Tcl Integrated Development Environment User Guide

Version T-2022.03-SP1, June 2022

Feedback

Contents

About This Guide

This guide describes the Tcl Integrated Development Environment tool and provides procedures for accessing the basic functionality through the GUI.

These topics are available:

- Related Products and Trademarks
- Conventions
- Customer Support

Related Products and Trademarks

This guide refers to the following products:

Synopsys SolvNet® support site

OASIS®

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Italic`	Indicates a user-defined value, such as *`object_name`*.
Bold	• Within syntax and examples, indicates user input—text you type verbatim. • Indicates a graphical user interface (GUI) element that has an action associated with it.
[ ]	Denotes optional parameters, such as: `write_file [-f` *`filename`*`]`
`...`	Indicates that parameters can be repeated as many times as necessary: *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low	medium	high`

Convention	Description
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > **Copy**	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing the C key.

Customer Support

Customer support is available through the Synopsys SolvNet customer support website and by contacting the Synopsys support center.

Accessing SolvNet

The SolvNet support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNet site:

1. Go to the web page at https://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

If you need help using the site, click **Help** on the menu bar.

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys Global Support site on synopsys.com. There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support site and open a case online (Synopsys user name and password required).

Feedback

1

Using The Tcl Debugger

This chapter describes using the Tcl Debugger Integrated Development Environment tool Graphical User Interface (GUI).

These topics are included in this chapter:

About the Tcl Debugger

The Tcl Debugger is a dedicated tool for detecting and fixing source code errors (or bugs) and is encapsulated in the fwIDE module. It uses an MP (multi-process) approach, which ensures a robust debugging process while at the same time meeting the basic principles of debugging:

- The program that is executing and the debugger cannot be executed in the same thread.
- There must be minimal intrusion by the debugger on the application that is being debugged.

Interaction between the Editor and the main application is done via a socket- based pipeline.

No matter how thoroughly you inspect your source code, you can overlook some command which, although syntactically correct, leads to incorrect results. For example, you could unintentionally use the same variable for different purposes and not notice this inconsistency. It is difficult to locate such errors, unless you examine the run-time behavior of the macro. The Tcl Debugger provides that functionality.

The Tcl Editor has a Multiple Document Interface (MDI). It supports tabbed editing, which allows you to work with multiple open files in a single window. Additionally, the Tcl Debugger uses the Scintilla editor component. Scintilla is a free open source library that provides a text editing component function that supports syntax highlighting as well as many features to make code editing easier. The control supports line numbering in the margin and line markers such as code breakpoints.

If you are working in **Ribbon** GUI mode, you will also see a ribbon at the top of the Tcl Debugger window. For additional information about **Ribbon** GUI mode, see the section *The Ribbon* in the *IC Validator WorkBench User Guide*. The ribbon in the Tcl Debugger contains groups of commands, which are organized by function, into three tabs. Each tab in the ribbon—**File**, **Home**, and **View**—contains menu items, buttons, drop-down lists, and fields that you can use to perform standard file open and save operations, text editing operations, and a set of specific operations described in this chapter.

The main title bar of the Tcl Debugger shows the full path of the script that is currently open. Whenever you edit the text, the tabbed document icon changes to red to indicate that the source code has been modified. You can also change the order of your tabs by dragging a tab along the tab bar. Line numbers are shown on the left to help you easily identify lines of the source code.

