

C TCL SCRIPTING

In This Appendix

This appendix contains the following topics:

- [“Overview” on page C-1](#)
- [“Tcl Output and Command Issuance” on page C-2](#)
- [“Tcl Command Listings and Examples” on page C-4](#)
- [“Tcl Command Reference” on page C-16](#)
- [“Additional Tcl Resources” on page C-62](#)

Overview

VisualDSP++ includes an interpreter for the Tool Command Language (Tcl) scripting language. This well-documented C-like language, developed by UC Berkely researchers, provides an excellent means of scripting repeated sequences of debugging operations. You can use this powerful language to develop comprehensive test applications of DSP systems.

Analog Devices has enhanced Tcl version 8.3 with several procedures to access key debugging features. Use the power of the Tcl language with Analog Devices’ extensions to script your work in VisualDSP++.

Tcl Output and Command Issuance

You can view the output of Tcl commands in the Output window's Console tab. Tcl output is logged to `VisualDSP_log.txt`, which by default, is located in the directory:

```
C:\Program Files\Analog Devices\VisualDSP\Data\
```

View this file to analyze the Tcl output.

Issue a command or try a simple command by typing it in the Output window's Console tab. For extensive scripting, use the following methods:

Command-Line Issuance

Load a script from a DOS command window by typing:

```
idde -f filename
```

Optionally, add `-s` and the *session name* to specify a previously created session. When no session name is specified, the last session is used.



If the script encounters an error during execution, VisualDSP++ automatically exits.

Output Window Issuance

Load a script from the Output window's Console tab by typing the command:

```
source filename
```

Note: Similar to C/C++, Tcl uses a backslash (\) as its escape character. When you specify paths in the Windows environment, you must escape the escape character. For example:

```
source c:\\my_dir\\my_subdir\\my_file.tcl
```

Note: You can also use forward slashes to delimit directories in a path; for example:

```
source c:/my_dir/my_subdir/my_file.tcl
```

Command execution is deferred until a line is typed without a trailing backslash. This feature permits the entry of an entire block of code (or entire Tcl procedures) for the Tcl interpreter to evaluate at once.

Menu Issuance

You can quickly issue Tcl scripts of frequent use. From the File menu, choose Recent Tcl Scripts, and then select the Tcl script.

Editor Window Issuance

From an open Editor window that contains a Tcl script, right-click and choose Source Tcl Script.

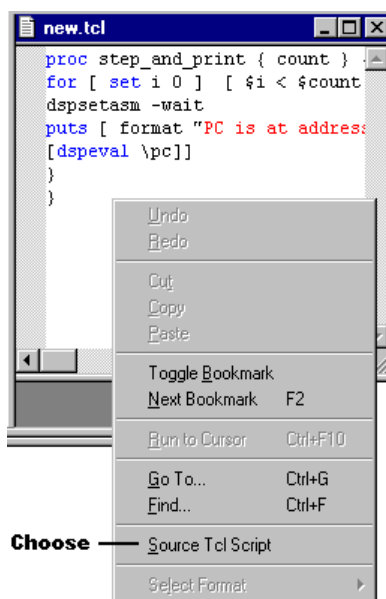


Figure C-1. Selecting Source Tcl Script

User Tool Issuance

Click a toolbar user tool or choose a user tool from the Tools menu.

Tcl Command Listings and Examples

This section provides the following information:

- [“Target Query and Manipulation Commands” on page C-4](#)
- [“GUI Manipulation Commands” on page C-7](#)
- [“Project Build and Maintenance Commands” on page C-8](#)
- [“Tcl Script Example” on page C-9](#)
- [“Example Regression Test” on page C-10](#)

Target Query and Manipulation Commands

Use these Tcl commands to query the target and manipulate values.

Table C-1. Target Query and Manipulation Command Summary

Command	Description
“dspsetbreak” on page C-53	Sets (inserts) a breakpoint
“dspcancelbreak” on page C-18	Cancels (deletes) a breakpoint
“dspgetbreak” on page C-25	Returns information about the breakpoint
“dspeval” on page C-23	Evaluates an expression
“dspgetmemblock” on page C-27	Fetches a block of memory

Table C-1. Target Query and Manipulation Command Summary

“dspgetmeminfo” on page C-29	Gets information about types of memory. This information is used by other commands.
“dspgetprocessors” on page C-30	Gets the names of the processors in a multiprocessor debug session
“dspgetstate” on page C-30	Gets the current state of a processor
“dspreset” on page C-49	Gets the C-language software stack for a processor
“dsphalt” on page C-32	Requests a halt of the processor
“dspload” on page C-33	Loads a file to the target
“dsplookupline” on page C-34	Looks up the start and end address corresponding to a line in a file
“dsplookupsymbol” on page C-35	Looks up the address of a symbol identified by a label
“dspplotrotate” on page C-37	Rotates a waterfall plot by azimuth and elevation
“dspplotwin” on page C-38	Configures and displays a plot in a Plot window
“dspreset” on page C-49	Sends a reset message to the target
“dsprestart” on page C-50	Sends a restart message to the target

