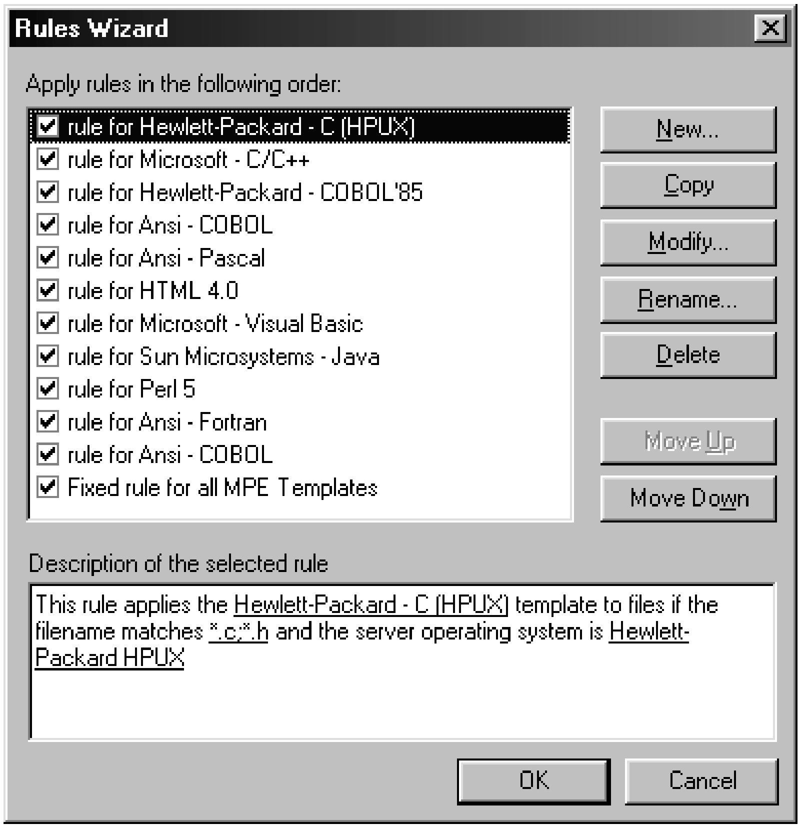
1. From the File menu, select Open (CTRL+O)
2. In the File name box, enter the name of the file you want to open.
3. From the Open As dropdown, select the Code Editor Template to use
4. Click OK

Note — The template selected in the file open dialog box will always override the normal template if the file is included in the current project.

Rules Wizard

Obviously selecting the correct Code Editor Template in the Open dialog box is an awkward addition to the steps required to open a file. Not only is it easy to select the wrong template, (unless you know before hand the content of the file), it can easily be forgotten altogether in the rush to quickly edit a file.

The Rules Wizard is intended to make the process of opening files with the correct Code Editor Template far simpler. By allowing the user to determine which Code Editor Template should be used, based on specific file properties such as filename, server type, file location and line numbers, selecting the right template becomes much easier.



Apply rules in the following order:

- ✓ rule for Hewlett-Packard - C (HPUX)
- ✓ rule for Microsoft - C/C++
- ✓ rule for Hewlett-Packard - COBOL'85
- ✓ rule for Ansi - COBOL
- ✓ rule for Ansi - Pascal
- ✓ rule for HTML 4.0
- ✓ rule for Microsoft - Visual Basic
- ✓ rule for Sun Microsystems - Java
- ✓ rule for Perl 5
- ✓ rule for Ansi - Fortran
- ✓ rule for Ansi - COBOL
- ✓ Fixed rule for all MPE Templates

Description of the selected rule

This rule applies the Hewlett-Packard - C (HPUX) template to files if the filename matches *.c;*.h and the server operating system is Hewlett-Packard HPUX

The Rules Wizard displaying the description of a selected rule

The Rules Wizard is invoked after a file has been successfully opened but before the file has been displayed to the user, this allows the rules to match properties based on the content of the file as well as the filename.

Rules can be easily added and modified using the Rule Wizard which takes you through the process of creating a new rule, determining the conditions that must be met in order to open a specific Code Editor Template.

To add a rule to the Rules Wizard

1 In the Rules Wizard dialog, click New.

2 Determine the Code Editor Template that should be used if every condition of the new rule is satisfied.

3 Click the check box next to each condition the rule must satisfy.

4 For each condition, enter/select the appropriate values.

5 Finally enter a name for the new rule and click Finish.

The following conditions can be matched by the Rule Wizard. It is important to remember that every condition specified must be matched for the rule to be satisfied.

Filename

Specify a filename specification to match. The wildcard tokens '*' and '?' can be used to match sub strings and individual characters respectively. Multiple filename specifications may be entered using a semi-colon as a delimiter.

File Location/Directory

Enter a file specification to match a file's directory. Wildcard tokens and multiple specifications may be entered.

Server Name

Specify the domain name or IP address of the server.

Operating System

Select the operating system of the server from a predefined list.

Line Numbers

Specify the line number position and number of digits in the file.

Qedit File Type

Select the Qedit file type of the file.

MPE File Type

Determine the specific MPE file properties that should be matched.

The new rule will be added to the end of the list of rules in the Rules Wizard. As each rule is evaluated in order, it is important to check that the conditions of a previous rule will not be satisfied before the new rule is checked. Use the rule description to check the preceding rules in the list.

Note The rules wizard can also be used as the Code Editor Template for project files and folders. See Chapter 8, Developing Projects, for more information.

Once a new rule has been added, it is important to check that it will work. This can be easily checked by opening a file that should match the conditions of the new rule. If the Code Editor Template used is not correct, use the file properties to determine which rule was used, return to the Rules Wizard and fix the problem.

To determine which rule was satisfied for an open file

1. With the focus in the code editor, select Properties from the View menu.
2. In the Properties dialog box, select the Advanced tab.

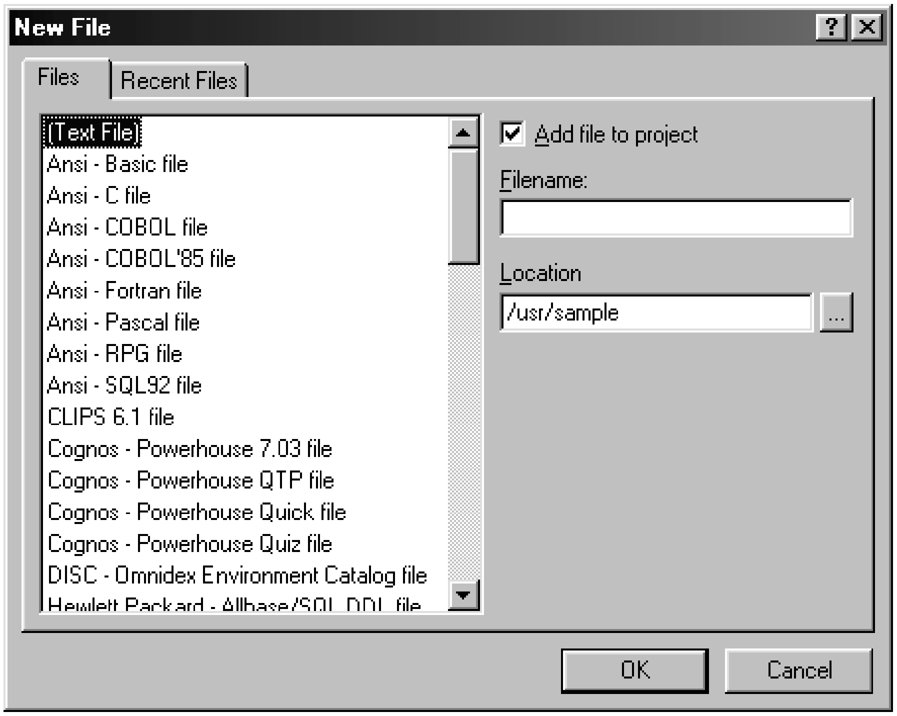
Creating New Files

When creating new files, you first have to determine the Code Editor Template. This determines the format of the new file, whether the file contains line numbers, specific editing characteristics, and how special characters are interpreted.

In this example, a new file is created using the *Normal* Code Editor Template. This has no special format and will use the standard editor settings.

To create a new file

1. From the File menu, select New (CTRL+N).
2. From the list, select Normal, and then click OK.



(Text File)
Ansi - Basic file
Ansi - C file
Ansi - COBOL file
Ansi - COBOL'85 file
Ansi - Fortran file
Ansi - Pascal file
Ansi - RPG file
Ansi - SQL92 file
CLIPS 6.1 file
Cognos - Powerhouse 7.03 file
Cognos - Powerhouse QTP file
Cognos - Powerhouse Quick file
Cognos - Powerhouse Quiz file
DISC - Omnidex Environment Catalog file
Hewlett Packard - Allbase/SQL DDL file

[✓] Add file to project

Filename:

Location: /usr/sample

Note
MPE/iX

Templates also specify specific properties for MPE/iX file types including record length, Qedit® compatibility, file code, type and mode.

Saving Files

Programmer Studio provides all the standard save options found in other Windows applications, including Save, Save As and Save All.

Once you have successfully saved a file, the file name will appear at the top of the most recently used (MRU) file list. Using the MRU, you can quickly open the file again by simply selecting the file name from an item in the File menu.

To save a new file

1 From the File menu, select Save (CTRL+S). The Save As dialog opens.

2 Navigate to and open the folder the file is to be saved into.

3 In the File name box, enter the name for the new file.

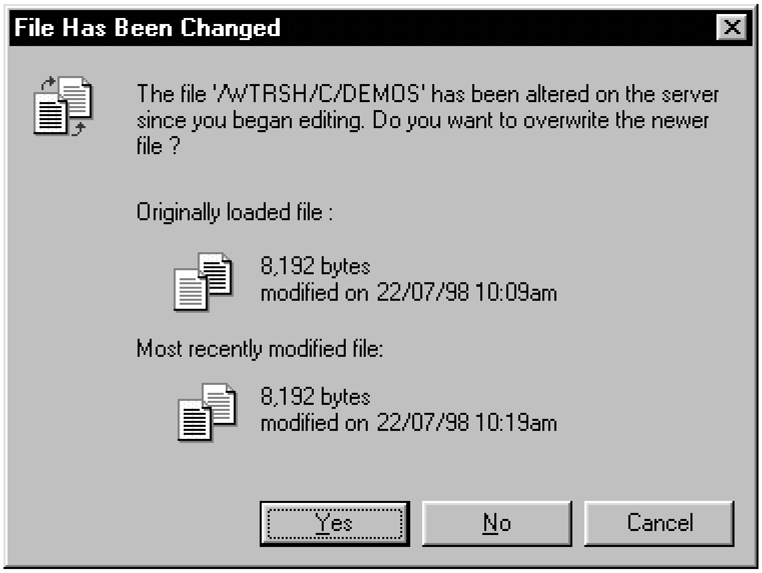
4 Click Save to save the file and close the dialog box.

When you make changes to a file, an asterisk appears in the title bar after the file's name to indicate the file has been modified. This modified tag disappears each time you save the file, appearing again if the file is modified.

To save a file using its original file name

1 From the File menu, select Save (CTRL+S).

Before replacing the existing file, Programmer Studio first checks to ensure the file has not been changed since it was last loaded or saved. If a change has been detected, the user is prompted to overwrite the newer version of the file with details of the differences in size and last modified times.



File Has Been Changed

The file '/WTRSH/C/DEMOS' has been altered on the server since you began editing. Do you want to overwrite the newer file ?

Originally loaded file :

8,192 bytes
modified on 22/07/98 10:09am

Most recently modified file:

8,192 bytes
modified on 22/07/98 10:19am

The file changed warning when saving a file when a newer version exists.

To save a file using a different file name

1. From the File menu, select Save As. The Save As dialog opens.
2. Navigate to and open the folder the file is to be saved into.
3. In the File name box, enter the name for the new file.
4. Click Save to save the file and close the dialog box.

Note If a file already exists with the file name used, you will be prompted to replace the file before saving.

6

Using the Code Editor

Using Programmer Studio you edit code in the same way that you edit text in most Windows based word-processing or development programs.

Scrolling the Editable Area and Moving the Insertion Point

To scroll the editable area to part of the code that does not appear in the Code Editor you use the scroll bars to the right and bottom of the window. The vertical scroll bar indicates the position of the first visible line in relation to the total number of lines in the file.

To position the insertion point, you can use either the mouse or the keyboard. Using the mouse, scroll the editable area using the scroll bars, position the pointer and click the left mouse button. Using the keyboard to position the insertion point will automatically scroll the editable area ensuring the new insertion point is always visible.

To move the insertion point	Press
One character to the right	RIGHT ARROW
One character to the left	LEFT ARROW
To the next line	DOWN ARROW
To the previous line	UP ARROW
To the next non-visible line	PAGE DOWN
To the previous non-visible line	PAGE UP
To the start of the file	CTRL+HOME
To the end of the file	CTRL+END
To the start of the current line	HOME
To the end of the current line	END
One word to the right	CTRL+RIGHT ARROW
One word to the left	CTRL+LEFT ARROW

When using the mouse to determine the insertion point, you may find that the actual point of insertion does not appear where expected. This is because the insertion point can only appear before existing characters, or at the end of a line. For example, clicking the mouse to the right of the last character on a line will position the insertion point after the last character. Clicking the mouse inside a tab character will position the insertion point before or after the tab character, whichever is nearest.

Note	The editable area is always scrolled into view when moving the insertion point or editing the file.

Tip	You can use CTRL+↑ and CTRL+ ↓ to scroll the editable area by one line up and down respectively without moving the insertion point.

Moving the Insertion Point to a Specific Line

As the insertion point is moved within the editable area, the position of the insertion point will appear on the status bar. The status bar displays the insertion point relative to the first line in the file. To scroll a specific line into view, you can use the scroll bar to the right of the editable area, the keyboard, or the Go to command from the Edit menu.

Editing Code

Once you have identified the point at which you want to begin editing and have set the insertion point, you will see a flashing cursor. This is your visual indication of the actual insertion point. The cursor will move as you insert or delete text, continually reflecting the insertion point.

The appearance of the cursor represents the current editing mode. A single vertical line cursor indicates that new text will be inserted at the current insertion point. A block cursor indicates that new text will overwrite text following the insertion point.

Selecting Text

When editing more than one character, you will need to identify a selection. For example, to delete a sentence, create a selection, and then delete it. You can create a selection using either the mouse or the keyboard.

Creating a selection

- ♦ Place the insertion point at the start of the selection, hold the mouse button down, and drag the insertion point to the end of the selection.
- ♦ Place the insertion point at the start of the selection, hold down the SHIFT key, and click the insertion point to the end of the selection.
- ♦ Place the insertion point at the start of the selection, hold down the SHIFT key, and use the cursor keys to move the insertion point to the end of the selection.

Tip You can quickly select an entire word by simply double-clicking on the word. To select all text on the current line press CTRL+L, or click in the left hand margin.

To remove an existing selection, click anywhere in the editable area, or press one of the arrow keys.

To extended or restrict an existing selection, move the insertion point using either the mouse or keyboard while keeping the SHIFT key pressed.

Undoing Mistakes

If you make a mistake in the Code Editor, you can "undo" the last action or command. For example, if you delete a selection, you can restore it. If you then decide you wanted to delete the selection after all you can "redo" it.

As you make changes in the Code Editor, your actions are recorded so that you can "undo" them if required. The number or previous actions that you can "undo", and subsequently "redo", is virtually unlimited based on the amount of available memory.

Moving, Copying and Pasting Text

The following steps show you how to move, copy, and paste text in Programmer Studio using the clipboard. The clipboard is a shared resource that all Windows applications can use to provide a temporary location for storing data to be moved or copied. This allows text to be copied from one window to another, either between Code Editor windows or between Programmer Studio and Notepad.

The instructions below also include the short-cut key combination for the appropriate commands. These appear in brackets after the menu item reference.

To move or copy text using the Clipboard

1. Select the text you want to move or copy.
2. To move the text, select Cut from the Edit menu (CTRL+X)

   — Or —

   To copy the text, select Copy from the Edit menu (CTRL+C)
3. Position the insertion point where you wish to insert the text.
4. From the Edit menu, select Paste (CTRL+V).

Note The Paste operation will automatically delete any existing selection before adding the contents of the clipboard.

Note Programmer Studio also supports legacy Windows editor keystrokes SHIFT+DELETE, CTRL+INSERT, SHIFT+INSERT for cut, copy and paste respectively.