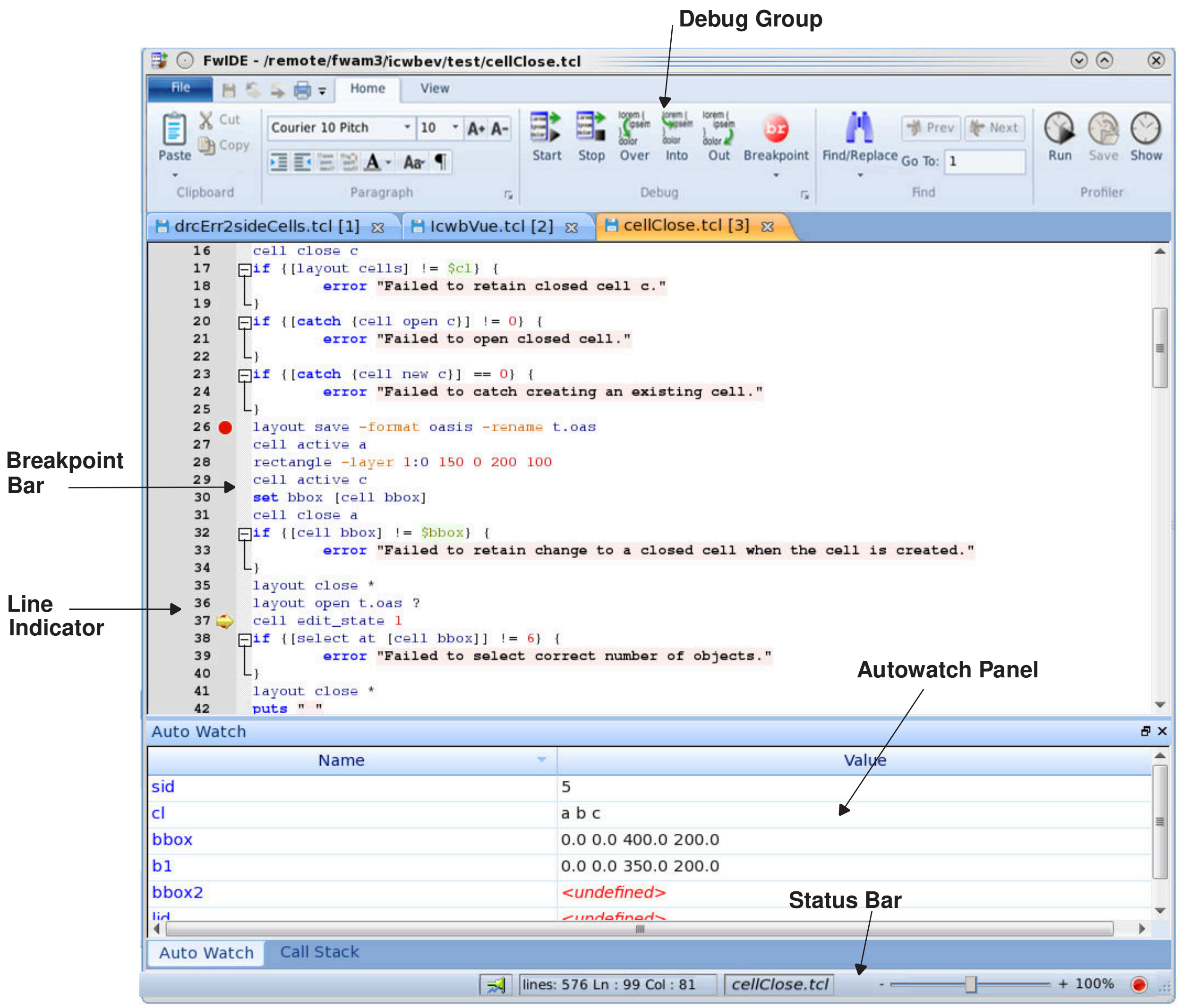
The status bar at the bottom of the Tcl Debugger provides socket connection status, line and column number information, the currently active file name, a zoom slider that you can use to enlarge or reduce the size of the text and line numbers in the window, and a status indicator to indicate whether a debug session is in progress.

Running the Tcl Debugger

To start the Tcl Debugger, select **Tcl Debugger** from the **Macros** group on the **View** tab.

The Tcl Editor window opens, which is shown in Figure 1. Choose **File > Open** and you can select a file to open from a list of recent files, or you can browse to the location of a file you want to open.