Tcl Command Listings and Examples

Table C-1. Target Query and Manipulation Command Summary

“dsp_{run}” on page C-51	Sends a run message to the target
“dsp_{set}” on page C-52	Evaluates an expression and assigns the value to another expression
“dsp_{setmemblock}” on page C-55	Sets a block of memory
“dsp_{setswstack}” on page C-57	Changes the debug focus
“dsp_{stepasm}” on page C-58	Steps the target a single disassembly step
“dsp_{stepin}” on page C-59	Steps the target a single source language step
“dsp_{stepout}” on page C-60	Steps the target out of the current subroutine in a source language
“dsp_{stepover}” on page C-61	Steps the target a single source language step, skipping over any subroutine calls
“dsp_{waitforhalt}” on page C-62	Delays further execution until the target halts

GUI Manipulation Commands

These Graphic User Interface (GUI) manipulation commands create windows and menu items without using the GUI.

Table C-2. GUI Manipulation Command Summary

Command	Description
“dspaddmenuitem” on page C-16	Adds a menu item (command) to the menu bar
“dspcheckmenuitem” on page C-19	Queries or sets the “checked” attribute for a menu item
“dspclickmenuitem” on page C-20	Simulates a mouse click of a menu item
“dspdeletemenuitem” on page C-21	Deletes a menu item
“dspenablemenuitem” on page C-22	Queries or sets the “enabled” attribute of a menu item
“dspmemorywin” on page C-36	Displays a Memory window
“dspregisterwin” on page C-48	Creates a custom register window

Project Build and Maintenance Commands

Use these Tcl commands to operate at the project level.

Table C-3. Project Build and Maintenance Command Summary

Command	Description
“dspprojectload” on page C-46	Loads the project into VisualDSP++
“dspprojectbuild” on page C-43	Builds the currently loaded project configuration
“dspprojectinfo” on page C-45	Returns information about the currently loaded project
“dspprojectaddfile” on page C-41	Adds a file to the current project
“dspprojectaddfolder” on page C-42	Adds a folder to the current project
“dspprojectremovefile” on page C-46	Removes a file from the current project
“dspprojectremovefolder” on page C-47	Removes a folder from the current project
“dspsetbreak” on page C-53	Closes the currently loaded project

Tcl Script Example

The following example shows how to create and run a Tcl script that prints to the Output window's Console tab.

This Tcl procedure, named `step_and_print`, single-steps through your assembly code a specified number of times. When you run the script, you must supply the number of steps (the count).

To create the Tcl script:

1. Use a text editor to type the following code.

```
proc step_and_print { count } {
    for { set i 0 } { $i < $count } { incr i } {
        dspstepasm -wait
        puts [ format "PC is at address 0x%x\n" [dspeval \ $pc]]
    }
}
```

2. Save to C:\Temp\test.tcl.

To run the new Tcl script

1. From the Output window's Console tab, type the `Tcl source` command followed by the path and file name of the new Tcl script.

```
>source C:\\Temp\\test.tcl
```

Note: Use double backslash characters.

2. Press Enter.

The Tcl script loads.

3. In the Console tab, call the function and supply the step count, for example:

```
step_and_print 10
```

4. Press Enter.

Tcl Command Listings and Examples

The function executes. The program counter single-steps ten times (the step count in this example) and halts.

```
{bmct tcl_2.bmp}
```

Note: Output resulting from commands entered in the Output window's Console tab is saved to `VisualDSP_log.txt`.

Example Regression Test

Use Tcl with Analog Devices' extensions to build sophisticated behaviors. Below is the implementation of a regression test used by Analog Devices software developers to test the VisualDSP++'s debugging capability. This particular test ensures that function parameters evaluate correctly.

Note: At one time, this test was performed manually with the GUI, taking several minutes. It is now tested with a Tcl script and takes seconds, a considerable productivity booster.

Two procedures are defined to help implement this test. The `goto_line` procedure runs the program to a certain line number. The `assert` procedure trips an error if a given expression evaluates to zero (false).