Drag-and-Drop Editing

Drag-and-drop editing is the easiest way to move or copy a limited selection a short distance. However, when moving or copying a large amount of text, Cut, Copy, and Paste, are often more convenient.

To move or copy text using the mouse

1. Select the text you want to move or copy.
2. Point to the selected text, and then hold down the left mouse button.
3. Keeping the mouse button down, drag the insertion point to the new location.
4. To move the text, release the mouse button.

   — Or —

   To copy the text, hold down CTRL while releasing the mouse button.

Visual Elements

Programmer Studio provides a number of visual interface elements to increase programmer productivity, from the text ruler to selected range tool tips.

Text Ruler

The text ruler provides a visual indication of the position of characters relative to the first character of each line. It is important to remember that the toolbar presents only an indication of character position; it does not reflect tab stops.

To display the text ruler

From the View menu, select Ruler

Virtual Line Numbers

As the insertion point is moved in the Code Editor, the status bar displays the number of current line and character position. In some cases however, it can be useful to have the number of each line displayed in the Code Editor.

Virtual line numbers can be displayed to the left of each line displayed in the Code Editor. These start at one and incremented to the end of the file and are identical to the line number appearing in the status bar.

To display virtual line numbers

From the View menu, select Line Numbers, Virtual.

Note Remember not to confuse virtual line numbers with the physical line numbers appearing in a file, especially if the selected Code Editor Template supports files with line numbers such as COBOL.

Selection Margin

The selection margin is displayed to the left of the editable area in the Code Editor providing an easy way to highlight complete lines and view the modified state of each line. The selection margin markers display the modified state of each line, modified, new or newly modified line.

Margin Marker	Indicates
◤	An existing line has been modified
◢	A new line has been added
■	A new line has added and modified

The selection margin can be used to highlight a complete line of text by simply clicking in the margin using the left mouse button. To select multiple lines, click in the selection margin and drag the mouse pointer up or down while keeping the left mouse button down. Release the left mouse button to finish the selection.

To hide or show the selection margin

1 From the Tools menu, select Options

2 Select the Editor tab

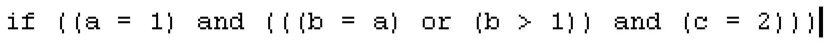
3 Click Selection margin

Bracket and Brace Matching

Bracket and brace matching is especially useful for programming languages that use either brackets ' (' and ') ' or curly braces ' { ' and ' } ' to organize code blocks and expressions. Often typing compound expressions and code blocks can leave the programmer in doubt as to which end bracket or brace corresponds with which starting bracket or brace. Consider the following situation:

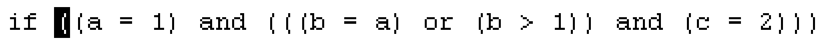
```
if ((a = 1) and (((b = a) or (b > 1)) and (c = 2))
```

Bracket and brace matching allows the programmer to quickly determine if the expression is complete, by highlighting the corresponding open bracket for each close bracket typed. For example, typing a closing bracket at the end of the example expression;



```
if ((a = 1) and (((b = a) or (b > 1)) and (c = 2)))
```

highlights the corresponding opening bracket. See the example below.



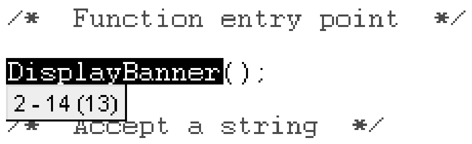
```
if ((a = 1) and (((b = a) or (b > 1)) and (c = 2)))
```

To enable or disable the bracket and brace matching

1. From the Tools menu, select Options
2. Select the Editor tab
3. Click Enable bracket and brace highlight

Selected Range Tool-Tips

For those situations when it is important to determine the range of characters in the current selection, selected range tool-tips can be displayed. The range tool-tips display the actual character position of the first, last and number of inclusive characters in the current selection.



```
/*  Function entry point  */

DisplayBanner();
2 - 14 (13)
/*  Accept a string  */
```

Selected range tool-tips displayed for a single line selection

Note Selected range tool-tips are only displayed for single line selections. Also, tab characters included in a selected range are only counted as single characters.

To enable or disable selected range tool-tips

1 From the Tools menu, select Options

2 Select the Editor tab

3 Click Selected range ToolTips

Finding and Replacing Text

To find and change the text in your code, use the Find and Replace commands on the Edit menu. For example, to change the name of a variable referenced in your code, use Find to locate the text you specify and Replace to change instances of the text.

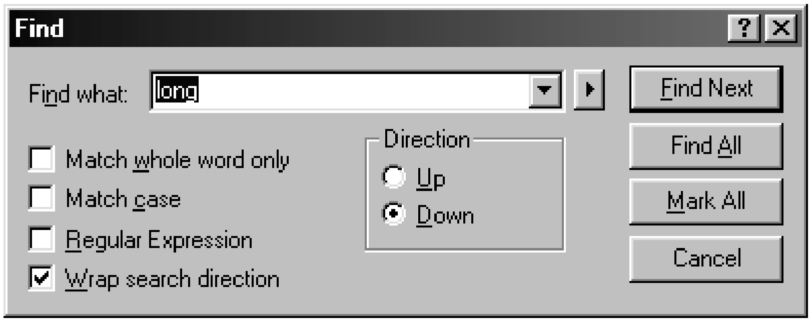
The find and replace dialog boxes are both modeless, this allows the user to switch back to the code editor without closing the find window. Selecting Find or Replace when the dialog box is visible will simply active the window placing the input focus in the relevant field.

Finding Text

From the Edit menu, select Find (CTRL+F). In the Find what box type the text you wish to find. Then choose Find Next to begin the search.

When Programmer Studio finds the text, you have the following options: -

- Choose Find Next to highlight the next occurrence of the find text.
- Choose Find All to list every occurrence of the find text in Find output window.
- Choose Mark All to add a bookmark to each line that contains the find text.



Using the Find dialog box to find the text "long".

To cancel the search and return the input focus to the code editor, press Cancel. To continue the search after having closed the Find dialog, press F3. This will use the previous find text criteria, search options and direction specified.

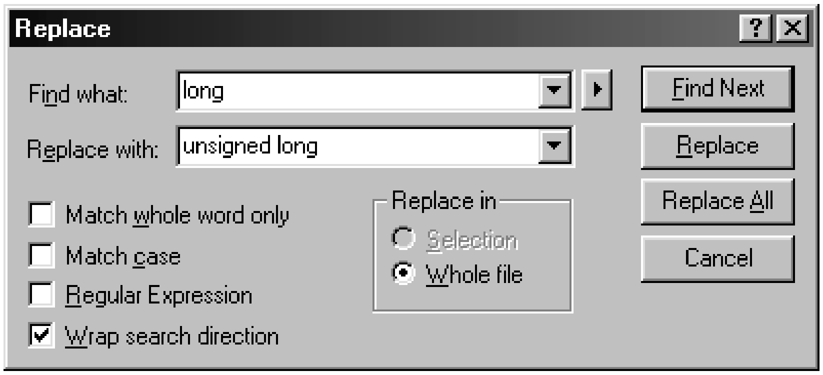
Tip The find and replace dialog boxes use the currently selected text in the code editor as the initial criteria in the Find what box. If there is no selection, the word at the current insertion point is used. Clicking the down arrow to the right of the Find what box displays the previous criteria used.

Replacing Text

From the Edit menu, select Replace (CTRL+H). In the Find what box, type the text you wish to find and the replacement text in the Replace with box. Then choose Find Next to begin the search.

When Programmer Studio finds the text, you have the following options: -

- ♦ Choose Find Next to highlight the next occurrence of the find text.
- ♦ Choose Replace to replace the text and find the next occurrence.
- ♦ Choose Replace All to find and replace every occurrence of the text without prompting the user.



Find what: long
Replace with: unsigned long
[ ] Match whole word only
[ ] Match case
[ ] Regular Expression
[✓] Wrap search direction
Replace in: ( ) Selection [✓ Whole file]

Using the Replace dialog box to replace the test "long" with "unsigned long".

The replace dialog box provides the additional option to replace the text specified within the confines of the current selection, this option is only available if a selection has been created in the code editor.

To undo the effects of the last replacement, select Undo from the Edit menu. This will reverse the last change made. If you confirmed each replacement individually, you will need to undo each change separately.

Search Options

By default, Programmer Studio searches the entire file in the code editor for text matching the criteria specified. The search options can be used to alter how text is matched

To	Select
Find whole words and not sub-strings inside words. For example, *help* and not *helping*.	Match whole word only
Find words matching the case of the text in the Find what box. For example *help* and not *HELP*.	Match case
Continue the search after reaching the start or end of the file (depending upon the direction) to search the entire file.	Wrap search direction

Advanced Search Criteria

Programmer Studio provides a more flexible approach to matching text using regular expressions. Regular expressions can be very simple, such as **f?r**, which finds any three letter word beginning with f and ending in r. Or it can be very complex, with several parts described in parentheses that are individually evaluated.

By combining small expressions to form larger complex (compound) expressions, you can create very specific search criteria. For example, you can find any word beginning with "pre" which ends with "ed", such as "pretended" or "presented".

Regular expressions can be used to search for text that can vary between occurrences. For example, searching for a particular assignment operation which has been typed with varying amounts of white space, is now possible using regular expressions.

To find text using regular expressions

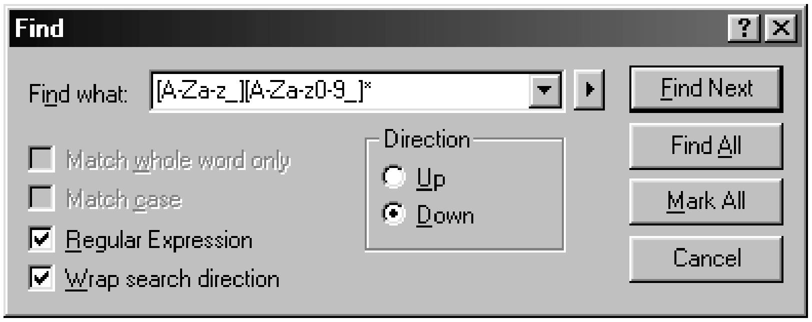
1 From the Edit menu, select Find (CTRL+F)

2 Select the Regular Expression check box

3 In the Find what box, type the search criteria

You can use the right arrow to the right of the Find what box to select from a list of standard regular expressions and predefined compound regular expressions

4 Choose the Find Next button



Find what: [A-Za-z_][A-Za-z0-9_]*

[ ] Match whole word only
[ ] Match case
[✓] Regular Expression
[✓] Wrap search direction

Direction: Up [✓ Down]

Using the find dialog to find any legal variable names in C

Using Regular Expressions

The following section describes the different regular expressions which can be used in the Find what box for the find and replace dialog box.

Simple Expressions

In its most simple form, a simple expression consists of one or more characters. For example,

```
a
if
move
while
```

A sequence of characters is called a character string. This matches the sequence of characters specified to complete the pattern.

Character Classes

A character class is a number of characters enclosed in square brackets. For example,

```
[0123456789]
```

matches any one of the characters inside the brackets. This character class matches any single digit. This digit character class can be written more simply as:

```
[0-9]
```

The '-' indicates a range of characters between the '0' and '9'. The order in which a range of characters is specified is related to the order in which characters appear in the ASCII character set. A complete list of the 7-bit ASCII characters can be found in Appendix B.

Another example of a range within a character class is,

```
[a-z]
```

this would match any lower case character, while

```
[a-zA-Z]
```

would match any uppercase or lowercase character.

If the first character after the '[' is a circumflex (^), the character class matches all characters which are *not* listed in the brackets. For example,

```
[^0-9]
```

would match all characters that are *not* digits.

Note A '-' or ']' as the first character after the '[' is interpreted literally to allow the user to include dashes and square brackets in character classes.

Special Character Classes

The period (.) character class matches any single character except a new line. For example,

```
f.r
```

would match any three character string starting with an 'f' and ending with a 'r'.

Repeating Patterns

Any regular expression, either simple or compound, followed by an asterisk, indicates that zero or more repetitions of the regular expression will be matched. For example,

```
[0-9][0-9]*
```

matches a pattern of characters starting with a digit, followed by zero or more additional digits. This is a typical example of a regular expression used to match a pattern of characters representing a typical number.

```
[A-Z][a-z]*
```

The regular expression above matches an uppercase letter followed by zero or more lowercase letters.

The '+' character can be used in a similar manner to '*' to match one or more repetitions of the preceding regular expression. For example,

```
[0-9]+
```

matches a sequence of one or more digits.

Which is the same as the previous example,

```
[0-9][0-9]*
```

Using the brace brackets you can specify a specific number or repetitions for a regular expression. For example,

```
[0-9]{3}
```

matches a sequence of three digits. A range of repetitions can be defined within the brace brackets. For example,

```
[0-9]{1,10}
```

matches a sequence of 1 to 10 digits.

Optional Expressions

A regular expression immediately followed by a question mark (?) makes the expression optional. For example

```
A?
```

matches 0 or 1 occurrence of the letter 'A'.

Alternatives

Two regular expressions, simple or compound, may be separated by a vertical bar (|) to produce an expression that matches either one of the expressions. For example,

```
[a-z]|[A-Z]
```

matches either a single lowercase or uppercase letter.

Grouping

Using parentheses, expressions may be grouped together. For example,

```
(top|middle|bottom)
```

matches any of the strings top, middle or bottom. Grouping is especially useful when constructing compound regular expressions which use one or more of the expressions described above. For example,

```
([A-Za-z_][A-Za-z0-9_]*)|(-?+?[0-9])
```

matches any C style identifier *or* any integer constant.

Predefined Compound Expressions

Programmer Studio provides a number of predefined compound regular expressions for matching common string patterns. These greatly simplify the task of building a compound regular expression.

To Match	Use
An alphanumeric character	\:a
An alphabetic character	\:c
A control character	\:n
A numeric character	\:d
A graphical character	\:g
A lowercase character	\:l
An uppercase character	\:u
A printable character	\:p
White-space character	\:b
Hexadecimal character	\:h
Any string enclosed in quotes, such as ‘sample’ or “sample”	\:q
Any string enclosed in single quotes, such as ‘sample’	\:QS
Any string enclosed in double quotes, such as "sample"	\:QD

Formatting Text

Programming Studio version 2.0 introduced new formatting capabilities for the Code Editor, included sorting, changing case and inserting text. The following sections describe how each of these new features can be used to greatly improve programmer productivity.

Change Case

A popular requirement among programmers is the need to change the case of text without re-typing. The Change Case dialog box provides 5 different options for changing the case of the currently selected text;

Sentence Case

Converts all characters to lowercase, capitalizing the first character succeeding a period. When the selection does not include a period, the first character in the selection is capitalized.

Lowercase

Converts all characters to lowercase.

Uppercase

Converts all characters to uppercase.

Title Case

Converts all characters to lowercase, capitalizing the first character of each word.

Toggle Case

Reverses the case of the existing characters, converting all lowercase characters to uppercase, and all uppercase characters to lowercase.

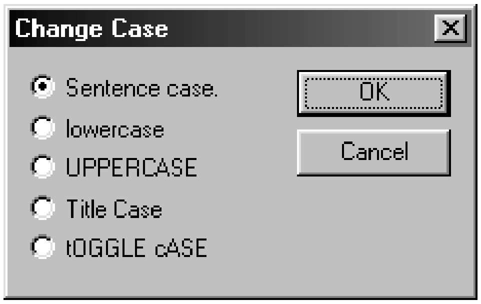
To change the case of text in the Code Editor

1 Select the text you wish to change the case of

2 From the Format menu, select Change Case

3 Select the case style to use

4 Click OK



[✓ Sentence case.] lowercase UPPERCASE Title Case tOGGLE cASE

Changing the case of select text to Sentence case

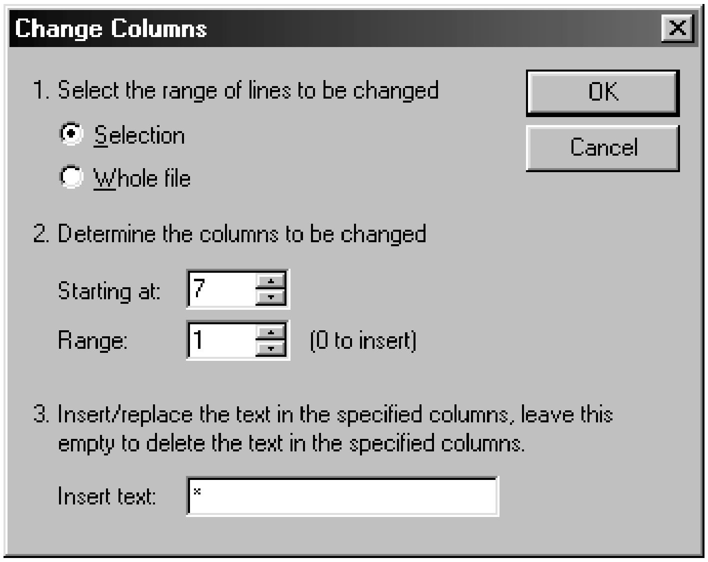
Change Columns

Change columns provides the functionality to insert into or delete from a selection on a column-by-column basis. Using change columns, inserting characters at a specified position, or deleting a range of columns from a fixed format file, can be achieved very quickly.