Figure 1 Tcl Editor Main Window



```
cell close c
if {[layout cells] != $cl} {
        error "Failed to retain closed cell c."
}
if {[catch {cell open c}] != 0} {
        error "Failed to open closed cell."
}
if {[catch {cell new c}] == 0} {
        error "Failed to catch creating an existing cell."
}
layout save -format oasis -rename t.oas
cell active a
rectangle -layer 1:0 150 0 200 100
cell active c
set bbox [cell bbox]
cell close a
if {[cell bbox] != $bbox} {
        error "Failed to retain change to a closed cell when the cell is created."
}
layout close *
layout open t.oas ?
cell edit_state 1
if {[select at [cell bbox]] != 6} {
        error "Failed to select correct number of objects."
}
layout close *
puts " "
```

Name	Value
sid	5
cl	a b c
bbox	0.0 0.0 400.0 200.0
b1	0.0 0.0 350.0 200.0
bbox2	<undefined>
lid	<undefined>

Creating a New Document

To create a new document, choose **File > New** or double-click the tab bar to create a new blank document. You can use your own template or you can create a new document with one of the default templates.

To search for existing templates to use for your new document, select your template from the **Search for templates** drop-down list or click the browser button to specify the directory path where your template is located.

Note:

If you want to add your own template file, you can place it in install_home/etc/tcl/templates.

The Tcl Debugger provides two default templates for creating a new document:

- **Blank Document**—Creates a blank document with no formatting.
- **Find Command Data Callback**—Creates a new document that will contain an example for `find init -data`.

The template file will contain two additional definitions, which will not be applied to new document:

- # TEMPLATE_NAME
- # TEMPLATE_ICON

The Tcl Debugger will use these definitions to show the template name and icon in the GUI. They are not mandatory for the template file, and if one of them is not specified, their default values are used.

Debugging Operations

The following operations are located in the **Debug** group on the **Home** tab:

- **Start**—initiates the debugging session for the script currently in view. If no breakpoint is set, the entire script is executed.
- **Stop**—stops the current debugging session.
- **Pause**—Pauses the running debug session. You can continue the execution with the **Over**, **Into**, **Out**, or **Continue** operations, or stop the execution by clicking **Stop**.
- **Over**—executes the currently highlighted command line and pauses again. If the debugging session is initiated by **Over**, the debugger highlights the first executable line in the source code, pauses, and waits for your instructions. These time-outs give you an opportunity to observe variables or to modify the configuration of breakpoints.
- **Into**—penetrates the body of the called procedure or script to facilitate stepwise execution of its internal commands. If a script is sourced, then it is opened and executed step by step. If there are several statements on the same line, then each part is executed step by step and the part to be executed is highlighted. This operation works as specified only if the current command calls a procedure or a script. In all other cases, it works like **Over**.
- **Out**—Executes all remaining instructions in the current scope and then jumps up one level.

- **Breakpoint**—Sets or deletes a breakpoint in the line of source code at the current cursor position. To delete all breakpoints, click the down arrow under **Breakpoint** and select **Delete All Breakpoints**. It is usually more convenient to click the breakpoint bar at the position next to the line of code where you want to set or remove a breakpoint.

  Note:

  The Tcl Debugger cannot set a breakpoint if you encounter an empty line, a comment, or an intermediate part of a split command. In these cases, the Tcl Debugger sets the breakpoint at a reasonable position. In multi-line commands, the breakpoint is always set at the first line.

- **Continue**—continues execution of the macro from the interrupted position. In this way, you concede control to the debugger. The interpretation of the macro continues until a breakpoint is encountered or the macro terminates, whichever occurs first.

To start a debugging session:

1. Load the macro file that you want to debug.
2. Set breakpoints at the lines of interest.

   This will interrupt the execution of the macro at these lines so that you can query the state of variables and determine the reasons for anomalies if any are observed.
3. If a bug is found, you can fix it, and without saving the corrected macro file, you can start another debugging session to ensure that the issue has been resolved.