Tcl Example – Performing a Regression Test

```
proc assert { e } {  
    if { 0 == $e } {  
        error "Assertion Failed!"  
    }  
}  
  
proc goto_line { file line } {  
    # Lookup the address corresponding to {file, line} pair.  
    # The return value from dsplookupline is a list of start  
    # and end address (we use lindex to extract the car of  
    # the list).  
    dspsetbreak [ lindex [ dsplookupline $file $line ] 0 ] \  
        -temporary  
    dsprun
```

```

        dspwaitforhalt
    }

    dspload argtest.dxe
    goto_line main.c 126

    assert [ expr 0x56          == [ dspeval "cNum" ] ]
    assert [ expr 0x7890        == [ dspeval "sNum" ] ]
    assert [ expr 0x1234        == [ dspeval "iNum" ] ]
    assert [ expr 0xdeaddead    == [ dspeval "lNum" ] ]
    assert [ expr 1.234         == [ dspeval "fNum" float ] ]
    assert [ expr 5.678         == [ dspeval "dNum" float ] ]

    goto_line main.c 58

    assert [ expr 0x56          == [ dspeval "cNum" ] ]
    assert [ expr 0x7890        == [ dspeval "sNum" ] ]
    assert [ expr 0x1234        == [ dspeval "iNum" ] ]
    assert [ expr 0xdeaddead    == [ dspeval "lNum" ] ]
    assert [ expr 1.234         == [ dspeval "fNum" float ] ]
    assert [ expr 5.678         == [ dspeval "dNum" float ] ]

    assert [ expr 0x56          == [ dspeval "g_cNum" ] ]
    assert [ expr 0x7890        == [ dspeval "g_sNum" ] ]
    assert [ expr 0x1234        == [ dspeval "g_iNum" ] ]
    assert [ expr 0xdeaddead    == [ dspeval "g_lNum" ] ]
    assert [ expr 1.234         == [ dspeval "g_fNum" float ] ]
    assert [ expr 5.678         == [ dspeval "g_dNum" float ] ]

    assert [ expr 0x56          == [ dspeval "c" ] ]
    assert [ expr 0x7890        == [ dspeval "s" ] ]
    assert [ expr 0x1234        == [ dspeval "i" ] ]
    assert [ expr 0xdeaddead    == [ dspeval "l" ] ]
    assert [ expr 1.234         == [ dspeval "f" float ] ]
    assert [ expr 5.678         == [ dspeval "d" float ] ]

    goto_line main.c 142

    assert [ expr 0xdeaf == [ dspeval "iNum" ] ]

    dsprun
    dspwaitforhalt

    exit

```

Tcl Command Listings and Examples

For reference, the source code for `main.c` is presented here.

Note: `ScareCompiler()` is an auxiliary stub function used to (intentionally) suppress compiler optimizations.

```
#include <stdio.h>

extern void ScareCompiler(void *pv);

char    g_cNum;
short   g_sNum;
int     g_iNum;
long    g_lNum;
float   g_fNum;
double  g_dNum;

int LotsOfArgs(char c, short s, int i, long l, float f, double
d)
{
    char    cNum;
    short   sNum;
    int     iNum;
    long    lNum;
    float   fNum;
    double  dNum;

    cNum = c;
    sNum = s;
    iNum = i;
    lNum = l;
    fNum = f;
    dNum = d;

    ScareCompiler((void*)&cNum);
    ScareCompiler((void*)&sNum);
    ScareCompiler((void*)&iNum);
    ScareCompiler((void*)&lNum);
    ScareCompiler((void*)&fNum);
    ScareCompiler((void*)&dNum);

    g_cNum = cNum;
    g_sNum = sNum;
    g_iNum = iNum;
    g_lNum = lNum;
    g_fNum = fNum;
    g_dNum = dNum;
```

```

ScareCompiler((void*)&g_cNum);
ScareCompiler((void*)&g_sNum);
ScareCompiler((void*)&g_iNum);
ScareCompiler((void*)&g_lNum);
ScareCompiler((void*)&g_fNum);
ScareCompiler((void*)&g_dNum);

/*****
Set a break here verify that the locals, arguments and globals
look ok. They should have the same values as the locals in
main().
    Examples:
        g_cNum, cNum, & c should all equal 0x56
        g_sNum, sNum, & s should all equal 0x7890
*****/

if(cNum != 0x56)
    printf("As If!\n");
if(sNum != 0x7890)
    printf("As If!\n");
if(iNum != 0x1234)
    printf("As If!\n");
if(lNum != 0xdeaddead)
    printf("As If!\n");
if(fNum != 1.234)
    printf("As If!\n");
if(dNum != 5.678)
    printf("As If!\n");
if(cNum && sNum && iNum && lNum && fNum && dNum)
    return (0xdeaf);
else
    return (0xbad);
}

/*****
TestArg test program
This tests the debugger and the debug information
generated by the compiler for the basic types. It tests
that the information is ok regardless of whether the
variable is a local or an argument to a function.
To use this file:
1. Load the argtest executable
2. Load the argtest layout file

```

Tcl Command Listings and Examples

3. Deposit break points at the specified locations
4. Enter the following into the expressions window
g_cNum
g_sNum
g_iNum
g_lNum
g_fNum
g_dNum
5. Run evaluating correctness based on the comments in the code.

```
*****/  
  
void main(void)  
{  
    char    cNum;  
    short   sNum;  
    int     iNum;  
    long    lNum;  
    float   fNum;  
    double  dNum;  
  
    cNum = 0x56;  
    sNum = 0x7890;  
    iNum = 0x1234;  
    lNum = 0xdeaddead;  
    fNum = 1.234;  
    dNum = 5.678;  
  
    ScareCompiler((void*)&cNum);  
    ScareCompiler((void*)&sNum);  
    ScareCompiler((void*)&iNum);  
    ScareCompiler((void*)&lNum);  
    ScareCompiler((void*)&fNum);  
    ScareCompiler((void*)&dNum);  
}
```

```

/*****
Set a break here
verify that the locals look exactly
as they were initialized. The ScareCompiler
function does nothing to the variables.
*****/

if(cNum > 0x56)
    printf("No way!\n");
if(sNum > 0x7890)
    printf("No way!\n");
if(iNum > 0x1234)
    printf("No way!\n");
if(lNum > 0xdeaddead)
    printf("No way!\n");
if(fNum > 1.234)
    printf("No way!\n");
if(dNum > 5.678)
    printf("No way!\n");

/* This should return 0xdead */
iNum = LotsOfArgs(cNum, sNum, iNum, lNum, fNum, dNum);
// Set a break here check that iNum is 0xdead
if(iNum == 0xbad)
    printf("That's just wrong\n");
else
    printf("No problems\n");
}