Change columns is especially useful when working with programming languages which enforce very strict rules on the formatting of source code. In COBOL for example, change columns can be used to quickly modify column 7 (indicator field) to insert a special character for comments or debugging.

To change columns in each line

1. If you want to change a subset of lines, first select these lines.
2. From the Format menu, select Change Columns
3. Click Selection or Whole file to determine the range of line effected
4. In the "Starting at" field, enter the starting column
5. In the "Range" field, enter the number of columns to be changed. To insert text, enter 0.
6. In the "Insert text" field, specify the contents of the columns being changed. To delete the range of columns, leave this field empty.
7. Click OK


Change Columns

1. Select the range of lines to be changed

(● Selection) ○ Whole file

2. Determine the columns to be changed

Starting at: 7

Range: 1 (0 to insert)

3. Insert/replace the text in the specified columns, leave this empty to delete the text in the specified columns.

Insert text: *

Changing column 7 in the selected range of lines to an asterisk

Tip	The *Starting at* field uses the current cursor position as it's initial value.

Note	If a line within the specified range is shorter than the starting column, the line will be padded with the spaces up to the specified column.

Deleting a range of text will shift the text in each line to the left. To clear the columns specified, enter the correct number of spaces in the *Insert text* field. Using change columns, a longer line of text than the size of the range of columns being changed can be specified, so care must be taken when making sweeping changes to a file.

Convert Tabs to Spaces

Although the Code Editor Template provide options to insert tabs as spaces while typing, Programmer Studio also provides the option to convert the tabs in the entire file or a selected range into spaces.

To convert tabs to spaces

1 To convert a block of text, first select the text to convert

2 From the Format menu, select Convert Tabs to Spaces

Comment and Uncomment

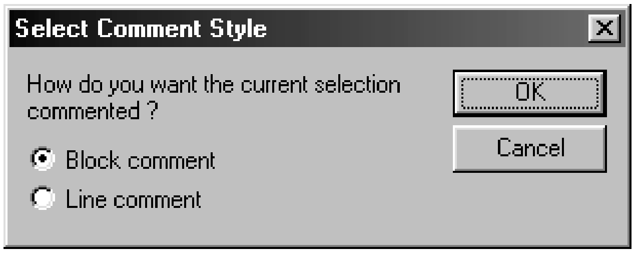
Comment and uncomment use the comment styles defined by the current Code Editor Template to allow the user to quickly comment or uncomment a selected block of text.

Commenting a block of text involves adding the comment identifiers into the selected text. If the Code Editor Template defines both single and multi-line comments the user is prompted to choose which style should be used.

To comment text in the Code Editor

1 Select the text into which you wish to insert comments

2 From the Format menu, select Comment



Select Comment Style

How do you want the current selection commented ?

(● Block comment) ○ Line comment

Selecting the comment style when commenting a block of text

Note A space is inserted after each single-line comment to ensure that an unrecognized keyword is not created from concatenated words. For example, inserting the keyword *COMMENT* at the start of the line *BEGIN PROGRAM* would create an unknown keyword *COMMENTBEGIN* unless an additional space is inserted after *COMMENT*.

Uncomment will delete either multi-line or single-line comments from the currently selected string based on the first comment type encountered from the beginning of the selection. If a space is encountered immediately after the comment keyword, this will also be deleted. Please see the note above for more information.

To uncomment text in the Code Editor

1 Select the text from which you wish to remove comments

2 From the Format menu, select Uncomment

Advanced Features

In addition to the standard editing features described above, Programmer Studio includes a number of more advanced features designed to increase productivity and reduce development time.

Bookmarks

Bookmarks provide the ideal way to mark specific lines in a file when moving around, providing the ability to jump to the next or previous bookmark, or cycle through the bookmarks.

Bookmarks have a toggle state. This means they can be added and removed in exactly the same way. For example, if a line has an existing bookmark toggling the bookmark will remove it, otherwise one will be added.

The following commands are available using the Edit, Bookmarks menu: -

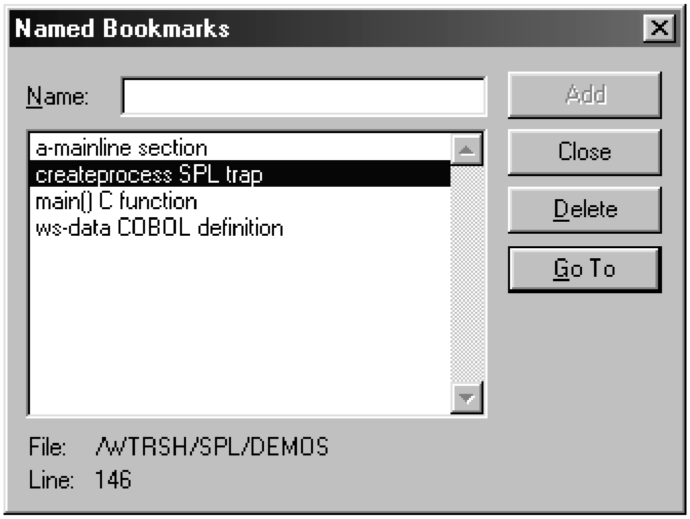
To	Select
Toggle (add or remove) a bookmark	Toggle Bookmark (CTRL+F2)
Move to the next bookmark	Next Bookmark (F2)
Move to the previous bookmark	Previous Bookmark (SHIFT+F2)
Clear all bookmarks in the file	Clear All Bookmarks (CTRL+SHIFT+F2)

Note The direction in which the insertion point is moved when selecting the next or previous bookmark is reversed when reaching the beginning or end of the file.

In addition to manually setting bookmarks, Programmer Studio allows bookmarks to be set on the results of a find. This can be particularly useful when determining what to include in partial find and replace operations. Please refer to Finding and Replacing Text in the previous section for more information.

Named Bookmarks

In addition to normal bookmarks, named bookmarks can be used to mark specific lines in any number of files. When selected, these bookmarks select the appropriate line in the file activating the Code Editor for the file if necessary. If the file in which a named bookmark is located is not currently loaded, Programmer Studio will automatically load the file and select the appropriate line.



a-mainline section
createprocess SPL trap
main() C function
ws-data COBOL definition

File: /WTRSH/SPL/DEMOS
Line: 146

Selecting a named bookmark to highlight in the Code Editor

Unlike standard bookmarks, lines with named bookmarks are not highlighted in the Code Editor.

To create a named bookmark

1 Highlight the line to be bookmarked in the Code Editor

2 From the Edit menu, select Bookmarks, Named Bookmarks (ALT+F2)

3 Enter the name of the new bookmark

4 Click Add to save the named bookmark

Named bookmarks exist only as long as the current project is open. Closing the current project by exiting Programmer Studio, using the Close Project menu item or opening a new project, will delete all named bookmarks.

To move to or delete a named bookmark

1. From the Edit menu, select Bookmarks, Named Bookmarks (ALT+F2)
2. Select the named bookmark from the list
3. Click Go To to move to the highlighted bookmark

   - Or -

   Click Delete to remove the highlighted bookmark

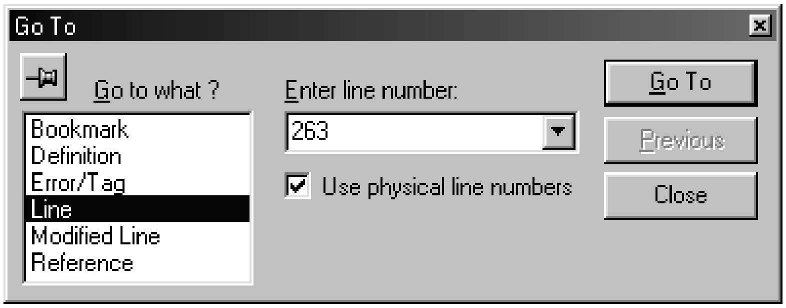
Note The commands to move to the next and previous bookmarks, toggle bookmark and clear all bookmarks do not affect Named Bookmarks.

Moving to Specific Position in the Code Editor

An alternative to using bookmarks or scrolling the editable area to find a line, is to use the "Go to" command. Go to provides a number of options for jumping to a particular line in the file. This line can be identified by virtual or physical line number, bookmark or Code Navigator item.

To go to a specific line

From the View menu, select Go To, Line (CTRL+G)



263

[✓] Use physical line numbers

Using the Go To dialog box with the default criteria

Depending upon the selection in the criteria list to the left, the drop down list to right will change to reflect the options available. Clicking Go To will place the insertion point on the desired line, highlight the line, and move the editable area to ensure the line is visible.

Normally the Go To dialog box would close after clicking the Go To button. If you wish to keep the dialog box open after returning the input focus to the Code Editor, click the pin button in the top left-hand corner.

Tip MPE/iX	MPE/iX style line numbers can also be entered in the line number field. For example, 3.2 would go to line 32000.

Open File Under Cursor

It is not uncommon to find that the files loaded into the Code Editor include references to other files, either in the form of comments, quoted strings, or compiler specific inclusion statements.

Programmer Studio makes it possible to quickly open these files simply by creating a selection and clicking the right mouse button.

To open a file from text string

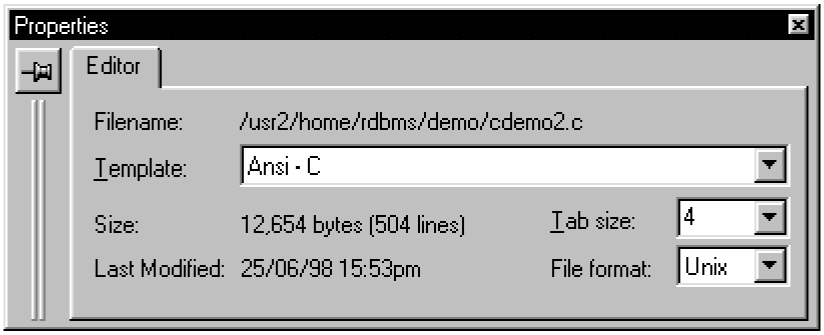
1. Create a selection that includes all the available components of the filename.
2. Right-click inside the selection
3. From the context menu, select Open File…

Programmer Studio uses the current file's location, the login directory, and any server specific dependencies to create a fully qualified file name. If a fully qualified file name can be determined, Programmer Studio will then attempt to open the file.

Note	Programmer Studio will use the same case used in the Code Editor to identify the file name.

File Properties

Once you have opened your file in the Code Editor, Programmer Studio can provide more information on the properties of the specific file, including file size, tab stop size, Code Editor Templates, file format, etc..



Filename: /usr2/home/rdbms/demo/cdemo2.c
Template: Ansi - C
Size: 12,654 bytes (504 lines)
Tab size: 4
Last Modified: 25/06/98 15:53pm
File format: Unix

Viewing the properties of a standard ANSI C source file

Changing Tab Size

Tab stops provide the ideal way to format code to visibly illustrate structure and form. The size of tab stops is set in the Programmer Studio options, however once a file has been opened this can be changed.

To open change the tab stop size in the current file

1 Make sure the insertion point is visible in the current file

2 From the View menu, select Properties (ALT+ENTER)

3 In the Tab Size box enter the new tab stop size

4 Press return

Changing the Format of a File

The format of a file describes the line delimiter used in the current file. When transferring files between the server and PC, it is important to check the format of the file, otherwise other editors may have problems displaying the file.

The following formats are supported

Format	**Line Delimiter**	**Hex**
Unix	Line Feed	0A
DOS	Carriage Return + Line Feed	0D + 0A
Mac	Carriage Return	0D

Changing the Code Editor Template

Code Editor Templates are beyond the scope of this chapter. However for those users familiar with using Templates, the properties dialog provides an option to change the template for the current file.

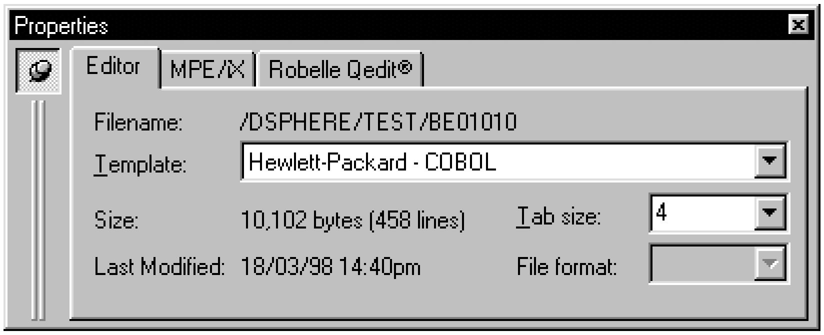
To change the Code Editor Template in the current file

1. Make sure the insertion point is visible in the current file
2. From the View menu, select Properties (ALT+ENTER)
3. From the Template box drop down list, select the Code Editor Template to use
4. Press return

Note Changing the Code Editor Template will only change the editor settings, color syntax highlighting and Code Navigator support. It will not affect the existing file's format or any line numbering support.

MPE/iX and Robelle Qedit Files

When editing files on MPE/iX or native Robelle Qedit files, the properties dialog will add new tabs to allow format specific properties to be set.



Filename: /DSPHERE/TEST/BE01010
Template: Hewlett-Packard - COBOL
Size: 10,102 bytes (458 lines)
Tab size: 4
Last Modified: 18/03/98 14:40pm
File format:

Viewing the properties of a Robelle Qedit file

Printing

This section describes the printing options available from the Code Editor.

Note Programmer Studio does not print any color syntax highlighting, or print any bookmarks or jump locations displayed in the Code Editor.

To print the current file

1 Make sure the insertion point is visible in the current file and there is no selection

2 From the File menu, select Print

3 Determine the number of copies to be printed

4 Click OK

In addition to printing the entire file, Programmer Studio allows a smaller range of text to be printed. This can be especially useful for very large files containing thousands of lines.

To print a range of text

1 Select the text you want to print

2 From the File menu, select Print

3 Determine the number of copies to be printed

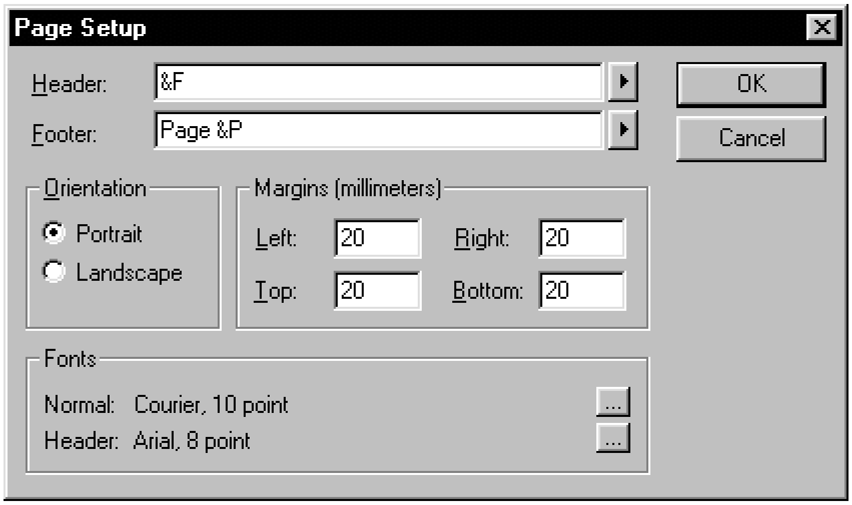
4 Click OK

Additional Print Options

Programmer Studio provides a number of formatting options when printing from the Code Editor, providing control over the printer font, header, footer etc.

