A REFERENCE INFORMATION

In This Appendix

This appendix contains the following topics:

Glossary

The following terms are important toward understanding VisualDSP++:

Breakpoint — A user-defined halt in an executable program. Toggle breakpoints (turn them on or off) by double-clicking on a location in a Disassembly window or Editor window.

Break condition — A hardware condition under which the target breaks and returns control of the target back to the user. For example, a break condition could be set up to occur when address 0x8000 is read from or written to. See also "**Watchpoints**."

Build — Performing a build (or project build) refers to the operations (preprocessing, assembling, and linking) that VisualDSP++ performs on projects and files. During a build, VisualDSP++ processes the files in your project that have been modified (or depend on files that have been modified) since the previous build. A build differs from a **rebuild all**. During a rebuild all, VisualDSP++ processes all the files in the project, regardless whether they have been modified.

Build type — This term has been replaced by "**configuration**."

Configuration (or project configuration) — You develop a project in stages (configurations). By default, a project includes two configurations: Debug and Release. A configuration refers to the collection of options (tool chain and individual options for files) specified for the configuration. You can add a configuration to your project at any time. You can delete a customized configuration that you created, but you cannot delete the Debug or Release configurations.

Current directory — The directory in which the `.DPJ` file is saved. The build tools use the current directory for all relative file path searches. See also "**Default intermediate and output file directories**."

Data set — A series of data values in DSP memory used as input to a plot. Create data sets and configure the data for each data set. You specify the memory location, the number of values, and other options that identify the data. 3-D plots require additional specifications for row and column counts.

Debug configuration — For a debug configuration, you can accept the default options, or you can specify the options you want and save them. The configuration refers to the specified options for all the tools in the tool chain. See also "**Configuration**."

Debug session (or session) — The combination of a target and a platform. For example, a session can be a JTAG emulator target connected to a platform consisting of five ADSP-21*xxx*'s. Another example of a debug session is an ADSP-21*xxx* EZ-KIT target connected to an ADSP-21*xxx* EZ-KIT board.

The DSP projects you develop are run as debug sessions. The two types of sessions are hardware and software. The processor, target, and platform define the session. When you set up a session, you set the focus on a series of increasingly more specific elements.

Debug target (or target) — The communication channel between VisualDSP++ and a DSP (or group of DSPs). Targets include simulators, emulators, and EZ-KITS. Several targets may be installed on your system. Simulator targets, such as the ADSP-21*xxx* Family Simulator, differ from emulator targets in that the DSP exists only in software.

Default intermediate and output file directories — These directories (folders) are `\Debug` (for the debug configuration) and `\Release` (for the release configuration). By default, VisualDSP++ creates these directories as children of the directory in which the `.DPJ` file is saved, which is called the project's current directory. See also "**Current directory**."

Dependencies — VisualDSP++ uses dependency information to determine which files, if any, are updated during a build. If an included header file is modified, VisualDSP++ builds the source files that include (#include) the header file, regardless of whether the source files have been modified since the previous build.

Dependency files — Usually user files or system header (`*.H`) files, these files are referenced from a source file by a preprocessor `#include` command.

Device — A single DSP. With regard to JTAG emulation and the JTAG ICE Configurator, a device refers to any physical chip in the JTAG chain.

DWARF-2 — Format for debugging source-level assembly code via improved line and symbol information

Editor window — A document window that displays a source file for editing. When an Editor window is active, you can move about within the window and perform typical text editing activities such as searching, replacing, copying, cutting, pasting, and so on.

ELF — Executable and Linking Format

Emulator — An emulator is software that "talks" to a hardware board containing one or more actual DSP chips. An emulator, though it replaces or enhances the DSP chip itself, serves as a hardware equivalent of the actual DSP and shows how your DSP code affects DSP performance characteristics.

Executable file — A file or program that has been written and built in VisualDSP++

Interrupts — Use interrupts to simulate external interrupts in your program. When you use interrupts with watchpoints and streams, your program simulates real-world operation of your DSP system.

JTAG — Joint Test Action Group. This committee is responsible for implementing the IEEE boundary scan specification, enabling in-circuit emulation of ICs.

Keyboard shortcuts — The keyboard provides a quick means of running the commands that are used most often, such as simultaneously typing the keyboard's **Ctrl** and **G** keys (indicated with the symbols **Ctrl+G**) to go to a line in a file.