```

Tcl Command Reference

The following are the descriptions of all Tcl commands with their syntax and arguments:

dspaddmenuitem

Syntax:

```
dspaddmenuitem menuItem callback  
                [-head] [-info value] [-help value]?
```

Purpose:

This Tcl command adds a menu item to the menu bar.

Returns: This command returns a cookie, representing the identifier for the menu item. The cookie is used in calls to `dspcheckmenuitem`, `dspenablemenuitem`, and `dspdeletemenuitem`. The cookie is passed to the callback function, allowing a single callback function to service multiple menu items.

Arguments:

`menuItem`

Specifies the path to the menu item.

It is of the format `MENU:SUBMENU:SUBMENU:ITEM`.

This specifies the menu tree that needs to be traversed to access the menu item.

`callback`

Specifies the Tcl procedure called when the menu item is clicked.

This function must be of the form:

```
proc function_name { id } { body }
```

The `id` parameter to this callback function is the cookie for the menu item (see Returns above).

-head

Specifies that the menu item is to be pre-pended to the menu; otherwise the menu item is appended to the menu.

-info value

Specifies an information string to be displayed in the application's status bar when the menu item has focus.

-help value

Specifies a callback to a help function. This callback has the same format as the function callback described above.

Example:

The following script creates a menu item that outputs a message when you choose the menu item.

```
proc callback { id } {
    puts [ format "Menu ID %d clicked.\n" $id ]
}

set id [ dspaddmenuitem "Custom:Menu #1" callback \
    -info "Demo menu item" ]
puts [ format "Menu ID %d installed.\n" $id ]
```

Tcl Command Reference

dspcancelbreak

Syntax:

```
dspcancelbreak [-processor value] id
```

Purpose:

This Tcl command cancels (deletes) the breakpoint identified by `id`.

Returns: `id`

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`id`

An identifier from a previous call to [“dspsetbreak” on page C-53](#). A breakpoint’s identifier can also be determined by using [“dspgetbreak” on page C-25](#).

Example:

```
# Cancel all breakpoints
foreach bp [dspgetbreak] { dspcancelbreak [lindex $bp 0] }
```

dspcheckmenuitem

Syntax:

```
dspcheckmenuitem id [value]
```

Purpose:

This Tcl command queries or sets the checked attribute of the menu item identified by `id`.

Returns: The string “checked” or “unchecked”

Arguments:

`id`

Obtained from a previous call to `dspaddmenuitem`.

`value`

Specifies the new value for this attribute. Valid values are `checked` and `unchecked`.

If `value` is not specified, the value of the attribute is not altered.

Tcl Command Reference

dspclickmenuitem

`dspclickmenuitem menuItem`

Purpose:

This Tcl command simulates a mouse click on a menu item.

Returns: 1 on success, 0 otherwise (that is, when the menu item does not exist).

Arguments:

`menuItem`

Specifies the menu item to click.

It is of the format `MENU:SUBMEU:SUBMENU:ITEM`, specifying the menu tree to be traversed to access the menu item.

Example:

This command refreshes the window.

```
> dspclickmenuitem Window:Refresh
```

dspdeletemenuitem**Syntax:**

```
dspdeletemenuitem id
```

Purpose:

This Tcl command deletes the menu item specified by *id*.

Returns: nothing

Arguments:

id

Obtained from a previous call to `dspaddmenuitem` (see [“dspaddmenuitem” on page C-16](#)).

Tcl Command Reference

dspenablemenuitem

Syntax:

```
dspenablemenuitem id [value]
```

Purpose:

This Tcl command queries or sets the enabled attribute of the menu item identified by `id`.

Returns: the string `enabled`, `disabled`, or `grayed`.

Arguments:

`id`

Obtained from a previous call to `dspaddmenuitem` (see [“dspaddmenuitem” on page C-16](#)).

`value`

Specifies the new value for this attribute.

Valid values are `enabled`, `disabled`, or `grayed`.

If `value` is not specified, the value of the attribute is not altered.

dspeval

Syntax:

```
dspeval [-processor value] expr [format]
```

Purpose:

This Tcl command evaluates an expression specified by `expr`. Valid forms of expressions are the expressions that can be accepted by the Expressions window.

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`expr`

The expression to be evaluated

`format`

Specifies the format used to format the output.

Valid values are hexadecimal (default), integer, unsigned integer, float, and octal.