Note:

- You cannot edit a script while in a debug session.
- It is strongly recommended to not use `user_socket` commands while in a debugging session as it can lead to unpredictable results.

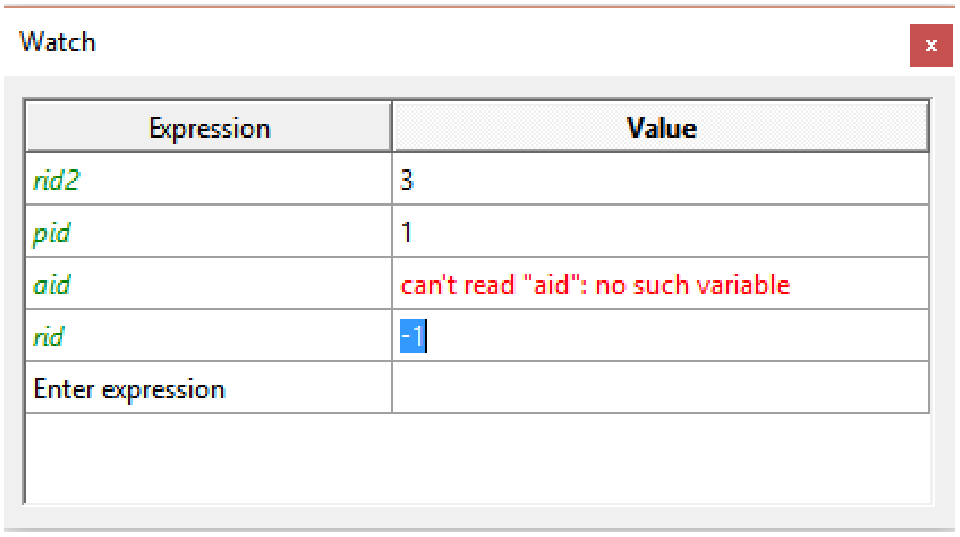
The Auto Watch Panel

The **Auto Watch** panel displays the current values of the script variables. By default, this list includes variables defined in the scope of the script that is being debugged. The Tcl Debugger compiles the list dynamically. It looks through the source code of the script and collects all distinct arguments of Tcl set commands it encounters. This procedure ensures that the final list contains no irrelevant variables, but it also excludes from consideration variables defined elsewhere, yet in the context of the current execution.

The Watch Panel

The **Watch** panel displays the results of a Tcl expression that you specify and want to evaluate. Suppose you reach a point in the source code where the **Watch** panel displays the state shown in Figure 2. You suspect that the value of variable rid is wrong; it must be 0, not -1. Before determining where and how this incorrect assignment was made, you decide to check whether or not your assumption is true. In the **Value** column, you double-click the value of rid and type 0. After debugging some commands that follow, you most probably can conclude whether the negative value of *rid* was a genuine bug.

Figure 2 The Tcl Debugger Watch Panel



Expression	Value
rid2	3
pid	1
aid	can't read "aid": no such variable
rid	-1
Enter expression	

The Script Profiler

The Script Profiler is used to collect time information about each executed command in a script. This helps you to analyze script performance issues, to detect obscure bottlenecks, and to look for faster solutions wherever possible.

The Script Profiler includes two types of counters:

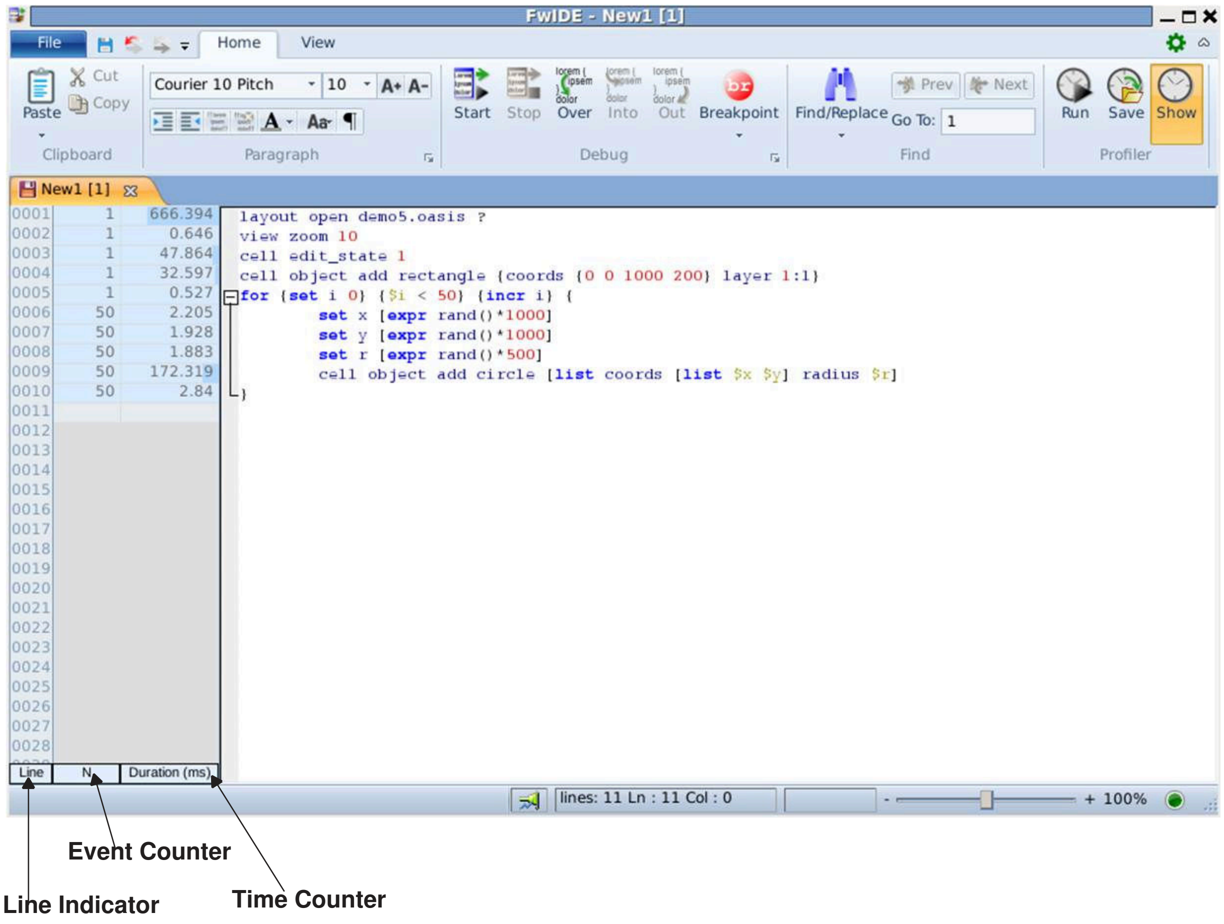
- An event counter, which counts how many times a line of source code has been visited during the execution of the script.
- A time counter, which accumulates the net time (in milliseconds) spent executing a line of source code.

Note:

The script is profiled line-by-line and reports the current line execution time. It does not include background processes spawned by the script line or GUI updates performed after the script line has been executed. Also, when calling a TCL procedure, the procedure calling line will show only the call time and not the procedure execution time. To see the procedure execution time, view the profiling information at the procedure body lines.

The Script Profiler is always active and the **Run**, **Save**, and **Show** buttons are located in the **Profiler** group on the **Home** tab. Click **Run** to initiate the Script Profiler. In response, the Tcl Debugger replaces the Editor's left-side panel with the Profiler histogram, which displays the same line numbers of the active source code from the Tcl Debugger along with event and time counters (see Figure 3). Click **Show** to toggle between showing or hiding the Profiler histogram. Click **Save** to save the profile as a text file.