Librarian — A utility that groups object files into library files. When you link your program, you can specify a library file and the linker automatically links any file in the library that contains a label used in your program. Source code is provided so you can adapt the routines to your needs.

Linear profiling — A debugging feature that samples the target's PC register at every instruction cycle. Linear profiling gives an accurate picture of where instructions were executed, since every PC value is collected. The trade-off, however, is that linear profiling is much slower than statistical profiling. A display of the resulting samples appears in the Linear Profiling Results window, which graphically indicates where the application is spending its time. Simulator targets support linear profiling. See also "**Statistical profiling**."

Linker — The linker creates a single executable program from separately assembled object files. It assigns memory locations to code and data in accordance with a user-defined `.LDF` file, which describes the memory configuration of the target system.

Makefile — VisualDSP++ can export a makefile (make rule file), based on your project options. Use a makefile (`.MAK`) to automate builds outside of VisualDSP++. The output make rule is compatible with the gnumake utility (GNU Make V3.77 or higher) or other make utilities.

Mixed mode — (Versus source mode) one of the two Editor window display formats. Mixed mode displays assembled code after the line of the corresponding C code.

Outdated file — A file that has been edited since the last time it was built

Pinning a window — A technique that statically associates a window to a specific processor

Platform — A configuration of DSPs with which a target communicates. For simulation, a platform is typically one or more DSPs of the same type. For emulation, you specify the platform using the JTAG ICE Configurator, and the platform can be any combination of devices. The platform represents the hardware upon which one or more devices reside. You typically define a platform for a particular target.

For example, if three emulators are installed on your system, a platform selection might be emulator two. Several platforms may exist for a given debug target. For a simulator, the platform defaults to the identical DSP simulator. When the debug target is a JTAG emulator, the platforms are the individual JTAG chains. When the debug target is an EZ-ICE board, the platform is the board in the system on which you wish to focus.

Processor — An individual chip contained on a specific platform within a target. When you create the executable file, the processor is specified in the Linker Description File (`.LDF`) and other source files.

Project — This term refers to the collection of source files and tool configurations used to create a DSP program. Through a project, you can add source files, define dependencies, and specify build options related to producing your output executable program. A project file (`.DPJ`) stores your program's build information. VisualDSP++ enables you to manage projects from start to finish in an integrated user interface. Within the context of a DSP project, you, define project and tool configurations, specify project-wide and individual file options for debug or release modes of project builds, and create source files. VisualDSP++ facilitates easy movement among editing, building, and debugging activities.

Project configuration — This includes all of the settings (options) for the tools used to build a project.

Project file tree display — See "**Project window**."

Project window — This window displays your project's files in a "tree view," which can include folders to organize your project files. Right-clicking on an icon (the project itself, a folder, or a file) opens a menu, providing actions you can perform on the selected item. Double-clicking on the project icon or a folder icon opens or closes the tree list. Double-clicking a file icon opens the file in an Editor window.

Rebuild all — See "**Build**."

Registers — For information on available registers, see the corresponding processor documentation or view the associated Help.

Release configuration — You can accept the default set of options, or you can specify the options you want and save them. The configuration refers to the specified options for all the tools in the tool chain. See also “**Configuration**.”

Reset — This command resets the processor to a known state, clearing processor memory.

Restart — This command sets your program to the first address of the interrupt vector table. Unlike a reset, you do not need to reload memory.

Right-click — This action opens a right-click menu (sometimes called a context menu, pop-up menu, or shortcut menu). The commands that appear depend on the context (what you are doing). Right-click menus provide access to many commonly used commands.

Serial port data — You can automatically transfer serial port (SPORT) data to and from on-chip memory using DMA block transfers. Each serial port offers a time division multiplexed (TDM) multichannel mode.

Session — See “**Debug session**.”

Session name — Although the choice of target, platform, and processor define the session, you may want to further identify the session. You can modify the default session name when you first create the debug session to prevent confusion later. A session name can be any string and can include space characters. There is no limit to the number of characters in a session name, but the **Session List** dialog box can display about 32 characters.

Shortcut — See “**Keyboard shortcuts**.”