Escape Character:

When using the dollar sign (\$) character in an expression, escape this character with a backslash (\).

Note: The \$ is the variable signifier in Tcl.

Tcl Command Reference

Examples:

The following example returns the value of the expression or an error message if a problem was encountered:

```
> dspeval "\$PC"
```

The following example evaluates a C expression:

```
> dspeval "&my_array[5]"
```

The following example fetches the opcode at the PC:

```
> dspeval "\$PM (\$PC)"
```

dspgetbreak

Syntax:

```
dspgetbreak [-processor value] [id]
```

Purpose:

This Tcl command returns a list, containing information about the breakpoint identified by `id`.

The list consists of these elements:

- The `id` of the breakpoint
- The software overlay or hardware page in which the breakpoint is located (-1, if none)
- The address of the breakpoint
- The source file of the breakpoint ("", if unknown)
- The line number of the breakpoint (zero if unknown)
- "temporary" or "permanent"
- "enabled" or "disabled"
- "placed" or "unplaced"
- The skip count
- The test expression ("", if unknown)

Note: If `id` is omitted, a list of all breakpoint information is returned.

Tcl Command Reference

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`id`

A value previously returned by [“dspsetbreak” on page C-53](#).

dspgetmemblock

Syntax:

```
dspgetmemblock [-processor value]
                memory
                start
                count
                [-stride value]
                [-format value]
```

Purpose:

This Tcl command fetches a block of memory and returns a list comprised of the memory fetched. Each element of the list represents the value of a single address.

Arguments:

-processor value

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

value specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

memory

Specifies the memory from which to fetch the block. One of the strings outputted by [“dspgetmeminfo” on page C-29](#).

start

Specifies the first address to fetch in the block

count

Specifies the total number of addresses to fetch

Tcl Command Reference

`-stride value`

Specifies the distance between memory locations. If `value` is omitted, 1 is used.

`-format value`

Specifies the format used to format the output.

Valid values are hexadecimal (default), integer, unsigned integer, float, and octal.

Values vary from target to target. The default is hexadecimal.

Example:

```
> set pm [ lindex [ lindex [dspgetmeminfo] 0 ] 0 ]  
      Program(PM) Memory  
> dspgetmemblock $pm 0x20004 3 -format "Assembly"  
      {nop;} {jump __lib_start;} {nop;}
```

dspgetmeminfo

Syntax:

```
dspgetmeminfo [-processor value]
```

Purpose:

This Tcl command returns information about the types of memory within the target and returns a list of memories. Each element of the list is itself a list comprised of two elements:

- An ASCII string representing the canonical name of the memory
- The width of the memory in bits

The returned information is used by other commands to identify a particular memory by its name.

Arguments:

```
-processor value
```

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

Example:

The ADSP-2191 simulator's output:

```
> dspgetmeminfo
{ "PM" 24 } { "DM" 16 } { "IOM" 16 }
```

Tcl Command Reference

dspgetprocessors

Syntax:

```
dspgetprocessors
```

Purpose:

This Tcl command returns the names of the processors in a multiprocessor debug session. For a single-processor session, an empty list is returned.

These names are used in the `-processor` argument of other Tcl commands.

dspgetstate

Syntax:

```
dspgetstate [-processor value]
```

Purpose:

This Tcl command returns the current state of a processor.

Returns: One of the following strings: `reset`, `running`, `stepping`, `halted`, `loaded`, or `unknown`

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

dspgetswstack

Syntax:

```
dspgetswstack [-processor value]
```

Purpose:

This Tcl command returns the C-language software stack for a processor.



Do not confuse this stack with the internal hardware stack found on some targets.

Returns: This command returns a list of frames. Each frame is a list comprised of a name (C function) and a cookie value. The cookie value changes the debug focus to a different frame (see “[dspsetswstack](#)” on [page C-57](#)). If a frame currently has the debug focus, a third item in the list is the string focus. One frame only may have focus at a given time.

If the stack cannot be found or identified because the program is in an assembly language module (or no debug information is present), a null list is returned.

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

Tcl Command Reference

dsphalt

Syntax:

```
dsphalt [-processor value] [-wait]
```

Purpose:

This Tcl command sends a message to the target, requesting a halt.

Note: This command does not guarantee that the target will halt, only that a message is sent.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted

dspload

Syntax:

```
dspload [-processor value] fileName [-symbols] [-wait]
```

Purpose:

This Tcl command loads a file to the target. Use this command when you debug a ROM target.

Returns: 1 on success, an error message otherwise.

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`fileName`

Specifies the file to be loaded to the target

`-symbols`

Indicates that symbolic debugging information only be loaded from the file. The target itself is not loaded with the binary's image. Use this option to debug a ROM target.

`-wait`

Prevents further execution of a Tcl script until the target has halted

Tcl Command Reference

dsplookupline

Syntax:

```
dsplookupline [-processor value] file line
```

Purpose:

This Tcl command looks up the start and end address corresponding to `file line`.

Returns: A list comprising two elements, representing the start and end addresses. If line number information cannot be determined, an error is returned.