To display the additional print options

From the File menu, select Page Set-up


Page Setup

Header: &F

Footer: Page &P

Orientation
(● Portrait) Landscape

Margins (millimeters)
Left: 20 Right: 20
Top: 20 Bottom: 20

Fonts
Normal: Courier, 10 point
Header: Arial, 8 point

Page set-up dialog box displaying default settings

7
The Code Navigator

The Code Navigator is the key to much of the enhanced productivity enjoyed by Programmer Studio users. By taking a source file and compiling a virtual ‘map’ of the contents, Programmer Studio can offer many new options for navigating your code.

This chapter makes references to the procedural elements of a program as *functions*. Whilst this terminology may not be correct for many other programming languages, it is assumed the reader will make the necessary assumptions.

Introducing the Code Navigator

The Code Navigator is effectively an enhancement to the Code Editor. Once a file is loaded, the selected Code Navigator parses the content of the file creating the virtual ‘map’ and updating the specific visual components. This virtual ‘map’ is subsequently updated every time the file is saved.

Code Navigator support is provided on a per-programming language basis by external plug-in modules called Programmer Studio Extensions (PSX’s). These PSX’s are designed and written specifically for each language to create the ‘map’ of the components of a source file. The various Code Navigator tools use this ‘map’.

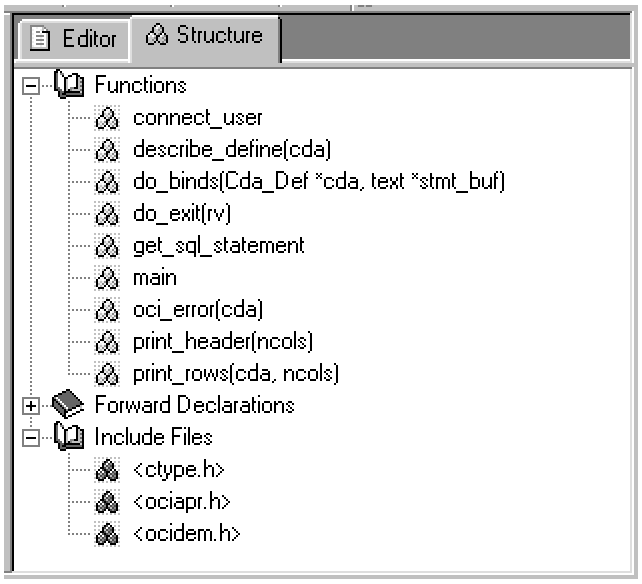
Currently Programmer Studio supports 12 languages. This will grow with later releases dependant upon user request. Any requests for Code Navigator support for additional programming languages should be made to Whisper Technology Technical Support.

Visual Components

The Code Navigator consists of three Programmer Studio components; the Structure view provides a collapsible tree view of the elements of your program; the Navigator Toolbar displays the *function* currently being edited and easy access to other functions, and finally, Navigator Tips can provide definitions of known *functions* while editing.

Structure View

Within the Code Editor, the structure view provides the complete ‘map’ created by the Code Navigator in the form of a collapsible tree view. This tree view can have up to four root level entries, from which the components of a file are organized in alphabetically sorted order.



```
⊟ Functions
    connect_user
    describe_define(cda)
    do_binds(Cda_Def *cda, text *stmt_buf)
    do_exit(rv)
    get_sql_statement
    main
    oci_error(cda)
    print_header(ncols)
    print_rows(cda, ncols)
⊞ Forward Declarations
⊟ Include Files
    <ctype.h>
    <ociapr.h>
    <ocidem.h>
```

Code Navigator structure view of an ANSI C file

Double-clicking on an entry in the tree view will automatically switch to the Code Editor, highlighting the line on which the entry appears.

Tip	Press CTRL+W to quickly switch between the Code Editor and Structure views.

For those programming languages supporting *function* parameters, Programmer Studio provides an option to display the parameter lists in addition to the *function* names within the structure view. This is illustrated in the screen shot above.

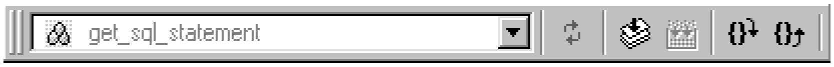
To view *function* parameter lists in the structure view

1. From the Tools menu, select Options
2. Select the Workspace tab
3. Click Display function parameter lists in Code Navigator tree
4. Click OK
5. From the View menu, select Refresh (F5)

Navigator Toolbar

The Code Navigator toolbar uses the contents of the virtual ‘map’ to create a drop-down list of all the *functions* and *sections* identified in the file, which are presented in a structured and alphabetical order.

On selecting an item from the drop-down list, the Code Editor will highlight the line on which the *function* exists and ensure the line is visible. Scrolling through the list, either using the keyboard or mouse, will highlight each *function* as it is selected. To cancel the selection, press escape to return to the original insertion point.



Code Navigator Toolbar view of an ANSI C file

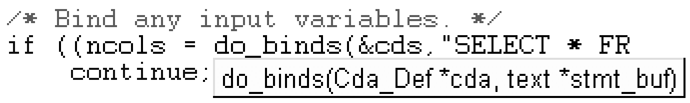
As the insertion point in the Code Editor moves, the Navigator toolbar is updated to reflect the name of the *function* or *section* appearing in the virtual ‘map’ at or before the current line. It is important to remember that the name appearing in the toolbar is based on the last time the Code Navigator parsed the file, either on a save or refresh. Any functions added or removed are not immediately updated.

In addition to the drop-down list, the navigator toolbar also includes buttons to refresh the Code Navigator’s virtual ‘map’ without saving the file, and to jump to the next or previous *function* in the file.

Tip	Press CTRL+Q to quickly activate and drop-down the Code Navigator *function* list.

Navigator Tips

In the majority of programming languages, *functions* definitions can include parameter lists, a series of variables passed into the *function* when it is called. If the parameter definition is included as part of the *function* declaration, the Code Navigator can display this information when the Code Editor detects the user is entering a call to the *function*.



```
/* Bind any input variables. */
if ((ncols = do_binds(&cds,"SELECT * FR
    continue; do_binds(Cda_Def *cda, text *stmt_buf)
```

Code Navigator Tips view of an ANSI C function

To view *function* parameter lists in Navigator Tips

1. From the Tools menu, select Options
2. Select the Workspace tab
3. Click Enable Code Navigator tips when editing
4. Click OK
5. From the View menu, select Refresh (F5)

Note Support for function parameter lists is currently provided for Basic, C, C++, Java and Pascal.

8 Developing Projects

The concept of project based program development is central to Programmer Studio's visual development environment. Each project determines how to connect to the remote server, the files, and commands to compile the program.

Understanding Projects

Projects are more than simply a collection of filenames. The project also stores associated file information, read-only attributes and editing preferences in addition to the commands you will use to compile these files.

Project Folders

Project folders allow the files included in a project to be organized into a hierarchy. This hierarchy can have an unlimited number of levels allowing the user to determine exactly how a project should be organized.

The structure of a project can be easily adapted to illustrate an existing file system structure or simply to logically organize files. When used together, these two organizational approaches provide the most powerful way to structure your project.

The Project Window

The hierarchy of the current project is presented in the project window as a collapsible tree view. This view can be expanded easily to view all the files in a project, or left partially collapsed to identify a specific group of files.

Project Settings

The project settings dialog provides access to the individual properties of the project, files, and folders in your project. Within the project settings you can set properties on each item or as a group.

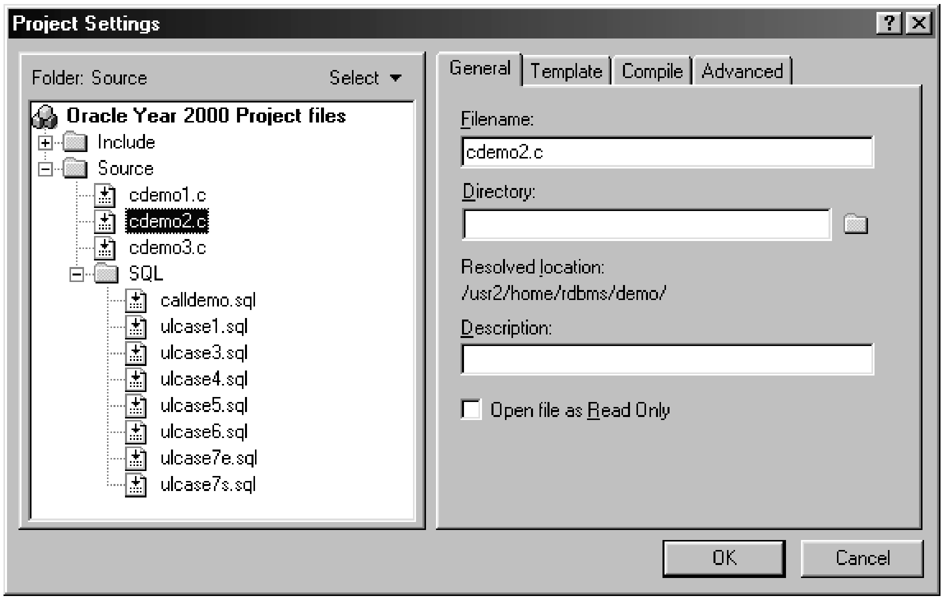
To view the settings for a project item

1 From the project window, select the file, folder or root item

2 From the Project menu, select Settings (ALT+F7)

— Or —

Right click on the selected tree item and select Properties



Filename: cdemo2.c

Directory:

Resolved location: /usr2/home/rdbms/demo/

Description:

☐ Open file as Read Only

The Project Settings dialog displaying the properties of the file cdemo2.c

The settings dialog is divided into two parts. A structured view of your project appears on the left and the properties for the selected item on the right. Depending upon the selection, the tabs to the right will change.

Note	Unlike the structure displayed in the main project window, files and folders cannot be deleted or moved in the Settings dialog.

A project can consist of three types of components; the project itself, project folders, and files. Each component has its own selection of properties and in some cases these are shared between items of different types.

Tip Using the Shift and Ctrl keys you can extend a selection to include a number of components. When this selection includes components of different types only those shared by the selection are displayed.

Use the Select drop down menu to extend the current tree view selection.

Adding, Moving and Removing Files and Folders

When adding files to a project it is important to remember that you are simply adding a reference to the file, a not a copy of the file itself. Therefore any changes made to the file will be reflected in any other projects that include the file.

It is important to understand the structure of a Programmer Studio project before you begin adding files, or indeed creating new project folders. Each level in the project, including the project root, has a Directory or location property. This property is used to determine the location of subordinate files, and the working directory for compiling your files.

Adding Files and Folders to the Project

Programmer Studio allows files and folders to be added at any level in the project hierarchy, from the root upwards. The only limitation is that new items must be added either to the project root or a project folder. Items cannot be added to the project's hierarchy below an existing file.

To add a file to the project

1 From the project window, select the item from the tree to which the new file is to be added

2 From the Project menu, select Add Files to Folder

— Or —

Right click on the selected tree item and select Add Files to Folder

3 Using the file dialog box, select the file to add to the project

4 Click OK

Tip To add multiple files to the project, use the SHIFT and CTRL keys to extend the selection in the file dialog box.

To add a folder to the project

1 From the project window, select the folder or project root to which the new folder is to be created

2 From the Project menu, select New Folder

— Or —

Right click on the selected tree item and select New Folder

3 In the name box, enter the name of the new folder

4 If the new folder represents a directory/location, enter a relative directory path in the directory box.

5 Click OK

The relative directory property for new folders is discussed in greater detail in the section titled Structuring your Project.

Moving and Removing Folders and Files

As each program grows, so will the requirements placed on the project. New files may be added and old ones deleted. New project folders may be added and the existing structure altered to fit. To make these changes as simple as possible, Programmer Studio provides the ability to delete items from the project and move existing files and folders within the hierarchy.

It is not possible to use Undo to reverse any changes made by moving or deleting items from the project.

To move a file or folder

1 From the project window, point to the file or folder to move, and then hold down the left mouse button.

2 Keeping the mouse button down, drag the pointer to the new location.

3 To move the file or folder, release the mouse button.

While moving the mouse, the pointer is updated to reflect the suitability of the item under the cursor as a destination for the move. Positioning the mouse pointer at the top and bottom of the tree view will scroll the contents of the window in the required direction.

Note	When moving files and folders, it is important to remember that the properties of the items selected are not affected. If a file or folder uses relative file locations, these may have been altered.

To delete an existing file or folder

1 From the project window, select the file or folder to delete

2 Press DELETE

— Or —

Right click on the selected tree item and select Remove

Note	Deleting a project folder will delete all files and folders below this entry.

Structuring Your Project

A well organized project is the key to productive development using Programmer Studio. The initial set-up of compile and build options and the easy identification of the many components of a complex program save many hours development time.

Project Hierarchy

A Programmer Studio project consists of a collection of folders and files that exist in a simple hierarchy. A key advantage of a hierarchical project model is the ability to set properties for a folder which will then be inherited by the folders and files which appear as descendants in the hierarchy.

The following example illustrates how a Code Editor Template may be specified at a project level, inherited by the files and folders in the project, and then overridden at either level.

	Code Editor Template	Resolves As
Project	Ansi – C	
test.c		Ansi – C
source		
main.c		Ansi – C
client	Microsoft C/C++	
win.c		Microsoft C/C++
include		
oradb.h	Oracle ProC	Oracle ProC

Although the property inheritance model requires some careful planning, the resulting project should require little modification when adding subsequent files and folders.

Relative vs. Absolute File Locations

Programmer Studio can identify files using either relative or absolute file names. An absolute filename takes the format of a fully qualified location, starting with a '/' to indicate a location from the root directory. Relative file names do not have a fully qualified location. Instead Programmer Studio uses the location property of project folders appearing above the file in the project hierarchy to determine an absolute location.

Take for example, a project containing these two files:

- */usr/dev/a*
- */usr/dev/b*

These files can be identified in a simple project in one of two ways, either as their absolute file name or relative to the project's directory. The two examples are illustrated below:

	Location	Resolves As
Project		
a	/usr/dev/a	/usr/dev/a
b	/usr/dev/b	/usr/dev/b

	Location	Resolves As
Project	**/usr/dev**	
a	a	**/usr/dev**/a
b	b	**/usr/dev**/b

From this simple example, the advantages of using relative file names and folder directories to identify file locations is obvious. Not simply in reducing project maintenance, but also providing an easy way to point the project at an alternative set of files in a different location as below.

	Location	Resolves As
Project	**/usr/copy**	
a	a	**/usr/copy**/a
b	b	**/usr/copy**/b

Note When adding files to the project or a project folder, Programmer Studio will attempt to resolve the location of the files added so that they are relative to the folder to which they are added.

Using Project Folders

To best illustrate the benefits of structuring a project we will start with a simple example; a project that contains 3 files and no project folders. In this example, the project was created with the Directory property of the project set to */dev*, into which the following files were added:-

- */dev/test.c*
- */dev/src/main.c*
- */include/test.h*

When adding files to a project, whether to the root or project folder, Programmer Studio attempts to store the files as relative whenever possible. In this example files added to the project which include */dev* (project Directory) as the start of their location have this removed.

	Location	**Resolves As**
Project	**/dev**	**/dev**
test.c	test.c	**/dev**/test.c
main.c	src/main.c	**/dev**/src/main.c
test.h	/include/test.h	/include/test.h

By storing the names of the project files as relative, Programmer Studio allows the user to change the location of the files. As long as the structure is moved intact, the project will only require the Directory property of the project to be changed.

Note The Resolved Location field in project settings dialog displays how Programmer Studio will interpret relative file and directory names.

In order to reflect the structure of the file existing system, a new folder *source* is added to the project. The Directory property of the new folder is set to *src* and the file */dev/src/main.c* is then moved into the new folder as *main.c*. Using the name resolution rules, the location of main.c is now determined by traversing the tree and building a path, for example:

	Location	Resolves As
Project	/dev	/dev
source	src	/dev/src
main.c	main.c	/dev/src/main.c

The resulting project looks like this:

	Location	Resolves As
Project	/dev	/dev
test.c	test.c	/dev/test.c
source	src	/dev/src
main.c	main.c	/dev/src/main.c
test.h	/include/test.h	/include/test.h

To complete this example we will create a folder for *test.h*. As this is the only project file from an alternate location this is really not necessary, however it will result it a much tidier project. The Directory property of the new folder will be absolute, as the location starts with a slash, and the parent project Directory will be ignored.