Simulator — The simulator is software that mimics the behavior of a DSP chip. Simulators are often used to test and debug DSP code before the DSP chip is manufactured. The simulator executes an executable program in software similar to the way a processor does in hardware. The simulator also simulates the memory and I/O devices specified in the `.LDF` file. VisualDSP++ lets you interactively observe and alter the data in the processor and in memory. The simulator reads executable files. A simulator's response time is slower than that of an emulator.

Source files — The C/C++ language and assembly language files that make up your project. There are other source files that a project uses, such as the `.LDF` file, which contains command input for the linker, and dependency files (data files and header files). View source files in Editor windows.

Source mode — (Versus mixed mode) one of the two Editor window display formats. Source mode displays C code only.

Statistical profiling — A debugging feature that provides a more generalized form of profiling that is well suited to JTAG emulator debug targets. With statistical profiling, VisualDSP++ randomly samples the target processor's program counter (PC) and presents a graphical display of the resulting samples in the Statistical Profiling Results window. This window graphically indicates where the application is spending time. JTAG sampling is completely non-intrusive so the process does not incur additional run-time overhead. See also "**Linear Profiling.**"

Stepping — A technique for moving through source or assembly code to observe instruction execution

Symbols — Labels for sections, subroutines, variables, data buffers, constants, or port names. For more information, refer to the related build tool documentation.

Target — see “**Debug target**.”

Tcl Scripting — VisualDSP++ includes an interpreter for the Tcl (Tool Command Language) scripting language. Analog Devices has extended Tcl version 8.3 with several procedures to access key debugging features. The power of the Tcl language, coupled with Analog Devices’ extensions, allows you to extensively script your work. Tcl command output displays in the Output window’s **Console** tab. The output is also logged to `VisualDSP_log.txt`

Tool Chain — The collection of tools (utilities) used to build a project configuration

Trace — (Sometimes called an execution trace or a program trace) provides a history of program execution. Trace results show how the program arrived at a certain point and show program reads, writes, and memory fetches.

VDK — The VisualDSP++ Kernel (VDK) is a software executive between DSP algorithms, peripherals, and control logic. Refer to the *VisualDSP++ Kernel (VDK) User’s Guide* for details.

VisualDSP++ — An Integrated Development and Debugging Environment (IDDE) for Analog Devices’ DSP development tools.

Watchpoints — For simulation only. Similar to breakpoints, watchpoints stop program execution. Watchpoints, however, allow you to set up conditions, such as a memory read or stack pop. Unlike breakpoints, watchpoints are not attached to a specific address. The program halts when a watchpoint’s conditions are met.

Workspace — You can open multiple windows and place them anywhere you want. After you open and arrange your windows, you can save the layout (configuration) as a workspace setting, which you can recall (load) at a later time. Each debug session’s default workspace is automatically saved when you close the debug session and is automatically restored when you load that session.

C++ Run-Time Libraries

Note: You must run VisualDSP++ to use the C++ run-time libraries. VisualDSP does not support C++.

The C and C++ run-time libraries are collections of functions, macros, and class templates that can be called from source programs. Many functions are implemented in the DSP assembly language.

C and C++ programs depend on library functions to perform operations that are basic to the C and C++ programming languages. These operations include memory allocations, character and string conversions, and math calculations. The libraries also include multiple signal processing functions that ease DSP code development. Using the run-time library simplifies software development by providing code for a variety of common needs.

The compiler provides a broad collection of C functions including those required by the ANSI standard and additional Analog Devices-supplied functions of value for DSP programming. In addition to the Standard C Library, this release of the compiler software includes the Abridged Library, a conforming subset of the Standard C++ Library.

For more information about the algorithms on which many of the C library's math functions are based, refer to the Cody and Waite text *Software Manual for the Elementary Functions* from Prentice Hall (1980).

For more information about the C++ library portion of the ANSI/ ISO Standard for C++, refer to the Plauger text *Draft Standard C++ Library* from Prentice Hall (1994) (ISBN: 0131170031).