Figure 3 The Tcl Debugger Script Profiler



Line	N	Duration (ms)
0001	1	666.394
0002	1	0.646
0003	1	47.864
0004	1	32.597
0005	1	0.527
0006	50	2.205
0007	50	1.928
0008	50	1.883
0009	50	172.319
0010	50	2.84
0011		
0012		
0013		
0014		
0015		
0016		
0017		
0018		
0019		
0020		
0021		
0022		
0023		
0024		
0025		
0026		
0027		
0028		

```
layout open demo5.oasis ?
view zoom 10
cell edit_state 1
cell object add rectangle {coords {0 0 1000 200} layer 1:1}
for {set i 0} {$i < 50} {incr i} {
        set x [expr rand()*1000]
        set y [expr rand()*1000]
        set r [expr rand()*500]
        cell object add circle [list coords [list $x $y] radius $r]
}
```

Each time-counter cell is also an inverted histogram bar. If it displays value t, the highlighted part of the cell reflects the ratio t/tmax, where tmax is the longest duration observed during the execution of this source. The resultant profile can be saved in a text file. The Script Profiler uses a special print format to illustrate the profiling results that takes in both line numbers and counter values.

Edit and Find Operations

Tcl Debugger operations are available in the **File**, **Home**, and **View** tabs. Many of these operations are standard—opening a file (**File** tab) and cut, copy, and paste edit operations (located in the **Clipboard** group on the **Home** tab). However, some operations are specific to the Tcl Debugger.

The following Tcl Debugger-specific operations are available in the **Paragraph** group on the **Home** tab:

- **Indent+** (or **Ctrl**+**]**)—Shifts the selected text to the right by one tab space. The tab spacing in the Script Editor equals four consecutive spaces.
- **Indent-** (or **Ctrl**+**[**)—Shifts the selected text to the left (if possible) by one tab space.
- **Comment** (or **Ctrl**+**K**)—Converts the selected text to Tcl-specific comments. For this, the command inserts a hash (#) character at the beginning of each selected line.
- **Uncomment** (or **Ctrl**+**Shift**+**K**)—Removes the leading # characters, if any, from the selected lines.
- **Aa** button—Converts character case. Highlight a word or string and click the down arrow on the right side of this button to change to uppercase or lowercase or to convert the string you select to a Tcl string or convert it from a Tcl string.
- **Highlight Color (A)** button—Click this button to change the background color of selected text. The default value is light grey.

The **Paragraph** group also contains drop-down lists and buttons that you can use to select the font and font size, increase or decrease the font size, choose a highlight color, and show the ends of lines.

The following operations are available in the **Find** group on the **Home** tab:

- **Find/Replace** (or **Ctrl**+**F**)—Opens the **Find and Replace** dialog box, which you can use to initiate a search to find (or find and replace) text in the current file or in all opened files with various search criteria that you specify. Click the down arrow under Find and select **Show results...**to view your search results in the **Find Results** dockable pane. You can select any search result that is displayed and click the icons in the top of the pane to navigate through the files to find your search results or to clear all search results.

- **Prev**—Click this button to find the previous occurrence of the search term.
- **Next**—Click this button to find the next occurrence of the search term.
- **Go To:**—A text entry field where you can specify a particular line number. Enter a line number and press **Enter**. The Tcl Debugger moves the cursor to the beginning of that line.

Modifying Default Settings

You can modify Tcl Editor options such as fonts, selection color, showing/hiding line numbers, highlighting text in the Editor, and hotkey assignments in the **Options** dialog box.

To change the default Tcl Debugger settings:

1. Click the **Settings** button (⚙) on the far-right side of the ribbon in the Tcl Debugger window or click the lower-right launch button in either the **Paragraph** or **Debug** groups on the **Home** tab.

   The **Options** dialog box opens.
2. Navigate the tree on the left to select a page.
3. Set the options.
4. Click **OK**.