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed. Omitting this argument steers the command toward the currently focused processor.

`file`

Specifies the name of the file

`line`

Specifies the line in the file

Example:

Set a breakpoint at a line.

```
> dspsetbreak [lindex [dsplookupline foo.c 100] 0]
```

Note: This example uses Tcl's `lindex` command to access the first element in the two-element list.

dsplookupsymbol

Syntax:

```
dsplookupsymbol [-processor value] [-memory value] label
```

Purpose:

This Tcl command looks up the address of a symbol.

Returns: The label's address

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-memory value`

Specifies the memory in which the look up is to be performed.

The form of this argument is one of the strings returned by the `dspgetmeminfo` command (see [“dspgetmeminfo” on page C-29](#)).

`label`

Specifies the symbol

Tcl Command Reference

dspmemorywin

Syntax:

```
dspmemorywin memory addr  
                [-rect { left top right bottom }]  
                [-title name]? [-track expr]?
```

Purpose:

This Tcl command opens a **Memory** window for the memory identified by `memory` at the address specified by `addr`.

Returns: 1

Arguments:

`memory`

One of the strings returned by `dspgetmeminfo` (see [“dspgetmeminfo” on page C-29](#)).

`addr`

Specifies the address

`-rect { left top right bottom }`

Specifies the coordinates of the rectangle enclosing the window. This list has four integer values. If the rectangle size is not specified, the MS-Windows system library sets the size.

`-title name`

Assigns the specified name to the window

`-track expr`

Focuses the window onto a specific activity. The expression `expr` is evaluated at every processor halt, and the window moves to the address based on the evaluation of the expression. Use this option to follow pointer movement.

dspplotrotate

Syntax:

```
dspplotrotate title az el
```

Purpose:

This Tcl command rotates a waterfall plot. You choose the degree of azimuth rotation and elevation rotation.

Returns: On failure, this command returns an error message in the Output window.

Arguments:

title

This title must match a title of a previously defined waterfall plot.

az

The azimuth rotation angle (from 0 to 360 degrees)

el

The elevation viewpoint (from -90 to +90 degrees)

dspplotwin

Syntax:

```
dspplotwin memtype addr count
           [-stride value]
           [-datatype value]
           [-plottype value]
           [-title value]
           [-setname value]
           [-xmemtype value]
           [-xaddr value]
           [-xstride value]
           [-xdatatype value]
           [-columncount value]
           [-add]
```

Purpose:

This Tcl command creates and displays a plot in a Plot window.

Returns: On failure, this command returns an error message in the Output window.

Arguments:

memtype

The Y-axis or Z-axis memory type, such as \$dm or \$pm. This depends of the plot type.

addr

The Y-axis or Z-axis address. This depends on the plot type.

count

The Y-axis or Z-axis row count. This depends on the plot type.

-stride value

The Y-axis or Z-axis stride. This depends on the plot type.

`-datatype value`

The Y-axis or Z-axis data type, such as float. This depends on the plot type.

`-plottype value`

Specifies the desired plot type. Choose: line, xy, constellation, eyediagram, waterfall, or image.

`-title value`

The plot's title. Surround the text with double-quote characters (").

`-setname value`

The name of this data set. Surround the text with double-quote characters (").

`-xmementype value`

The X-axis memory type, such as \$dm. This depends on the plot type.

`-xaddr value`

The X-axis address. This depends on the plot type.

`-xstride value`

The X-axis stride. This depends on the plot type.

`-xdatatype value`

The X-axis data type, such as float. This depends on the plot type.

`-columncount value`

The Z-axis column count

`-add`

Specifies that you are adding data to a previously defined plot.

Tcl Command Reference

Examples:

The following commands configure and display various plots.

Eye Diagram Plot

```
dspplotwin
    $dm eyedata 100
    -datatype float
    -plottype eyediagram
    -title "Eye Diagram"
    -setname "Input"
```

Constellation Plot

```
dspplotwin
    $dm ydata 100
    -datatype float
    -plottype constellation
    -xmentype $dm
    -xaddr xdata
    -xdatatype float
    -title "Constellation Plot"
    -setname "Input"
```

Waterfall Plot

```
dspplotwin
    $dm zdata 64
    -datatype float
    -plottype waterfall
    -title "Waterfall Plot"
    -setname "Input"
    -columncount 20
```

dspprojectaddfile

Syntax:

```
dspprojectaddfile [-folder name] filename
```

Purpose:

This Tcl command adds the file's `filename` to the currently loaded project.

Arguments:

`-folder name`

Specifies the folder in which to place the file.

If the folder does not exist, it is created.

If no folder is specified, the file is inserted at the root of the project file hierarchy.

`filename`

Specifies the name of the file to be added

Returns: An error condition occurs when:

- No project is loaded
- `filename` does not exist
- `filename` is already in the project

Tcl Command Reference

dspprojectaddfolder

Syntax:

```
dspprojectaddfolder [-ext extensions] foldername
```

Purpose:

This Tcl command adds the folder's `foldername` (for example, `folder1` or `folder1/folder2`) to the currently loaded project.