	Location	Resolves As
Project	/dev	/dev
test.c	test.c	/dev/test.c
source	src	/dev/src
main.c	main.c	/dev/src/main.c
external	/include	/include
test.h	test.h	/include/test.h

Note In these examples we have purposely limited the project folder Directory to a single server directory for simplification. It is possible to use any directory that complies with standard Unix path resolution rules.

The examples given here illustrate how to structure a project to represent the structure of the existing file system. This results in a well-organized project that is both simple and portable.

9 Compiling Your Files

Programmer Studio provides no specific compiler support for software developers. Instead it offers access to server-based compilers by executing a command on the remote server. This imposes the limitation that the command must not require further user input to reach completion.

Once executed, Programmer Studio traps the output from the specified command, displaying the results in the output window in real-time. You can cancel the command at any time by selecting Stop from the View menu.

Compile and Build Commands

Programmer Studio provides a two-step model for compiling project files into the finished program; compiling individual files and then building a program file from the compiled files.

This model is provided purely as a guide for developers. It can be readily applied to programming languages that compile, and then link a number of individual source files into a single executable program file, like C and C++. For other languages however, this can be easily ignored.

Compile and build commands are specified as project properties. For a file to be compiled or built, it must be included in the current project. Each file in a project can have its own compile command and each project folder its own build command.

Compiling a File

Before a file can be compiled, the project settings for the file need to specify the command to be executed on the server. Programmer Studio imposes no restriction on what command is executed, as it is the responsibility of the developer to ensure the command is valid and can run without further user intervention.

To set the compile command for a project file

1 From the project tree view, select the file

2 From the Project menu, select Settings (ALT+F7)

3 Select the Compile tab

4 Click Override folder compile command

5 In the Compile command box, enter the complete command that is to be executed on the remote server to compile this file.

6 Click OK

Note Remember to include the filename and location in the compile command.

Once a compile command has been assigned to a project file there are two ways to compile the file; either using the file in the project tree or, more simply, choosing compile if the file is open and the insertion point is visible in the Code Editor.

To compile a project file

1 From the project tree view, select the file

2 From the Project menu, select Compile (CTRL+F7)

— Or —

Right click on the selected tree item and select Compile

To compile the currently open file

1 Make sure the insertion point is visible in the Code Editor

2 From the Project menu, select Compile (CTRL+F7)

By default, Programmer Studio will automatically save all the files that have been changed before compiling. This is done to ensure any errors or warnings generated by the compiler can be correctly identified in the Code Editor.

Command Variables

To prevent typing mistakes and save time entering repetitive compile commands, variables can be used in the command text that is evaluated prior to execution. Variables can be used to identify the name of a file, a fully qualified filename, Code Editor Template, etc.

The following table lists the available compile command variables and gives the output for the file **/usr/dev/sample.pas**.

For	Enter	Sample
File path	$(FilePath)	/usr/dev/sample.pas
File name	$(FileName)	sample.pas
File directory	$(FileDir)	/usr/dev
File extension	$(FileExt)	pas
File base name	$(FileBase)	sample
Code Editor Template	$(Template)	Ansi – Pascal
Project name	$(ProjectName)	Sample Project

In addition to the standard compile command variables, MPE/iX compatible file names and locations are also supported. The following table gives the output for **/SYS/PUB/SAMPLE**.

For	Enter	Sample
MPE/iX file path	$(MpeFilePath)	SAMPLE.PUB.SYS
MPE/iX file location	$(MpeFileDir)	PUB.SYS

Tip Clicking the arrow to the right of the Compile command box in the project settings dialog displays a popup menu containing command variables to choose from.

Folder Compile Commands

Even using compile command variables, entering a compile command for every file in a large project is not only likely to be very time consuming, but will probably introduce problems. To simplify this process each project folder also has a compile command that is used for all subordinate files without specific compile commands.

To set the compile command for a project folder

1 From the project tree view, select a folder

2 From the Project menu, select Settings (ALT+F7)

3 Select the Compile tab

4 Click Override folder compile command

5 In the Compile command box, enter the compile command

Tip Clicking the arrow to the right of the Compile command box displays a menu containing command variables

The following two examples illustrate how compiler commands at different levels of the project hierarchy are evaluated.

	Compile Command	Resolves As
Project		
source	cc $(FilePath)	
main.c		cc /dev/src/main.c
test.c		cc /dev/src/test.c
client.c	compile $(FileName)	compile client.c

	Compile Command	Resolves As
Project	cc $(FilePath)	
source		
main.c		cc /dev/src/main.c
source	compile $(FileName)	
test.c		compile test.c

Note The variables used in these examples are purely an indication of sample compiler commands. There are no limits on which variables or how many variables can be used for a compile.

At this stage, new users may find it useful to review the Project Structure section in Chapter 3, as the considerations for project structure and identifying file locations can be combined with folder compile commands.

Build Commands

Build Commands exist at the project folder level, to provide a command which is used in conjunction with compile commands, for those developers needing to implement a two-stage process to create a program file.

Build Commands offer the option to initiate a compile of all the files contained in the folder that can then be followed by the build command.

Compiler Results

The resulting output from a compile command can produce many hundreds of lines, even when successful. Examining every line of output, checking for warnings and errors is not only time consuming, but missing a vital error can lead to many lost hours of precious development time.

The solution to this problem is to allow Programmer Studio to identify possible warnings and errors in the compiler output, highlighting each offending line for closer examination. As Programmer Studio provides no specific compiler support, this detection of errors is achieved using Compiler Output Masks.

Compiler Output Masks

Compiler Output Masks provide Programmer Studio with instructions on how to detect warnings and errors in the compiler output in order to highlight these lines. If the Compiler Output Mask indicates the location of a filename within the output, this file is then opened and the offending line highlighted.

A Compiler Output Mask is constructed from a series of text identifiers, which are matched verbatim, and special variables indicating known compiler components such as line numbers, numeric identifiers and filenames.

The mask can contain any number of literal fields or variables from the following list:-

To match	Use
Any number of consecutive spaces	Space
A single character not including spaces	$(Char)
Any number of consecutive characters not broken by a tab, space or literal character.	$(String)
A numeric value	$(Number)
Text on the next output line	$(LineBreak)
Continue scanning on the current output line in order to complete the mask	$(ScanFor)
Scanning upwards from the current output line in order to complete the mask	$(ScanUp)
The line on which the error can be found	$(Line) *
The number appearing on the line on which the error can be found	$(LineNumber) *
The filename containing the error	$(FileName) *
MPE/iX style filenames – FILE.GROUP.ACCOUNT	$(MpeFile) *
MPE/iX style line numbers including a period	$(MpeLine) *
DOS style filenames – C:\NETLOG.TXT	$(DosFile) *

* Indicates variables that may only be used once within the Compiler Output Mask.

If the Compiler Output Mask identifies a filename that is not absolute, Programmer Studio will attempt to match the filename against the files included in the current project. Where two files are included with identical names, the first file found will be used.

The following examples illustrate how to create a Compiler Output Mask to detect possible errors within the sample compiler output. To help in identification, Compiler Output Mask variables appear in bold.

Example:	`cc: "hello.c", line 3: error 1507: Function...`
Mask:	**`$(String)`**`: `**`$(FileName)`**`, line `**`$(Line)`**`: `**`$(String)`**

Example:	`/usr/examples/hello.c(3) : error...`
Mask:	**`$(FileName)($(Line)) : $(String)`**

Example:	`File: /usr/examples/hello.c`
	`Line 3`

Mask:	**`File: $(FileName)$(LineBreak)Line $(Line)`**

Example:	`Error 523, unknown item at /usr/examples/hello.c line 3`
Mask:	**`$(ScanFor)at $(FileName) line $(Line)`**

Example:	`File: /usr/examples/hello.c`
	`Error...`
	`Line 3`

Mask:	**`Line $(Line)$(ScanUp)File: $(FileName)`**

Using Compiler Output Masks

Programmer Studio comes with a number of compiler output masks for many of the popular compilers on different servers. Code Editor Templates provide the ideal way to organize these compiler output masks by programming language type, allowing the user to simply select from a familiar list of programming language names.

To select the compiler output masks for the project

1. From the project tree view, select the project icon
2. From the Project menu, select Settings (ALT+F7)
3. Select the Output tab
4. From the Compiler Error Detection list, select the programming languages used by the project
5. Click OK

As the output from compilers can differ greatly between platforms, compilers and even compiler versions, Programmer Studio also allows new compiler output masks to be added to a project as required without the need to edit the Code Editor Templates.

To add a custom compiler output mask to the project

1 From the project tree view, select the project icon

2 From the Project menu, select Settings (ALT+F7)

3 Select the Output tab

4 Click the Custom button

5 In the Output mask box, enter the new compiler output mask

6 Click OK

Examining Resulting Errors

Using the Compiler Output Masks, Programmer Studio allows the user to quickly move forward and backward through the resulting output to the next and previous line matching the mask criteria.

As each warning and error is highlighted, Programmer Studio will attempt to load the file containing the error, (if it is not already loaded), and highlight the line on which the error is determined to have occurred.

To highlight the next error matching the Compiler Output Mask

1 Ensure the Output tab is activated in the Output Window

2 From the Edit menu, select Go To, then Next Error/Tag (F4)

In addition, by highlighting the first warning or error in the output window, Programmer Studio provides the option to determine at the end of a compile or build, the total number of lines matching the Compiler Output Mask, and also the option to automatically highlight the first error.

To automatically highlight the first error in the output window

1 From the Tools menu, select Options

2 Select the Build tab

3 Click Highlight the first error found

4 Click OK

10
Integrated Debugging

With version 3.0, Programmer Studio has evolved to include support for server based symbolic debuggers presented to the programmer within the standard development environment.

This evolution brings the prospect of remote program development even closer to the Windows-based IDE's that have become the standard for PC development.

Requirements and Supported Debuggers

Programmer Studio is essentially a client/server development environment, requiring server-based software to provide access to files and a command line. With version 3.0, this approach has been expanded to include a TELNET client within the development environment to allow server based software to be remotely controlled.

Obviously this places an immediate requirement on the user, - that the target server supports connection via TELNET clients. For the majority of Unix users this is not an issue as TELNET is the standard for establishing remote sessions. For HP3000 users, from MPE/iX 5.5 onwards TELNET is also available, although this may need to be set up before using the new features in Programmer Studio.

In addition, it is a requirement that the debugger used can be started from the directory containing the debug target (usually the program).

Programmer Studio supports the following symbolic debuggers. This list will continue to be expanded based upon user request.

- **GDB** (GNU Free Software Foundation) and compatibles (WDB HP-UX).
- **XDB** / Symbolic Debugger/iX (Hewlett-Packard).
- **TRAX** COBOL (Corporate Computer Systems)

How it Works

Programmer Studio is able to provide integrated debugging by assuming the role of a user controlling a debugger via the command line interface.

Each debug feature, step into, quickwatch, break, etc. is converted into the correct command line instruction and then sent to the debugger. The resulting response is then parsed and displayed to the user in the appropriate format. Programmer Studio provides this support through external modules, which are specific to each debugger used. These modules can then be enhanced and support provided for new debuggers without requiring the user to upgrade the software.

Working Model

Programmer Studio begins the process of debugging by establishing a TELNET session to the remote server. Following a successful connection, a number of challenge response sequences, (typically user name followed by password), are automatically completed before the session command line is detected.

The next step is to start the debugger on the server, set any user-defined breakpoints, and then begin executing the debug target (program).

Programmer Studio will then assume the debugger is running and will wait for the debugger to reach the first available statement (if this was the start option), a user-defined breakpoint, or a message to indicate the program has terminated.

Using TELNET

Unlike many of the protocols used in today's applications, such as the web and email, TELNET does not provide a programmatic interface for logging on. Instead text prompts are displayed to the user to which responses specific to the prompt are expected. After an undefined number of prompts have successfully been completed, the user is "logged-on" and able to use the TELNET session.

Typically these prompts and responses are simply for a username and then a password. These are easily configured, however some systems employ additional levels of security.

As a solution to this, Programmer Studio allows the user to determine a series of prompts to wait for, the correct responses, and finally the prompt that signifies the connection has been successful. These settings may either be specified by hand or the New Session Wizard can take the user through a successful connection step-by-step and complete the required fields.

Getting Started

Programmer Studio provides integrated debugging as part of a Programmer Studio project, so you will need to have a project set-up and ready to use before you can begin debugging. Please refer to Chapter 8, Developing Projects, in the Programmer Studio User Guide.

It is recommended that all source/list files included within the debug target (program) that may need to be opened within Programmer Studio should be included in the current project.

Before continuing, please also ensure that the target has indeed been compiled to include the information required by the debugger, or you have access to the relevant symbol or list files. For GDB this will usually involve using a compiler command line option. For XDB and TRAX this may involve creating a listfile during the compile.

Selecting your Debug Settings

Before you can begin debugging a program you will need to select which symbolic debugger you use and provide details of the debug target and other information. The following section describes each of the debug settings in detail.

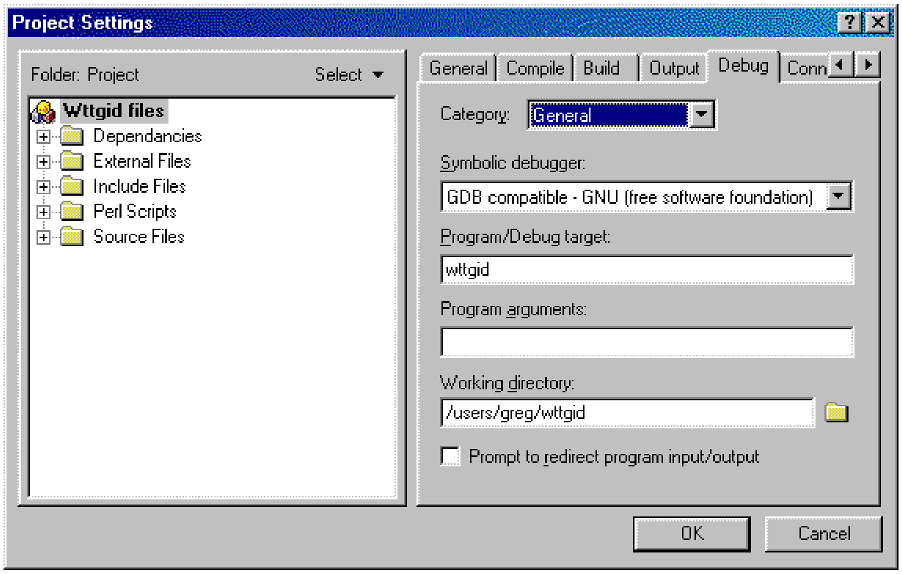
Project Debug Settings

To specify integrated debug settings within Programmer Studio, open an existing project. After a successful connection, select Settings from the Debug menu. The debug tab of the project properties will now be displayed.

The debug tab is subdivided into three categories, each is described below in detail:-

General

The general category specifies the debugger interface to be used, the debug target, (typically the program name), the command line parameters for the debug target, and the directory the debugger should be executed within.



Category: General

Symbolic debugger: GDB compatible - GNU (free software foundation)

Program/Debug target: wttgid

Program arguments:

Working directory: /users/greg/wttgid

Prompt to redirect program input/output

Viewing the General category in the project debug settings

Symbolic Debugger

This drop-down determines the symbolic debugger that will be used. Depending upon the selection, the remaining fields in this tab will be active and the status of items in the Debug menu will change.

Note Changing the specified symbolic debugger will delete all existing breakpoints.

Program/Debug Target

This field specifies the target to be debugged. This is typically a program name although this may depend upon the selected debugger used.

Note In Unix, commands are case-sensitive so be sure to ether the target name correctly.

Program Arguments

This field determines the additional arguments that should be passed to the target program within the debugger. This should not be confused with arguments passed to the debugger itself.

Working Directory

This field determines the directory in which the debugger is executed. This is typically the location of the debug target but may be any valid directory location. The working directory is selected immediately after the session login has been completed, using the *cd* command.

Note MPE/iX users debugging within the native file system should leave this field empty, instead using the session login to determine the active group.