File Types

These are the files used to build a project:

Extension	Name	Purpose
.ASM	Assembly source file	Source file comprising assembly language instructions
.C	C source file	Source file comprising ANSI standard C code and Analog Devices extensions
.CPP .CXX .HPP .HXX	C++ source file	Preprocessed compiler files that are inputs to the C/C++ compiler. These files comprise ANSI standard C++ code.
.DPJ	Project file	Contains a description of how your source files combine to build an executable program
.LDF	Linker Description File	This linker command source file is a text file that contains commands for the linker in the linker's scripting language
.S .PP .IS	Intermediate files	Preprocessed assembly files generated by the preprocessor
.DOJ	Assembler Object file	Binary output of the assembler
.DLB	Archiver file	The archiver's binary output in ELF format
.H	Header file	A dependency file used by the preprocessor, and a source file for the assembler and compiler

Extension	Name	Purpose
.DAT	Data file	A dependency file used by the assembler for data initialization
.DXE .SM .OVL .DLO	Debugging files	Binary output files from the linker in ELF/DWARF format
.MAP	Linker Memory Map file	An optional output for the linker. This text file contains memory and symbol information for executable files.
.TCL	Tool Command Language file	A Tcl scripting language file used to script work
.OBJ	Assembled Object file	(Previous releases only, replaced by .DOJ) The output of the assembler
.LST	Listing file	An optional file output by the assembler
.ACH	Architecture file	(Previous releases only, replaced by .LDF)
.TXT	Linker Command-Line file	(Previous releases only, replaced by .LDF) This ASCII text file contains command-line input for the linker
.EXE	Debugging file	(Used in previous releases, replaced by .DXE)
.VDK	VisualDSP++ Kernel Support file	Enables VDK support

Keyboard Shortcuts

VisualDSP++ includes keyboard shortcuts (also called shortcut keys) for the operations that you use most often. These keyboard shortcuts appear in the tables below. You can also run commands by:

- Choosing a command from a drop-down menu on the menu bar
- Clicking a toolbar button
- Right-clicking from a particular context, such as from the Project window
- Clicking a configured user tool
- Clicking a button within a dialog box
- Running a Tcl script (from the **File** menu or Output window)
- Choosing a command from the application's control menu

Working with Files

Use the following keyboard shortcuts when working with files:

Action	**Shortcut**
Open a new file	**Ctrl+N**
Open an existing file	**Ctrl+O**
Save a file	**Ctrl+S**
Print a file	**Ctrl+P**
Go to the next window	**F6**
Go to the previous window	**Shift+F6**

Moving within a File

Use the following keyboard shortcuts to move within a file:

Action	Shortcut
Move the cursor to the left one character	**Left Arrow** (←)
Move the cursor to the right one character	**Right Arrow** (→)
Move the cursor to the beginning of the file	**Ctrl+Home**
Move the cursor to the end of the file	**Ctrl+End**
Move the cursor to the beginning of the line	**Home**
Move the cursor to the end of the line	**End**
Move the cursor down one line	**Down Arrow** (↓)
Move the cursor up one line	**Up Arrow** (↑)
Move the cursor one page down	**Page Down**
Move the cursor one page up	**Page Up**
Move the cursor right one tab	**Shift**
Move the cursor left one tab	**Shift+Tab**
Move the cursor left one word	**Ctrl+Left Arrow** (←)
Move the cursor right one word	**Ctrl+Right Arrow** (→)
Go to the next bookmark	**F2**
Go to a line	**Ctrl+G**
Find text	**Ctrl+F**
Find the next occurrence of text	**F3**

Cutting, Copying, Pasting, Moving Text

Use the following keyboard shortcuts to edit text:

Action	**Shortcut**
Copy text	**Ctrl+C** or **Ctrl+Insert**
Copy text	Select with cursor and **Ctrl**+drag
Cut text	**Ctrl+X** or **Shift+Delete**
Delete text	**Delete** (selection or forward)
Delete text	**Backspace** (selection or backward)
Move text	Select with cursor and drag
Move selected text right one tab	**Tab**
Move selected text left one tab	**Shift+Tab**
Paste text	**Ctrl+V** or **Shift+Insert**
Undo the last edit	**Ctrl+Z** or **Alt+Backspace**
Redo an edit command	**Shift+Ctrl+Z**
Replace text	**Ctrl+H** or **Ctrl+R**

Selecting Text within a File

Use the following keyboard shortcuts to select text within a file:

Action	Shortcut
Select all text in a file	**Ctrl+A**
Select the character on the left	**Shift+Left Arrow** (←)
Select the character on the right	**Shift+Right Arrow** (→)
Select all text to the beginning of the file	**Shift+Ctrl+Home**
Select all text to the end of the file	**Shift+Ctrl+End**
Select all text to the beginning of the line	**Shift+Home**
Select all text to the end of the line	**Shift+End**
Select all text to the line below	**Shift+Down Arrow** (↓)
Select all text to the line above	**Shift+Up Arrow** (↑)
Select all text to the next page	**Shift+PgDn**
Select all text to the above page	**Shift+PgUp**
Select the word on the left	**Shift+Ctrl+Left Arrow** (←)
Select the word on the right	**Shift+Ctrl+Right Arrow** (→)
Select, by column	Place cursor, press and hold down **Alt** and drag the cursor (selects by column-character instead of by line-character)

Working with Bookmarks in an Editor Window

Use the following keyboard shortcuts when working with bookmarks in an Editor window:

Action	Shortcut
Toggle a bookmark	**Ctrl+F2**
Go to next bookmark	**F2**

Building Projects

Use the following keyboard shortcuts to build projects:

Action	Shortcut
Build the current project	**F7**
Build only the current source file	**Ctrl+F7**

Using Keyboard Shortcuts for Program Execution

Use the following keyboard shortcuts for program execution:

Action	**Shortcut**
Load a program	**Ctrl+L**
Reload a program	**Ctrl+R**
Dump to file	**Ctrl+D**
Run	**F5**
Run to cursor	**Ctrl+F10**
Halt	**Shift+F5**
Step over	**F10**
Step into	**F11**
Step out of	**Alt+F11**
Halt a Tcl script	**Ctrl+H**

Working with Breakpoints

Use the following keyboard shortcuts when working with breakpoints:

Action	Shortcut
Open the **Breakpoints** dialog box	**Alt+F9**
Enable/disable a breakpoint	**Ctrl+F9**
Toggle (add or remove) a breakpoint	**F9**

Obtaining Online Help

Use the following keyboard shortcuts to obtain online Help:

Action	Shortcut
View online Help for the selected object	**F1**
Obtain context-sensitive Help for controls (buttons, fields, menu items)	**Shift+F1**

Miscellaneous

Use the following keyboard shortcuts for windows and workspaces:

Action	Shortcut
Refresh all windows	**F12**
Select workspace 1 through 10	**Alt+1 … Alt+0**

IDDE Command-Line Parameters

You can invoke VisualDSP++ from a DOS command line.

Syntax:

```
idde.exe [-f script_name]
         [-s session_name]
         [-p project_name]
```

Note: Specify the full path to idde.exe.

Parameters:

Item	Description
`-f script_name`	Loads and executes the Tcl script specified by script_name. Use this parameter to automate regression tests. You can also manipulate VisualDSP++ by running a Tcl script from a library of common Tcl commands that you create. If an error is encountered while executing this script, VisualDSP++ automatically exits.
`-s session_name`	Specifies the session to which VisualDSP++ connects when it starts. The session must already exist. This parameter is useful when you are debugging more than one target board. Having multiple shortcuts to idde.exe allows you to run a different session. This overrides VisualDSP++'s default behavior of always connecting to the last session.
`-p project_name`	Specifies the project to load at startup. The project must already exist.

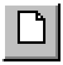
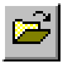
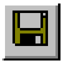
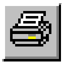
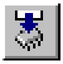
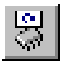
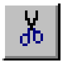
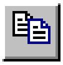
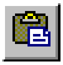
Examples:

```
idde.exe -f "c:\\scripts\\myscript.tcl"

idde.exe -s "My 21160 JTAG Emulator Session"

idde.exe -p "c:\\projects\\myproject.dpj"
```

Toolbar Buttons

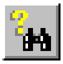
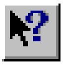
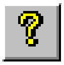
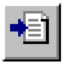
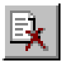
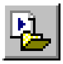
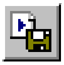
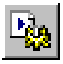
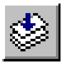
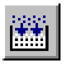
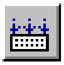
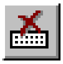
The toolbar, which comprises separate toolbars, provides quick mouse access to commands. The toolbar is a Windows docking bar. You can move it to different areas of the screen by dragging it to the selected location.

The following table describes the toolbar buttons.