The modified settings are applied to the current session.

To save the current configuration so it applies to future sessions, you can click **Save** in the **Options** dialog box. Your changes to the settings are saved in the `HOME/.synopsys/icvwb` (see the section *Startup Files* in the *IC Validator WorkBench User Guide*). `HOME` on Windows is `%UserProfile%`.

You can reset the modified settings to the defaults at any time by clicking **Restore Defaults** in the **Options** dialog box.

These options are available for you to modify:

- General Default Settings
- Hotkeys Settings
- Text Editor Settings

General Default Settings

The General options control the text fonts for the Tcl scripts (fixed-width font) and the other GUI features (default font). Click the button beside either **Default:** or **Fixed-width:** to open the **Select Tcl Debugger Font** dialog box where you can change the font, font style, size, and font effects. Click **OK** in this dialog box to confirm your changes, and then in the **Options** dialog box, click either **OK** to apply the font changes for the current session or **Save** to apply the changes and save the font settings for future sessions.

Hotkeys Settings

To view and change hotkey assignments, select **Hotkeys**. To change or assign the hotkey for an action, click the corresponding button, and then press the key or key combination you want to assign to the hotkey. The button updates with the key or key combination you entered. To unassign a hotkey, click the desired button to select it and then right-click and select **Delete shortcuts** from the context menu.

Text Editor Settings

To view and change text editing options, select **Text Editor**.

- **Turn on auto completion**—When auto-completion is enabled, the Tcl Editor will display a drop-down list of word choices as you are typing. If the word you want to type is in the list, select it and double-click to add it to your script. Auto-completion is enabled by default. The list of auto-completion choices is based on Tcl keywords, all registered commands in the application, the current contents of the document, and the characters immediately to the left of the cursor.
- **Threshold**—This is the number of characters that you have to type before the auto-completion word list is displayed. The default value is 1. If you enter a value of 0, then the word list is disabled.
- **Case sensitive**—If you select this check box, the auto-completion lists are case-sensitive. This option is enabled by default.
- **Replace word**—If you select this check box, the word you are typing will be replaced with the word you select from the auto-completion list. This option is disabled by default.
- **Show caret line**—Select this check box to enable the background color of the line where the cursor is located. To change this color, click the **Background color:** button, and then select a color from the color palette; click **More...** to view additional color choices, and then double-click to apply the color change to the button. This option is disabled by default.

- **Brace options**—This option enables you to configure the brace-matching mode. The character pairs {}, [] and () are treated as braces. If you select **No**, brace-matching is disabled. **Strict** enables brace-matching for a brace immediately before the current position; **Sloppy** enables brace-matching for a brace immediately before or after the current position. **Sloppy** is the default option.
- **Colors**—You can specify the background color used to display matched and unmatched braces. (The character pairs {}, [], and () are treated as braces.) The default color for **Matched:** is yellow. If one of the characters is missing or different in the pair, it will be highlighted with blue (the default color for **Unmatched:**) when you select the other character. Click the boxes beside **Matched:** or **Unmatched:** to change the background color used to display matched and unmatched braces. Select a color from the color palette; click **More...** to view additional color choices, and then double-click to apply the color change to the button.
- **Show line numbers**—Select this check box to show line numbers in the Tcl Editor. This option is enabled by default.

Coloring Options

To view and change colors for Tcl keywords and framework commands, select **Coloring**.

- **Highlighting styles**—Click the color button to change the color of an attribute. Select a color from the color palette; click **More...** to view additional color choices, and then double-click to apply the color change to the button. Change the font style of an attribute by selecting the check boxes (**Bold**, **Italic**, **Underline**) on the corresponding rows.
- **Background color:**—Click this button to change the background color of the Tcl editor. The default background color of the editor is white. Select a color from the color palette; click **More...** to view additional color choices, and then double-click to apply the color change to the button.
- **Selection color:**—Click this button to change the background color of selected text. Select a color from the color palette; click **More...** to view additional color choices, and then double-click to apply the color change to the button.