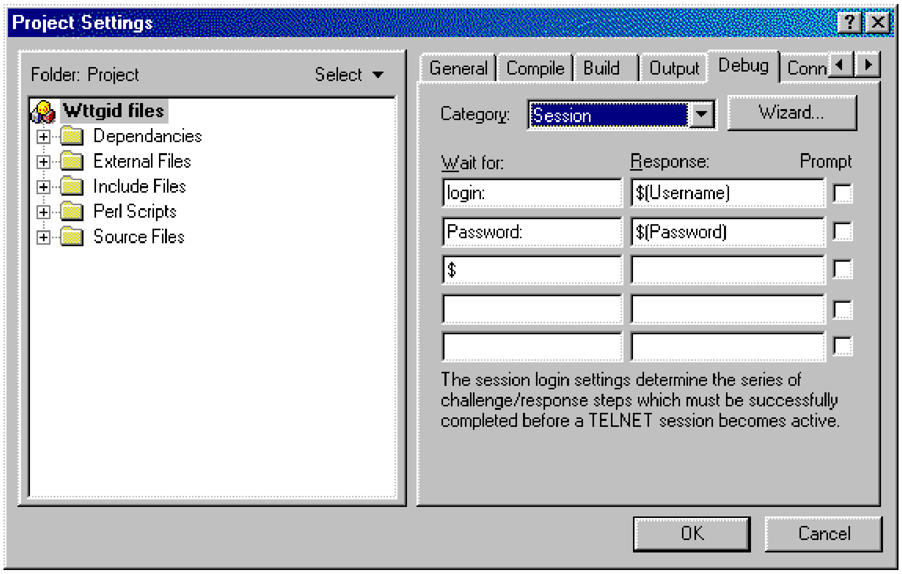
Prompt to Redirect Program Input/Output

This option prompts the user for a terminal identifier to which all debug target input and output is then subsequently redirected. This can be particularly useful when the program has complex I/O that may become confused with the debugger command line when allowed to run within the debug session tab in the output window.

Note This is currently only supported on Unix platforms.

Session

The session category specifies the process of a TELNET login, described as a series of challenges and responses. Two fields specify the command prompt to be expected and then the correct response. A *Wait for* field without a response indicates a successful login. The *Wizard* button starts the TELNET session wizard that will take you step-by-step though a login process.



Category: Session

Wait for:	Response:	Prompt
login:	$(Username)	☐
Password:	$(Password)	☐
$		☐
		☐
		☐

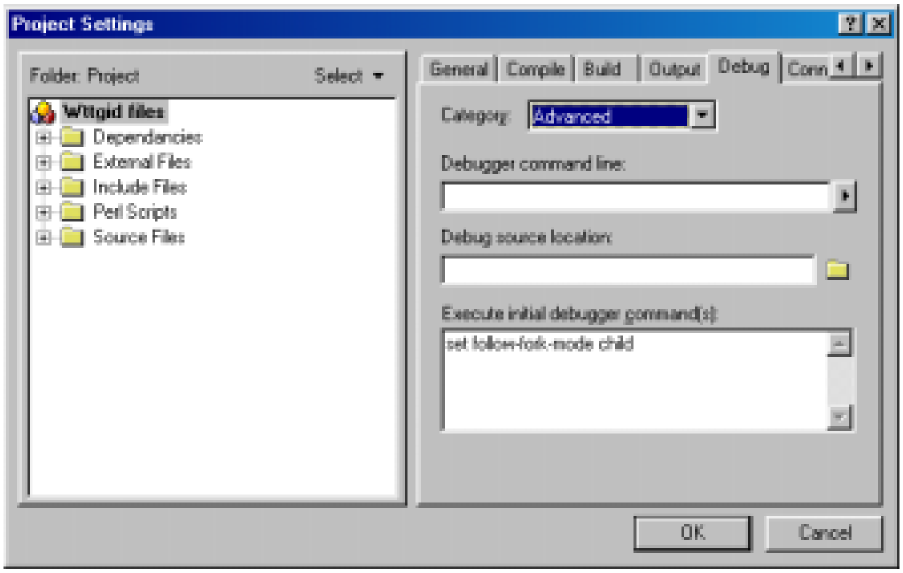
The session login settings determine the series of challenge/response steps which must be successfully completed before a TELNET session becomes active.

Viewing the Session category in the project debug settings

The TELNET session wizard is described in detail in the next section.

Advanced

The advanced category specifies additional options for debugger execution. Only users familiar with the functionality of the target debugger should modify these settings.



Category: Advanced

Debugger command line:

Debug source location:

Execute initial debugger command(s):

```
set follow-fork-mode child
```

Viewing the Advanced category in the project debug settings

Debugger Command Line

This field allows an advanced user to override the command that would normally be executed to start the host based debugger. This may be useful in circumstances where the debug command line requires a fully qualified location or requiring special options to be included.

The button to the right of the field provides a popup menu containing variables which may be used within the debugger command line, substituted for actual settings at run-time. For example, if the debug target was *wttgid*,

gdb *$(DebugTarget)*

would be converted to,

gdb *wttgid*

A more complex example is that required by the TRAX COBOL debugger. In this example the debug command line includes a fully qualified program and both the target and target arguments.

RUN **TRAX.RUN**;INFO="*$(DebugTarget)* F *$(DebugTargetArgs)*"

In most cases the average user will probably not need to specify an alternate debugger command line. However for those users who wish to, the option is there.

Debug Source Location

The field provides Programmer Studio with the location of files identified by the debugger when stepping into a debug target or when a break is reached in program execution (as a result of step over, step into or reaching a breakpoint). Most debuggers identify files using a fully qualified filename. For those just displaying a filename, Programmer Studio requires a source directory to look in.

A good example of this requirement is when using the TRAX debugger, in which filenames are identified by module name only and do not include the actual source (list file) location.

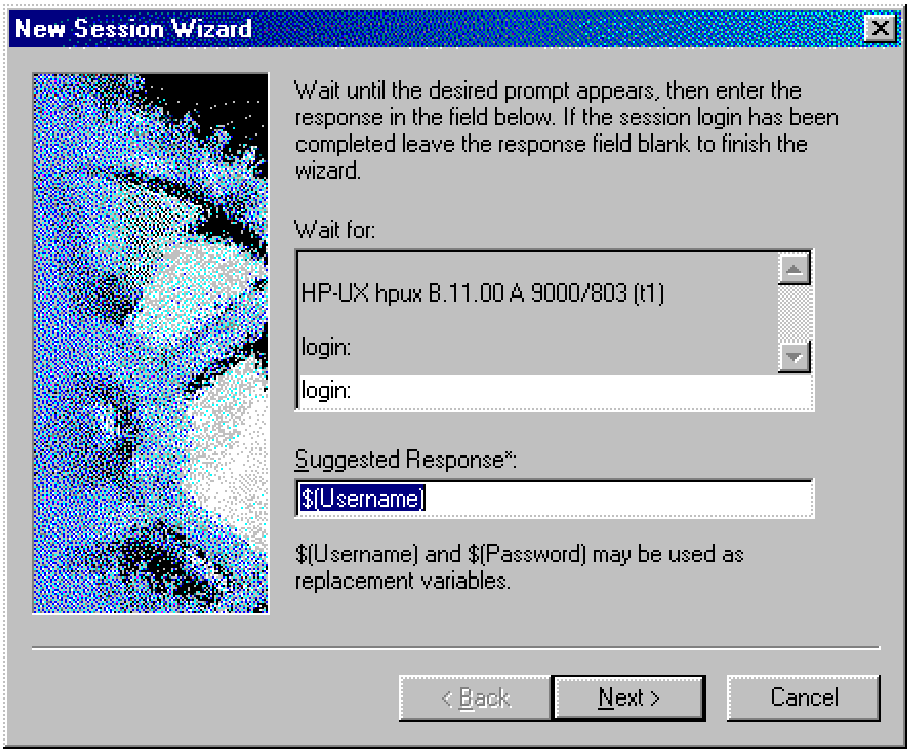
Execute Initial Debugger Commands

This field allows the user to enter any number of commands which should be executed prior to the setting of the initial breakpoints and before the program is started within the debugger.

This can be very useful for setting debugger options which affect the execution of the program being debugged. However, these commands are not expected to require further input and will return the debugger to its normal input prompt after completion.

TELNET Session Wizard

The New Session Wizard uses the server name specified in the current project and attempts to establish a TELNET connection. If successful, the prompt immediately received is displayed to the user, who has to then determine if this is indeed a valid prompt or if additional text is expected.


New Session Wizard

Wait until the desired prompt appears, then enter the response in the field below. If the session login has been completed leave the response field blank to finish the wizard.

Wait for:

```
HP-UX hpux B.11.00 A 9000/803 (t1)

login:
login:
```

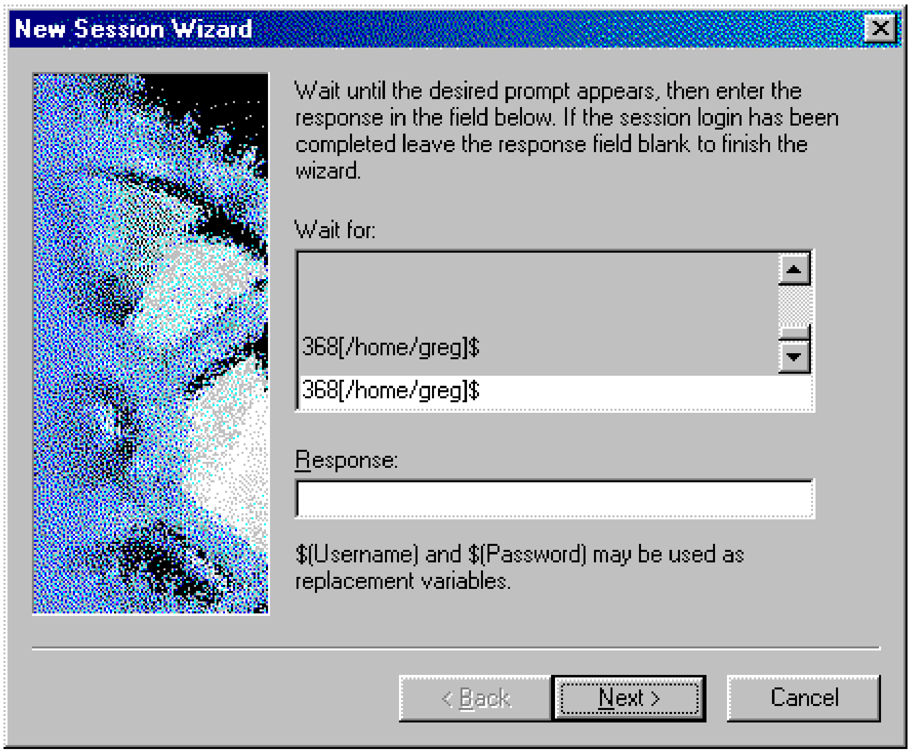
Suggested Response*: $(Username)

$(Username) and $(Password) may be used as replacement variables.

Selecting a response for the displayed prompt in the TELNET session wizard

As text is received from the TELNET server, this is displayed in the Wait for field with the very last line displayed in the editable field. Wait for the expected prompt to appear in the Wait for field and then enter your selected response.

As each response is received the New Session Wizard will attempt to help the user by providing responses to "login" and "password" prompts using the current project settings.


New Session Wizard

Wait until the desired prompt appears, then enter the response in the field below. If the session login has been completed leave the response field blank to finish the wizard.

Wait for:

```
368[/home/greg]$
368[/home/greg]$
```

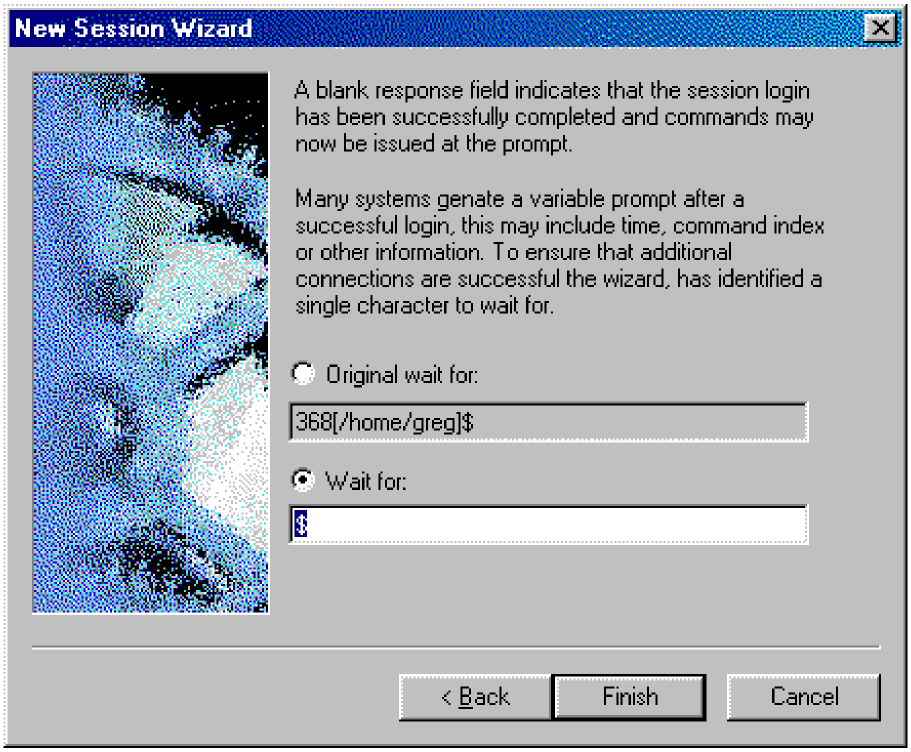
Response:

$(Username) and $(Password) may be used as replacement variables.

Selecting the login completion response in the TELNET session wizard

This process should be repeated until the login process has been completed, at which point the Suggested response field should be left blank and the Next button clicked.

The New Session wizard will now check the format of the prompt indicating the connection is successful. As no session prompt is the same, some include date or time, the user may be prompted to select a more general prompt to wait for (typically the last character in the original wait for text).



A blank response field indicates that the session login has been successfully completed and commands may now be issued at the prompt.

Many systems genate a variable prompt after a successful login, this may include time, command index or other information. To ensure that additional connections are successful the wizard, has identified a single character to wait for.

Original wait for:

368[/home/greg]$

[✓ Wait for:]

$

Confirming the login completion response in the TELNET session wizard

Following the successful completion of the New Session Wizard, the session settings will be entered in the project settings. This lists the prompts and responses selected during the wizard. If you wish to use the New Session Wizard again, click on the *Wizard* button.

Debugging a Program

Once the Debug settings have been specified within the project settings, the user may set breakpoints and begin debugging the specified target. It should be remembered that all debuggers are different, and although Programmer Studio attempts to duplicate functionality across various platforms, no two debuggers behave exactly the same way.

Before you begin debugging, it is recommended that you familiarize yourself with the debugger you are intending to use. This document is not intended to be an in-depth guide to interactive debugging, simply to provide an introduction to the interface that Programmer Studio provides.

Setting Breakpoints

Before debugging the specified target, most users wish to set breakpoints, - identified points within the target which, when reached, cause the debugger to suspend target execution offering options to the user.

Breakpoints may either be set by location, filename and line number, or by function/procedure entry point. Once the debugger has loaded the target, Programmer Studio will attempt to set these breakpoints, displaying an error message if this location is invalid.

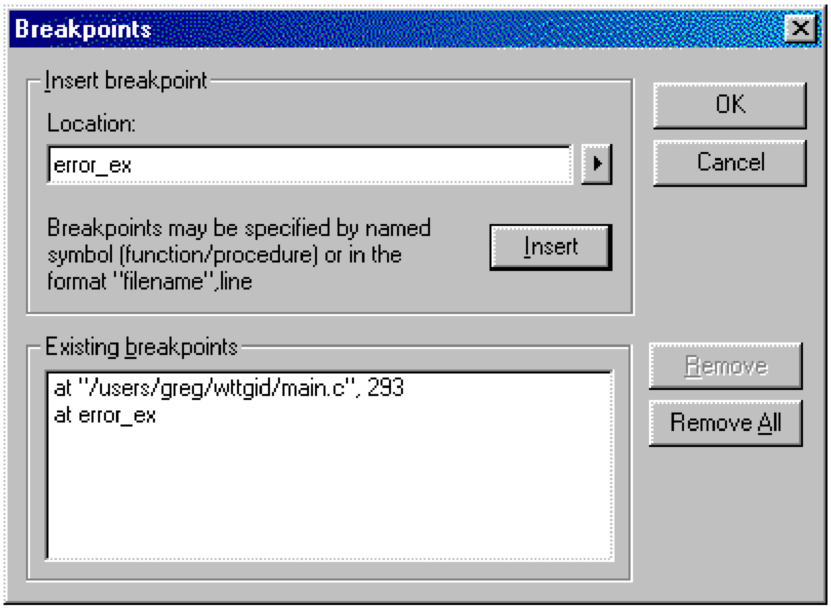
Setting a breakpoint within the Code Editor

A breakpoint can be easily set within the Code Editor by selecting Toggle Breakpoint from the Debug menu (F9). If the current line is already set as a breakpoint, this will remove the breakpoint. It is important to ensure the line highlighted is valid, as Programmer Studio will not be able to determine this for you. Your symbolic debugger documentation will provide details of what constitutes a valid Breakpoint line.