Button	Purpose
	Creates a new document
	Opens an existing document
	Saves the active document or template with the same name
	Prints the active document
	Loads a program into the target
	Reloads the most recent program into the target
	Cuts selected data from the document and store it on the clipboard
	Copies the selection to the clipboard
	Pastes the contents of the clipboard at the insertion point

Button	Purpose
	Undoes previous edit command (multi-level undo)
	Redo. Undoes the previous Undo command (multi-level redo)
	Find a text block in an Editor window
	Find again. Repeats the previous find command.
	Replaces the selected text with other text
	Searches through files for text or regular expressions
	Go to. Moves to the specified location
	Displays the current source file
	Toggles the bookmark at selected line in the active Editor window
	Goes to the next book-marked line in the Editor window
	Goes to the previous book-marked line in the Editor window
	Clears all bookmarks in the Editor window

Toolbar Buttons

Button	Purpose
	Opens the online Help to the **Search** tab
	Provides context-sensitive Help for a button command, or portion of VisualDSP++
	Opens the **About VisualDSP++** dialog box
	Adds a source file to the project
	Removes a source file from the project
	Opens an existing project
	Saves the open project
	Opens the **Project Options** dialog box, in which to specify project options
	Builds the selected source file
	Builds the project (update outdated files)
	Builds all files in the project
	Stops the current project build

Button	Purpose
	Arranges windows as tall non-overlapping tiles
	Arranges windows as wide non-overlapping tiles
	Arranges windows so they overlap
	Closes all open windows
	Refreshes all the debugging windows
	Run. Starts or continues the current program
	Restarts the current program
	Stops the current program
	Resets the target
	Toggles a breakpoint for the current line
	Clears all current breakpoints
	Enables or disables one breakpoint

Toolbar Buttons

Button	Purpose
✋	Disables all breakpoints
📝	Steps one line
{}↷	Steps over the current statement
{}↷	Steps out of the current function
→{}	Runs the program to the line containing the cursor
👓	Opens the Expression window
🗔	Opens the Locals window
🗔	Opens the Call Stack window
🗔	Opens the Disassembly window
🔨1	User tool (one of ten). Runs the command associated with the user tool
1	Workspace (one of ten). Opens the associated workspace

Text Operations

Regular Expressions in Find/Replace Operations

Regular Expressions vs. Normal Searches

Normally, when you search for text, the search mechanism scans for an exact, character-by-character match of the search string, which does not have to be an entire word. Every character in the search string is examined. If there are embedded spaces, for instance, the exact number is matched.

Regular expression matching provides much more flexibility and power than a normal search. A regular expression can be a simple string, which yields the same matches as normal searches. Some characters in a regular expression string, however, have special interpretations, which provide greater flexibility.

For example, with regular expression matching, you can find the following:

- All occurrences of either `hot` or `cold`
- Occurrences of `for` followed by a left parenthesis, with any number of intervening spaces
- A `;` (semicolon) only when it is the last character on a line
- The string `ADSP` followed by a sequence of digits

You can use a regular expression as the search pattern for replacement. In that case, there are ways to identify and recover the variable portions of the matched strings.

Specific Special Characters

Regular expressions assign special meaning to the following characters.

Note: If you need to match on one of these characters, you must escape it by preceding it with a backslash (`\`). Thus, `\^` matches the `^` character, yet `^` matches the beginning of the line.

Character	Description
^	A caret matches the beginning of the line
$	A dollar sign matches the end of the line
.	A period (.) matches any character
[abc]	A bracketed sequence of characters matches one character, which may be any of the characters inside the brackets. Thus, `[abc]` matches an a, b, or c.
[0-9]	This shorthand form is valid within the sequence brackets. It specifies a range of characters, from first through last, exactly as if they had been written explicitly. Ranges may be combined with explicit single characters and other ranges within the sequence. Thus, `[-+.0-9]` matches any constituent character of a signed decimal number; and `[a-zA-Z0-9_]` matches a valid identifier character, either lowercase or uppercase. Ranges follow the ordering of the ASCII character set.
[^abc] [^0-9]	A caret (^) that is the first character of a sequence matches all characters except for the characters specified after the caret.

Character	Description
(*material*)	The material inside the parentheses can be any regular expression. It is treated as a unit, which can be used in combination with other expressions. Parenthesized material is also assigned a numerical tag, which may be referenced by a replace operation.