Arguments:

`-ext extensions`

Specifies extensions for the folder.

`Foldername`

Specifies the name of the folder.

A parent folders specified in the path name that does not currently exist is created.

Returns: An error condition occurs when no project is loaded.

An attempt to add a folder currently in the project results in success.

dspprojectbuild

Syntax:

```
dspprojectbuild projconfig [-all]
```

Purpose:

This Tcl command builds the configuration `projconfig` of the currently loaded project.

Arguments:

`projconfig`

Identifies the project configuration

`-all`

Performs a “rebuild all”. If not specified, an increment build is preformed

Returns: Upon success, returns the text output by the build process.

An error condition returns when:

- `projconfig` does not exist
- No project is loaded
- The build fails for any reason

Tcl Command Reference

dspprojectclose

Syntax:

```
dspprojectclose [-save] [filename]
```

Purpose:

This Tcl command closes the currently loaded project.

Arguments:

`-save`

Specifies that changes to the project be saved.

If the flag is omitted, the project is closed without saving.

`filename`

Specifies the file name to save to (that is, “Save As”).

If the filename is not specified, the project is saved to the location from which it was loaded.

Returns: An error condition occurs when:

- No project is loaded
- The project cannot be saved (for example, when the project file is read-only)
- `filename` is specified and `-save` is not

dspprojectinfo

Syntax:

```
dspprojectinfo
```

Purpose:

This Tcl command returns information about the currently loaded project.

The return value is a list comprising of two members:

- A list of the project's configurations
- A list of the project's input files. Each element of the list is a list containing the filename and project folder. Project folders use a file hierarchy-like syntax, with "/" to indicate the root of the project's file hierarchy.

Returns: An error condition returns when no project is loaded.

Example:

```
% dspprojectinfo
{debug release} {{ "C:/ProjectA/ProjectA.c" "/" } \
{ "C:/ProjectA/ProjectA.h" "/Include Files" }}
```

In this example, the returned information indicates that the project has two configurations (debug and release) and comprises two input files.

Tcl Command Reference

dspprojectload

Syntax:

```
dspprojectload projectname
```

Purpose:

This Tcl command loads the project's `projectname` into VisualDSP++.

Returns: An error condition returns when:

- `projectname` does not exist
- Another project is already loaded into VisualDSP++

dspprojectremovefile

Syntax:

```
dspprojectremovefile filename
```

Purpose:

This Tcl command removes the file `filename` from the currently loaded project.

Returns: An error condition occurs when:

- No project is loaded
- `filename` is not included in the project

dspprojectremovefolder

Syntax:

```
dspprojectremovefolder foldername
```

Purpose:

This Tcl command removes the folder's `foldername` from the currently loaded project.

Returns: An error condition occurs when:

- No project is loaded
- `foldername` is not included in the project

Tcl Command Reference

dsregisterwin

Syntax:

```
dsregisterwin { { name x y [type] } }  
              [-title name]  
              [-format value]  
              [-rect { left top right bottom }]
```

Purpose:

This Tcl command creates and displays a custom register window.

Returns: 1

Arguments:

```
{ { name x y [type] } }
```

A list of registers. Each element of this list is itself a list, containing information for a single register, including: the ASCII name of the register, the X and Y coordinates that position the register in the window (measured in characters), and (optionally), the type of the register.

Valid values for **type** are `normal` (default), `nodata`, `nolabel`, and `private`.

```
-title name
```

Assigns a title to the register window

```
-format value
```

Specifies the base format for which the X and Y position were calculated (default value is hexadecimal)

```
-rect { left top right bottom }
```

A list comprising four integers. If the rectangle size is not specified, the MS-Windows system library picks the size.

dsppreset

Syntax:

```
dsppreset [-processor value]? [-wait]
```

Purpose:

This Tcl command sends a message to the target to reset.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted

Tcl Command Reference

dsprestart

Syntax:

```
dsprestart [-processor value] [-wait]
```

Purpose:

This Tcl command sends a message to the target to restart.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed. Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted

dsprun

Syntax:

```
dsprun [-processor value] [-wait]
```

Purpose:

This Tcl command sends a message to the target to run.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted

Tcl Command Reference

dspset

Syntax:

```
dspset [-processor value] left_expr right_expr
```

Purpose:

This Tcl command evaluates `right_expr` and assigns its value to `left_expr`.

Only rudimentary checking is performed. Modifiers like `const` are ignored.

Returns: `right_expr`

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.
`value` specifies the processor to which the command is directed. Omitting this argument steers the command toward the currently focused processor.

`left_expr`

Specifies the expression to be assigned a value. This expression must be an lvalue.

`right_expr`

Specifies the expression to be evaluated

Example:

Set the value of R0 to the address of a C variable.

```
> dspset \R0 "&my_variable"
```

dspsetbreak

Syntax:

```
dspsetbreak [-processor value]
             address
             [-expression value]
             [-count value]
             [-temporary]
             [-disabled]
```

Purpose:

This Tcl command sets (inserts) a breakpoint on the target at the address specified by `address`.