Setting a breakpoint within the Breakpoints Window

An alternative way to set a breakpoint is through the Breakpoints window. This lists all breakpoints currently set, by either filename and line number or by function/procedure name.

To display the Breakpoints window, from the Debug menu, select Breakpoints.



error_ex

Breakpoints may be specified by named symbol (function/procedure) or in the format "filename",line

at "/users/greg/wttgid/main.c", 293
at error_ex

Adding a named symbol breakpoint, "error_ex"

In this example screen shot, two breakpoints have been added to the current debug target. The first breakpoint identifies a particular line in a source file, in this case line 293 in a file called main.c. The second breakpoint is set at the entry point for a function/procedure named error_ex.

Note The case sensitivity of both filenames and function/procedure names is very important. Although specific operating systems and programmer languages may be case insensitive, please check before adding each breakpoint.

Pressing the OK button will update the breakpoints for the current debug target. All breakpoints set by filename and line number will be updated within the Code Editor to reflect their status. Breakpoints set by function/procedure name are not highlighted in the Code Editor, as these are evaluated at debug time by the symbolic debugger.

Starting a Debug Session

Programmer Studio provides a number of entry points from which to debug the specified target:

- ♦ To start the debug target and to stop only at specified breakpoints, from the **Debug** menu, select **Start Debugging**, then select **Go**.
- ♦ To start the debug target suspended at the first accessible statement, from the **Debug** menu, select **Start Debugging**, then select **Step Into**.
- ♦ To attach the debugger to a existing process running on the server (UNIX only), from the **Debug** menu, select **Start Debugging**, then select **Attach to Process**.

After selecting one of these options, Programmer Studio will establish a new TELNET session to the server and invoke the chosen debugger with the relevant instructions. Programmer Studio will then wait for the remote debugger to display a specific prompt to indicate the debug target is suspended or the program has terminated.

Controlling the Debug Target

Programmer Studio will suspend all further debug options until it is detected that the debugger has suspended execution of the debug target (this may have happened as result of a breakpoint being reached or selecting to start the program suspended).

Once the debugger prompt has been identified, the main window's caption will be modified to include the text [break] and many of the remaining options on the debug menu will become enabled. Certain items will be dependant on the line and text highlighted in the Code Editor.

This section will describe the functionality of each debug menu item and how it should function. It is very important at this stage to remember that Programmer Studio is merely controlling the debugger on the server, and that you may enter your own commands in the debug session window.

References to *current statement* indicate the current position of execution of the debug target and **not** the current/selected line in the Code Editor.

Continue

This will continue the execution of the debug target.

Restart

This will restart the debug target keeping existing breakpoints.

Stop Debugging

This will terminate the debug target and exit the debugger, closing the TELNET session.

Break

This will attempt to suspend the execution of the debug target, locating the current statement and highlighting this within the Code Editor. In the event the current statement does not identify a filename and line number, the Call Stack is displayed.

Kill Debug Session

This will simply close the current TELNET session, which in turn should terminate the debugger and target on the server. This option should be used with caution and only in cases where Stop Debugging fails.

Step Into

This will attempt to step into the sub-procedure/function located on the current statement. If the current statement does not identify a sub-procedure/function this may fail or Step Over (depending upon the debugger).

Step Over

This will attempt to execute the code represented by the current statement and break execution at the next statement in the current file. If there isn't a next statement in the current file, this may fail or highlight the next executable statement (depending upon the debugger).

Run to Cursor

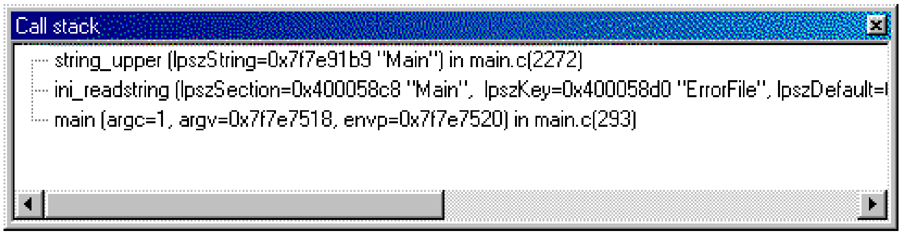
This will attempt to set a temporary breakpoint at the statement on which the cursor entry point is located in the Code Editor and then continue program execution.

Set Next Statement

This will attempt to move the current statement to the statement on which the cursor entry point is located in the Code Editor without executing code in between. This should be used with caution as it may leave the debug target in an unusable state.

View Call Stack

This will display the current call hierarchy for the current statement.

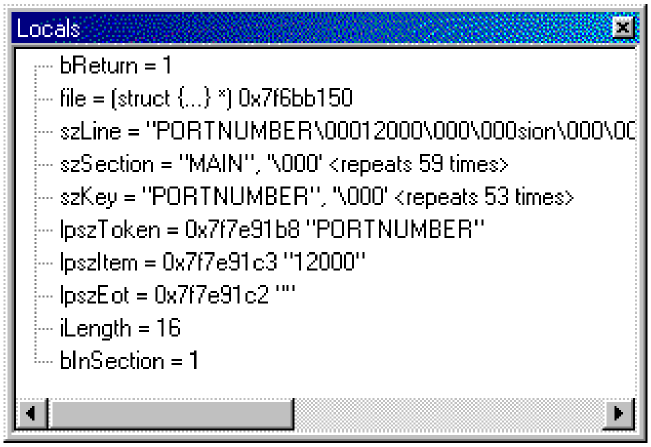


```
string_upper (lpszString=0x7f7e91b9 "Main") in main.c(2272)
ini_readstring (lpszSection=0x400058c8 "Main",  lpszKey=0x400058d0 "ErrorFile", lpszDefault=
main (argc=1, argv=0x7f7e7518, envp=0x7f7e7520) in main.c(293)
```

Viewing the current callstack

View Locals

This will display the contents of all local variables (if applicable to the programming language) in their native format.



```
bReturn = 1
file = (struct {...} *) 0x7f6bb150
szLine = "PORTNUMBER\00012000\000\000sion\000\0C
szSection = "MAIN", '\000' <repeats 59 times>
szKey = "PORTNUMBER", '\000' <repeats 53 times>
lpszToken = 0x7f7e91b8 "PORTNUMBER"
lpszItem = 0x7f7e91c3 "12000"
lpszEot = 0x7f7e91c2 ""
iLength = 16
bInSection = 1
```

Viewing the current list of local variables

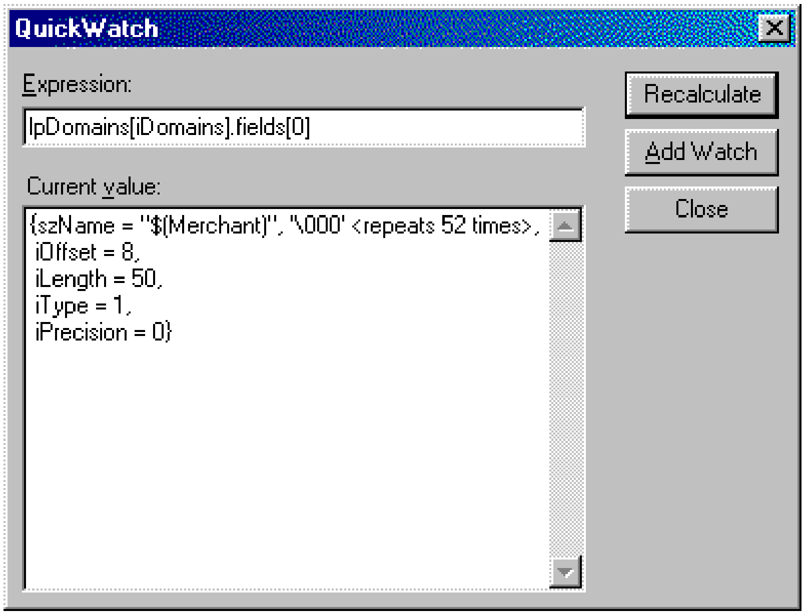
Warning In COBOL this typically displays all variables defined within the WORKING-STORAGE section of the program, and as such should be avoided.

Show Next Statement

This will highlight the debug target's current statement in the Code Editor, opening any necessary files.

Quick Watch

This will display the Quick Watch dialog box. This evaluates the currently selected text or the keyword at the insertion point in the Code Editor. The results of the evaluation will then be displayed in the native format.



```
lpDomains[iDomains].fields[0]
```

```
{szName = "$(Merchant)", '\000' <repeats 52 times>,
 iOffset = 8,
 iLength = 50,
 iType = 1,
 iPrecision = 0}
```

Displaying the value of the expression lpDomains[iDomains].fields[0]

Within the Quick Watch window a watch may be placed on the specified expression. This will then be evaluated in the Watch output window every time the debug target enters a suspended state.

Ending a Debug Session

The current debug session may be terminated either normally, by the program being debugged terminating, by instructing the debugger to end the debug session, or by terminating the debug session itself.

A debug session should only be 'killed' in the event Programmer Studio becomes confused as to the state the debugger is currently in. For example, in the event the debugger is terminated on the server.

Accessing the Remote Debugger

The TELNET session is accessible as a tab within the output window. Displayed to the right of Find in Files, the Debug Session tab contains the TELNET client in use by the current debug session.

This client may be used while the debugger is suspended (this can be determined by the presence of [break] in the Programmer Studio window caption) to access information not provided by the standard development environment command items. For example, the debug session window can be used to change local/active variables or display debugger specific information such as registers.

Care should be taken when accessing the remote debugger from the session window as this may leave Programmer Studio unable to determine the current state of the remote debug session. In the event that a command is used which changes the current statement, select Show Current Statement from the Debug menu.

Terminal Emulation

The debug session window, displayed in the output window, is itself a mini-terminal emulator. The terminal window is limited in functionality to a subset of VT100 (Unix) or HP2392 (HP3000) depending on the server connected to. The debug session window is not intended to replace your existing terminal emulator, instead simply to facilitate access to the host based debugger.

If you identify any shortcomings in the emulation for VT100 or HP2392 please email support@whispertech.com providing details of the problem and preferably programs which exhibit the problem.

TRAX COBOL

This section will take you through the process of setting up Programmer Studio to debug a COBOL program using the TRAX COBOL debugger. For the purposes of illustrating the interface between TRAX and Programmer Studio the examples used here are based on the Introductory Tutorial installed with TRAX.

Start by creating a project and connecting to the HP3000, use the manager user and login to the TRAX COBOL installation account (TXCBXL). The two files that will be used in this section are located in the LST group and are named HTLST and LURQ. Add these to the project specifying Hewlett-Packard COBOL as the Code Editor Template.

From the Debug menu, select Settings. The project settings dialog box will now appear with the Debug tab automatically selected. As previously described, the debug settings are subdivided into three categories, General, Session and Advanced.

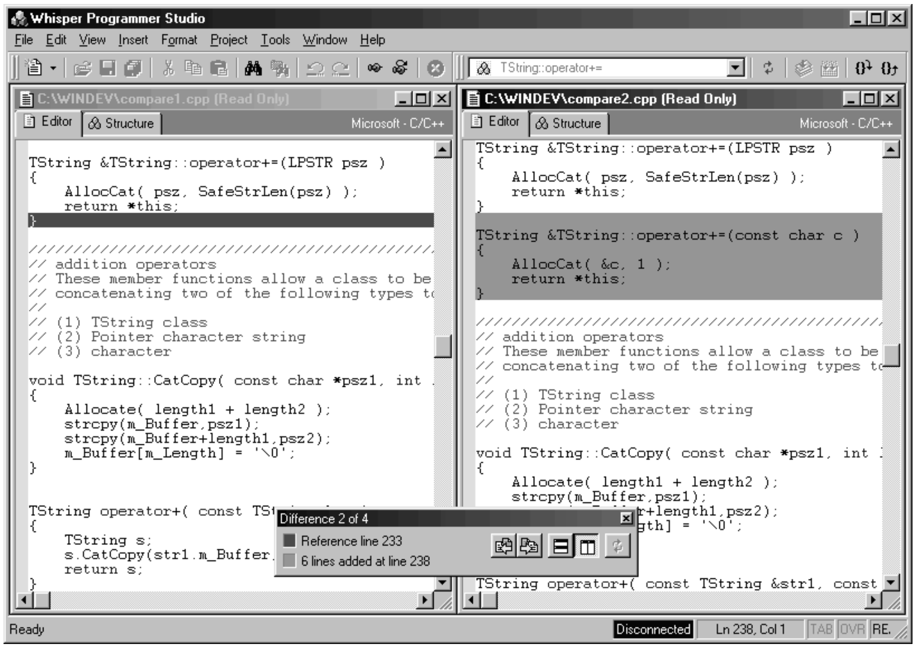
1. From the **Symbolic debugger** combo-box, select **TRAX…**
2. In the Program/debug target field, enter TSTPRG
3. From the **Category** combo-box, select **Session**
4. Click on the **Wizard** button
5. Complete the session wizard as described in the Getting Started section
6. From the **Category** combo-box, select **Advanced**
7. In the Debugger command line field, enter RUN TRAX.RUN;INFO="$(DebugTarget) F $(DebugTargetArgs)"
8. In the Debug source location field, enter /TXCBXL/LST
9. Click **OK** to save the settings

You should be able to debug the sample program from within Programmer Studio. Remember, Programmer Studio is only remotely controlling TRAX, so you can select the debug session tab in the output window at any time to enter your own commands.

11
Visual File Compare

A common task among programmers, especially those working in a team, is to determine differences between various versions of the same code. This can be a very time consuming task which, when done by hand, is far from accurate.

Programmer Studio does not include file comparison tools; instead it provides a conduit to external utilities which output results in one of two recognized formats. Using an external utility has the distinct advantage of allowing the user to determine which of the many tools available to use, while still providing a standard visual interface to the results generated.



Whisper Programmer Studio
File Edit View Insert Format Project Tools Window Help
TString::operator+=
C:\WINDEV\compare1.cpp (Read Only)
Editor
Structure
Microsoft - C/C++
C:\WINDEV\compare2.cpp (Read Only)
Editor
Structure
Microsoft - C/C++
Difference 2 of 4
Reference line 233
6 lines added at line 238
Ready
Disconnected
Ln 238, Col 1
TAB
OVR
RE.

```
TString &TString::operator+=(LPSTR psz )
{
    AllocCat( psz, SafeStrLen(psz) );
    return *this;
}

//////////////////////////////////////////////
// addition operators
// These member functions allow a class to be
// concatenating two of the following types tc
//
// (1) TString class
// (2) Pointer character string
// (3) character

void TString::CatCopy( const char *psz1, int .
{
    Allocate( length1 + length2 );
    strcpy(m_Buffer,psz1);
    strcpy(m_Buffer+length1,psz2);
    m_Buffer[m_Length] = '\0';
}


TString operator+( const TSt
{
    TString s;
    s.CatCopy(str1.m_Buffer.
    return s;
}
```

```
TString &TString::operator+=(LPSTR psz )
{
    AllocCat( psz, SafeStrLen(psz) );
    return *this;
}

TString &TString::operator+=(const char c )
{
    AllocCat( &c, 1 );
    return *this;
}

//////////////////////////////////////////////
// addition operators
// These member functions allow a class to be
// concatenating two of the following types tc
//
// (1) TString class
// (2) Pointer character string
// (3) character

void TString::CatCopy( const char *psz1, int 1
{
    Allocate( length1 + length2 );
    strcpy(m_Buffer,psz1);
           r+length1,psz2);
           gth] = '\0';


TString operator+( const TString &str1, const
```

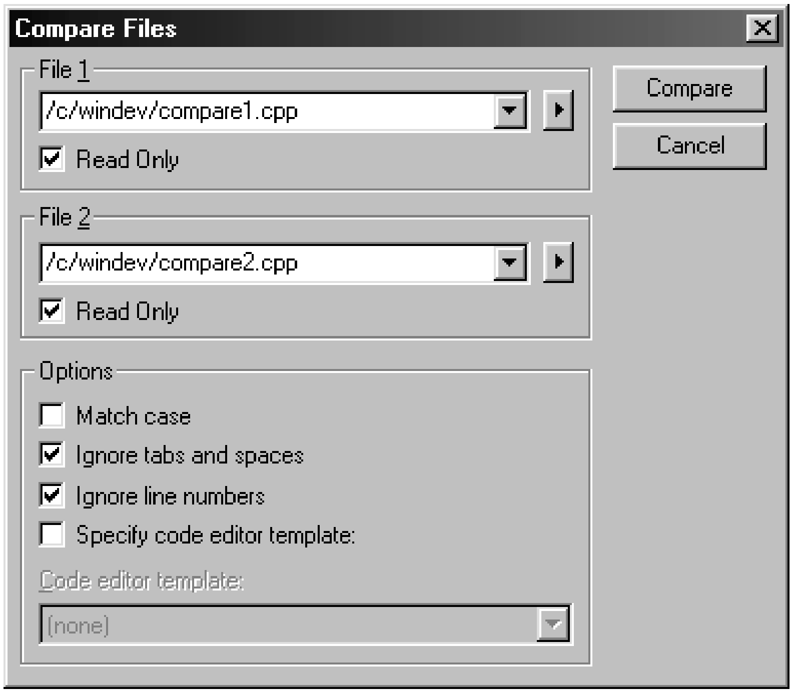
Visual File Compare highlighting a difference between two files

Programmer Studio supports file comparison result formats of DIFF compatible utilities and Microsoft FC.