Special Rules for Sequences

The normal special character rules of regular expressions do not apply within a bracketed sequence. Thus, `[*&]` matches an asterisk or ampersand.

Certain characters have special meaning within a sequence. These include `^` (not), `-` (range), and `]` (end of sequence). By placing these characters appropriately, you can specify these characters to be part of the sequence.

To search for a right bracket character, place `]` as the first character of the search string. To search for a hyphen character, place `-` as the first character of the search string after `]`, if present. Place a caret anywhere in the search string except at the front, where it means "not."

Repetition and Combination Characters

The following characters extend the meaning of the immediately preceding item, which may be a single character, a sequence in braces, or an entire regular expression in parentheses:

Character	Description
*	An asterisk matches the preceding any number of times, including none at all. Thus, `ap*le` matches `apple`, `aple`, `appppple` and `ale`. For example, `^ *void` matches only when `void` occurs at the beginning of a line and is preceded by zero or more spaces.
+	A plus character matches the preceding any number of times, but at least one time. Thus, `ap+le` matches `apple` and `aple`, but does not match `ale`.
?	A question mark matches the preceding either zero or one time, but not more. Thus, `ap?le` matches `ale` and `aple`, but nothing else.
|	The pipe character (|) matches either the preceding or following item. For example, `(hot)|(cold)` matches either `hot` or `cold`. Note: Spaces are characters. Thus, `(hot) | (cold)` matches “`hot `" or "` cold`".

Match Rules

If multiple matches are possible, the `*`, `+`, and `?` characters match the longest candidates. The `|` character matches the left-hand alternative first.

For more information, see the many reference texts available on this topic, such as *Mastering Regular Expressions, Powerful Techniques for Perl and Other Tools* by Jeffrey E. F. Friedl, (c) 1997 O'Reilly & Associates, Inc.

Tagged Expressions in Replace Operations

Use a tagged expression as part of the string in the **Replace** field for a replace operation.

Note: You must enclose a tagged expression between parenthesis characters.

In the **Replace** field, the following operators represent tagged expressions from the **Find** field:

Find Field	Replace Field
Entire matched sub string	\0
Tagged expressions within parentheses `()` from left to right	\1 \2 \3 \4 \5 \6 \7 \8 \9
Entire match expression	&

The replace expression can specify an ampersand (`&`) character, meaning that the `&` represents the sub-string that was found. For example, if the sub-string that matched the regular expression is "`abcd`", a replace expression of "`xyz&xyz`" changes it to "`xyzabcdxyz`". The replace expression can also be expressed as "`xyz\0xyz`", where the "`\0`" indicates a tagged expression representing the entire sub-string that was matched. Similarly, you can have another tagged expression represented by "`\1`", "`\2`".

Note: Although the tagged expression 0 is always defined, the tagged expressions 1, 2, and so on, are defined only when the regular expression used in the search has enough sets of parenthesis. Here are some examples:

<table>
<tr><th>String</th><th>Search</th><th>Replace</th><th>Result</th></tr>
<tr><td>Mr.</td><td>(Mr)(\.)</td><td>\1s\2</td><td>Mrs.</td></tr>
<tr><td>abc</td><td>(a)b(c)</td><td>&-\1-\2</td><td>abc-a-c</td></tr>
<tr><td>bcd</td><td>(a|b)c*d</td><td>&-\1</td><td>bcd-b</td></tr>
<tr><td>abcde</td><td>(.*)c(.*)</td><td>&-\1-\2</td><td>abcde-ab-de</td></tr>
<tr><td>cde</td><td>(ab|cd)e</td><td>&-\1</td><td>cde-cd</td></tr>
</table>

Comment Start and Stop Strings

You use comment start/stop strings for comment highlighting colors. You are allowed to set two types of comment start/stop strings for each file type.

These are example strings for typical file types:

- Start Comment Strings

String	Purpose
!	Starts an assembly style, single-line comment
/*	Starts a C/C++ style, multi-line comment
//	Starts a C/C++ style, single-line comment

- Stop Comment Strings

String	Purpose
Carriage return	(C and Assembly) Ends a single-line comment
*/	Ends a C/C++ style, multi-line comment
(blank)	Ends a C/C++ style, single-line comment