This command returns a non-zero value, representing the identifier of the breakpoint. This value can be used in subsequent calls to `dspcancebreak` and `dspgetbreak`.

Note: This functionality was available by means of the `dspbreakpoint` command in earlier software releases.

Returns: On success, this command returns a non-zero value that represents the identifier of the breakpoint; otherwise, a 0 is returned. The returned value can be stored and used in subsequent calls to [“dspcancebreak” on page C-18](#) and [“dspgetbreak” on page C-25](#).

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

Tcl Command Reference

`address`

The address to which a breakpoint is specified

`-expression value`

Specifies a condition that must be evaluated to TRUE to halt execution of the debug target at the breakpoint address.

If `expression` and `count` are omitted, execution of the debug target stops at the breakpoint.

Valid expressions are anything that the Expressions window accepts.

`-count value`

Delays the setting of the breakpoint.

`value` specifies the number of halts that pass before setting the breakpoint

`-temporary`

Cancels the breakpoint at the next halt (implements features like run-to-cursor).

`-disabled`

Disables the breakpoint

Example:

The following command sets a temporary breakpoint at `main()`.

```
dspsetbreak [dsplookup symbol main] -temporary
```

dspsetmemblock

Syntax:

```
dspsetmemblock [-processor value] memory start count
               [-stride value] [-format value] { fill ... }
```

Purpose:

This Tcl command sets a block of memory.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`memory`

Specifies the memory to set the block in and is one of the strings outputted by `dspgetmeminfo` (see [“dspgetmeminfo” on page C-29](#)).

`start`

Specifies the first address to set in the block.

`count`

Specifies the total number of addresses to set.

If the length of `fill` is less than `count`, the fill values wrap to provide values for all count addresses.

Tcl Command Reference

`-stride value`

Specifies the distance between memory locations. If `value` is not specified, 1 is used.

`-format value`

Specifies the format used to format the data.

Valid values are hexadecimal (default), integer, unsigned integer, float, and octal.

Values vary from target to target.

`fill`

This list specifies the value(s) with which to fill memory.

If the length of `fill` is less than the `count`, the fill values wrap to provide values for all count addresses.

Example:

This example fills ten addresses in data memory with a dummy fill value (as Microsoft Visual C++ does in `malloc()`).

```
> set dm [lindex [lindex [dspgetmeminfo] 1] 0]
Data(DM) Memory
> dspsetmemblock $dm 0x30000 10 0xcdcdcdcd
1
```

dspsetswstack

Syntax:

```
dspsetswstack [-processor value] cookie
```

Purpose:

This Tcl command changes the debug focus to the frame identified by cookie.

Returns: 1

Arguments:

-processor value

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

value specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

cookie

Specifies the frame.

This value is determined by calling `dspgetswstack` (see [“dspreset” on page C-49](#)).

Example:

Notice the movement of focus.

```
>dspgetswstack
{"foo()" 0x2fff5 focus} {"main(int, char**)" 0x2ffff}
>dspsetswstack 0x2ffff
1
>dspgetswstack
{"foo()" 0x2fff5} {"main(int, char**)" 0x2ffff focus}
```

Tcl Command Reference

dspstepasm

Syntax:

```
dspstepasm [-processor value] [-wait]
```

Purpose:

This Tcl command steps the target a single disassembly step.

Returns: 1

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed. Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted.

dspstepin

Syntax:

```
dspstepin [-processor value] [-wait]
```

Purpose:

This Tcl command steps the target a single source language step.

Returns: 1 on success, 0 when source stepping is not enabled at PC

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted.

Tcl Command Reference

dspstepout

Syntax:

```
dspstepout [-processor value] [-wait]
```

Purpose:

This Tcl command steps the target out of the current subroutine in a source language.

Returns: 1 on success, 0 otherwise (that is, source stepping not enabled at PC)

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed. Omitting this argument steers the command toward the currently focused processor.

`-wait`

Prevents further execution of a Tcl script until the target has halted.

dspstepover

Syntax:

```
dspstepover [-processor value] [-wait]
```

Purpose:

This Tcl command steps the target a single source language step, but skips over subroutine calls.

Returns: 1 on success, 0 otherwise (that is, source stepping not enabled at PC)

Arguments:

`-processor value`

Relevant only in a multiprocessor debug session and ignored during a single-processor debug session.

`value` specifies the processor to which the command is directed.

Omitting this argument steers the command toward the currently focused processor.

`-wait`

Indicates that the script should execute and halt the target, placing the target in a known state

Additional Tcl Resources

dspwaitforhalt

Syntax:

```
dspwaitforhalt
```

Purpose:

This Tcl command delays further execution of Tcl commands until the target has halted.

Returns: 1

Additional Tcl Resources

The following resources can aid you in using Tcl to enhance your debug sessions:

- *www.scriptics.com* is a resource for Tcl programmers
- *Practical Programming in Tcl & Tk*, by Brent B. Welch (ISBN 0136168302)