Comparing Files

Programmer Studio compares files by first loading each file, (if not already open), as they would be when edited normally. Once loaded, each file is subsequently saved on the PC removing any non-editable text, (line numbers and line tags), identified by the Code Editor template or file compare options. Finally, the file compare utility is invoked to compare the two files based on the compare options.

The Compare Files dialog box provides a number of options which determine how the file is formatted when passed to the comparison utility, and which options are used.


Compare Files

File 1: /c/windev/compare1.cpp
[✓] Read Only

File 2: /c/windev/compare2.cpp
[✓] Read Only

Options
[ ] Match case
[✓] Ignore tabs and spaces
[✓] Ignore line numbers
[ ] Specify code editor template:

Code editor template: (none)

Comparing two files, ignoring spaces and line numbers

Note When using file compare for the first time, Programmer Studio will prompt the user to select which compare utility to use, FC or DIFF. To change this at any time, use the File Compare tab in the Options dialog box.

To compare two files

1. From the Tools menu, select Compare Files
2. In the File 1 field, enter the name of the first file to compare
3. In the File 2 field, enter the name of the second file to compare
4. From the options, select additional compare settings
5. Click Compare to start the file comparison

Tip Use the arrow button to the right of the File fields to browse for local or remote filenames or select from a list of currently open files.

The file comparison can be cancelled at any time by pressing the Stop button, or select Stop from the View menu. Programmer Studio provides the following options for the file comparison:

Match case

This setting is passed to the comparison utility as an optional parameter. When selected, characters differing in case will be highlighted. By default this option is not enabled.

Ignore tabs and spaces

This setting is passed to the comparison utility as an optional parameter. When selected, compound tabs and spaces are treated as a single space when comparing lines. By default this option is enabled.

Ignore line numbers

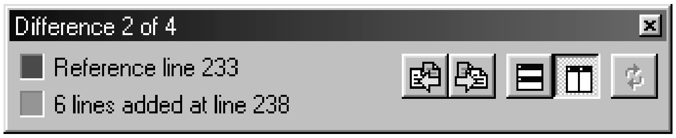
This setting instructs Programmer Studio to remove any detected line numbers from the contents of a file prior to invoking the comparison utility. If the Code Editor Template used by the file already removes line numbering while editing, this option has no effect.

Specify code editor template

This setting overrides the Code Editor Template that would normally be assigned to the file, (either by project settings or the rules wizard). This is especially useful when comparing files which would not normally be opened with the correct Code Editor Template, ensuring that differences in non-editable code are not highlighted.

Comparison Results

Following a successful comparison, the differences tool window will be displayed and the first difference highlighted in the Code Editors of the two files. The key provides a visual indication of the difference highlighted, the starting line position and the number of lines highlighted.



Difference tool window following a file compare

The toolbar buttons allow the user to move forwards and backwards through the differences highlighted, and also change the orientation of the windows between portrait and landscape.

Synchronizing Files

Advanced users may wish to use the Visual File Compare to synchronize two files. This can be particularly useful when consolidating changes made in two versions of the same file.

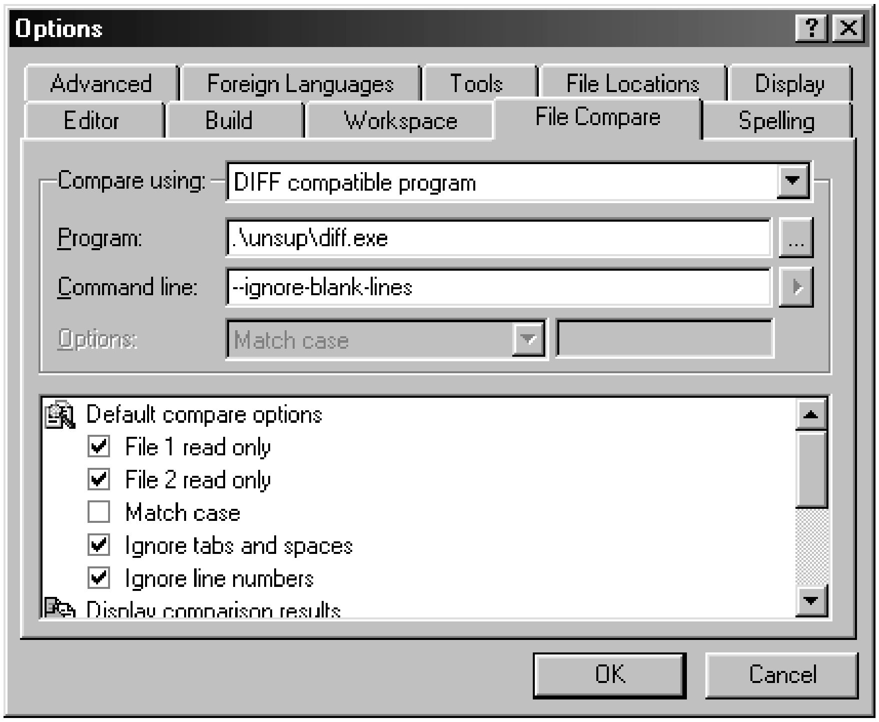
To allow synchronization, when comparing files, ensure that either file 1 or 2 (or both) are compared with the read-only option off. When the files are displayed with their respective differences, modify either file (or both) and click the refresh button in the differences tool window.

Refreshing the comparison will then re-compare the two files, updating the differences tool window and highlighting the first difference in the updated files.

Note Refreshing a file compare does **not** save any changes made to the file. Use the Save, Save As or Save Local items in the File menu to save any changes.

Advanced Options

The File Compare tab in the Options dialog box provides options to select the comparison utility to be used, determine additional options, and even set the default settings for the File Compare dialog box.



Compare using: DIFF compatible program

Program: .\unsup\diff.exe

Command line: --ignore-blank-lines

Options: Match case

Default compare options
- [✓] File 1 read only
- [✓] File 2 read only
- [ ] Match case
- [✓] Ignore tabs and spaces
- [✓] Ignore line numbers

Display comparison results

File Compare options, with an addition command line parameter for DIFF

To view file compare options

1 From the Tools menu, select Options

2 Select the File Compare tab

Depending upon the selection in the Compare using field, additional fields will become enabled for additional settings. The following sections describe the options available and their use.

Microsoft Windows FC.EXE

This uses the standard file compare utility installed as part of the Microsoft Windows operating system. FC is a good utility for comparing files reporting general differences, but lacks the accuracy of utilities written specifically for comparing line based files.

There are no additional options.

DIFF Compatible Program

DIFF is a popular utility among UNIX users which has been ported to many non-UNIX platforms, including Windows. DIFF is a far more sophisticated "differencing" utility then FC, providing a number of additional options to generate a very accurate list of differences.

The Program field specifies the location of the DIFF utility that will be used. The Command line field specifies additional command line options for the diff utility.

Note Programmer Studio will automatically add command line options for matching case and ignoring tabs and spaces based on the options selected in the File Compare dialog. The command line field is intended for additional parameters only.

Tip To determine which additional command line options are available using GNU DIFF (this version is installed with Programmer Studio under the GNU public license), run the program from the DOS prompt without any command line parameters.

Windows Application

For those users wishing to use a third-party comparison application such as WINDIFF, Programmer Studio provides the option to specify which program to launch when comparing files. This method does not use the file compare conduit, so results will not be displayed in Programmer Studio. This is the responsibility of the application being launched.

The Program field specifies the Windows application that will be launched, use the Command line field to determine the format of the command line parameters using the following predefined variables:

To match	Use
First file being compared	$(File1)
Second file being compared	$(File2)
Command line option to match case when comparing	$(MatchCase)
Command line option to ignore tabs and spaces	$(IgnoreSpaces)

When adding command line options, $(MatchCase) and $(IgnoreSpaces), enter in the options field below the actual text that should be inserted into the command line.

12
Using Find In Files

With larger programs consisting of a number of relatively small files, searching for text can be a real problem. Opening each file in turn to search for the specific string can be time consuming and never 100% accurate.

Find in files provides the ideal solution to this problem, by providing the capability to search for text in a subset of files. The search is quickly completed, returning a list of matching files into the output window, allowing the user to examine each occurrence in turn.

The search can be cancelled at any time by selecting Stop from the View menu.

To find a specific text string in a subset of files

1 From the Edit menu, select Find In Files (CTRL+SHIFT+F)

2 In the Find What box, enter the text string to search for

3 In the In File Types box, enter the UNIX style name pattern match which is to be used to determine the files to search

4 In the In Directory box, enter the location of the files to be matched against the criteria

5 To determine how you want the search to proceed, select any of the options described in the table below.

6 Click Find

Tip To specify a number of different name patterns use semicolon as a delimiter i.e. (t*;*.c).

The following options determine how the search proceeds:-

To	Select
Find whole words and not sub-strings inside words. For example, help and not helping.	Match whole word only
Find words matching the case of the text in the Find what box. For example help and not HELP.	Match case
Use regular expressions when searching	Regular Expression
Search files included in the project only	Search project files only
Continue the search in sub-directories	Search sub-directories
Display the file currently being searched in the Output Window	Display progress

Note See Chapter 6, Using the Code Editor, for more information on how to use regular expressions to find text strings.

As each occurrence of the search criteria is highlighted, Programmer Studio will attempt to load the file containing the text string, (if it is not already loaded), and highlight the line on which the criteria is matched

To highlight the line matching the Find in Files criteria

1 Ensure the Find In Files tab is activated in the Output Window

2 From the Edit menu, select Go To, from the secondary menu select Next Error/Tag (F4)

13 Remote Command Line

In addition to compile and build commands, Programmer Studio also provides access to the remote server command line for executing simple commands.

The following example uses the standard Unix command *ls* to display the contents of the current server directory. This is intended as an example. Any command can be used as long as it does not require user input.

To view the contents of the current server directory

1 From the View menu, select Command Line (F12)

2 At the command prompt enter *ls* and press return

The remote command can be cancelled at any time by selecting Stop from the View menu.

The remote command line window such should not be confused with a terminal emulator. It is provided purely as a tool for executing commands and not for running programs requiring user interaction.

A
Regular Expression Characters

Programmer Studio provides extensive support for searching files using regular expressions. A regular expression is simply a string of characters which provide a method for describing a pattern using special characters (operators) and literal characters.

A regular expression can be either simple or compound. A simple regular expression describes one character. A compound regular expression describes a sequence of simple and compound expressions.

To Match	Use
Any single character	.
a, b or c	[abc]
Any lowercase alphabetic character	[a-z]
Any character except e	[^e]
Any character except lowercase alphabetic characters	[^a-z]
Either expression a or expression b	a | b
Groups a compound expression into a single expression or reference by operators such as * or ?	(expression)
The preceding item is optional and matched at most once	?
The preceding item will be matched zero or more times	*
The preceding item will be matched one or more times	+
The preceding item is matched exactly n times	{n}
The preceding item is matched n or more times	{n,}
The preceding item is optional and is matched at most m times	{,m}
The preceding item is matched at least n times, but not more than m times	{n,m}

In addition to the standard expressions, specific compound expressions are predefined:

To Match	Use
An alphanumeric character	\:a
An alphabetic character	\:c
A control character	\:n
A numeric character	\:d
A graphical character	\:g
A lowercase character	\:l
An uppercase character	\:u
A printable character	\:p
White-space character	\:b
Hexadecimal character	\:h
Any string enclosed in quotes, such as ‘sample’ or “sample”	\:q
Any string enclosed in single quotes, such as ‘sample’	\:QS
Any string enclosed in double quotes, such as "sample"	\:QD

B

7-bit ASCII Character Set

Decimal	Hex	Octal	Char	Description
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)
16	10	020	DLE	(data link escape)
17	11	021	DC1	(device control 1)
18	12	022	DC2	(device control 2)
19	13	023	DC3	(device control 3)
20	14	024	DC4	(device control 4)
21	15	025	NAK	(negative acknowledge)
22	16	026	SYN	(synchronous idle)
23	17	027	ETB	(end of trans. block)

Decimal	Hex	Octal	Char	Description
24	18	030	CAN	(cancel)
25	19	031	EM	(end of medium)
26	1A	032	SUB	(substitute)
27	1B	033	ESC	(escape)
28	1C	034	FS	(file separator)
29	1D	035	GS	(group separator)
30	1E	036	RS	(record separator)
31	1F	037	US	(unit separator)
32	20	040	SPACE	
33	21	041	!	
34	22	042	"	
35	23	043	#	
36	24	044	$	
37	25	045	%	
38	26	046	&	
39	27	047	'	
40	28	050	(	
41	29	051	)	
42	2A	052	*	
43	2B	053	+	
44	2C	054	,	
45	2D	055	-	
46	2E	056	.	
47	2F	057	/	
48	30	060	0	
49	31	061	1	
50	32	062	2	
51	33	063	3	

Decimal	Hex	Octal	Char	Description
52	34	064	4	
53	35	065	5	
54	36	066	6	
55	37	067	7	
56	38	070	8	
57	39	071	9	
58	3A	072	:	
59	3B	073	;	
60	3C	074	<	
61	3D	075	=	
62	3E	076	>	
63	3F	077	?	
64	40	100	@	
65	41	101	A	
66	42	102	B	
67	43	103	C	
68	44	104	D	
69	45	105	E	
70	46	106	F	
71	47	107	G	
72	48	110	H	
73	49	111	I	
74	4A	112	J	
75	4B	113	K	
76	4C	114	L	
77	4D	115	M	
78	4E	116	N	
79	4F	117	O	

Decimal	Hex	Octal	Char	Description
80	50	120	P	
81	51	121	Q	
82	52	122	R	
83	53	123	S	
84	54	124	T	
85	55	125	U	
86	56	126	V	
87	57	127	W	
88	58	130	X	
89	59	131	Y	
90	5A	132	Z	
91	5B	133	[	
92	5C	134	\	
93	5D	135	]	
94	5E	136	^	
95	5F	137	_	
96	60	140	`	
97	61	141	a	
98	62	142	b	
99	63	143	c	
100	64	144	d	
101	65	145	e	
102	66	146	f	
103	67	147	g	
104	68	150	h	
105	69	151	i	
106	6A	152	j	
107	6B	153	k	

Decimal	Hex	Octal	Char	Description
108	6C	154	l	
109	6D	155	m	
110	6E	156	n	
111	6F	157	o	
112	70	160	p	
113	71	161	q	
114	72	162	r	
115	73	163	s	
116	74	164	t	
117	75	165	u	
118	76	166	v	
119	77	167	w	
120	78	170	x	
121	79	171	y	
122	7A	172	z	
123	7B	173	{	
124	7C	174	|	
125	7D	175	}	
126	7E	176	~	
127	7F	177	DEL	