Feedback

2

Tcl Debugger Commands

This chapter describes the commands that can be executed from the Tcl Debugger command pane.

These sections list and describe the viewing and editing commands:

- fwIDE goto line
- fwIDE profiler run
- fwIDE profiler save

default fwIDE_font_default

Sets the default GUI font in the Tcl Debugger.

Syntax

```
default fwIDE_font_default [font]
```

Returns

If you do not specify an argument, the command returns the current setting.

If you specify an argument, the command does not return anything.

Arguments

font

Specifies the name for the font (or any other string accepted by the Qt `font_create` command). It is a comma-separated list of attributes.

Description

Specifies the font of the text in menus, dialogs, view windows, and so on.

Default
" "

Examples

```
default fwIDE_font_default
default fwIDE_font_default {Sans Serif,12,-1,5,50,0,0,0,0,0}
```

See Also

- default fwIDE_options
- default hotkeys

default fwIDE_options

Sets Tcl Debugger options.

Syntax

```
default fwIDE_options [option_values]
```

Returns

If you do not specify an argument, the command returns the current setting.

If you specify an argument, the command does not return anything.

Arguments

`[option_values]`

Specifies the options available for customizing Debugger attributes.

`option_values` is a key-value pair. Allowed key-values include the following:

- `enableAutocomplete` *`boolean`*—Specifies the Boolean value to turn the auto complete capability on (1) or off (0).
- `autocompleteThreshold` *`integer`*—Specifies the threshold for the number of letters from where the automatic displaying of the autocomplete options appears as you type. The value should be a positive integer number.
- `autocompleteCaseSensitive` *`boolean`*—Specifies whether or not the auto-completion lists are case-sensitive.
- `autoCompleteReplaceWord` *`boolean`*—Specifies whether to remove the rest of the word to the right of the cursor on autocomplete.
- `braceMatchMode` *`no|strict|sloppy`*—Specifies the brace matching mode. The mode can be one of the following: `no|strict|sloppy`.

  The meaning for each of these values is as follows:

  - `no`—Brace matching is disabled.
  - `strict`—Brace matching is enabled for a brace standing immediately before cursor position.
  - `sloppy`—Brace matching is enabled for a brace standing immediately before or after cursor position.
- `matchedBraceBgColor` *`color`*—Specifies the background color used to display matched braces. The value should be in #RRGGBB format or an X-windows defined color (such as red, green, and so on).
- `unmatchedBraceBgColor` *`color`*—Specifies the foreground color used to display unmatched braces.
- `caretLineVisible` *`boolean`*—Enables(1) or disables(0) caret line.

- `caretLineWidth` *`width`*—Specifies the width of the caret in pixels.
- `caretLineBgColor` *`color`*—Specifies the background color of the line containing the caret. In addition to accepting #RRGGBB values, this option will also accept an X-windows defined color (red, green, and so on).
- `selectionColor` *`color`*—Specifies the color for the selection. In addition to accepting #RRGGBB values, this option will also accept an X-windows defined color (red, green, and so on).
- `fixedFont` *`font`*—Specifies the name for the font (or any other string accepted by the Qt `font_create` command). It is a comma-separated list of attributes.
- `showLineNumbers` *`boolean`*—Specifies whether to show (1) or not show (0) line numbers in the Tcl Debugger.
- `text_highlight` *`format`*—Specifies the coloring and formatting for text.
- `modifier_highlight` *`format`*—Specifies the coloring and formatting for modifiers.
- `instructionWord_highlight` *`format`*—Specifies coloring and formatting for instruction words.
- `number_highlight` *`format`*—Specifies the coloring and formatting for numbers.
- `string_highlight` *`format`*—Specifies coloring and formatting for string constants.

Description

Specifies the options available for customizing Tcl Debugger attributes.

Default
`{autoCompleteReplaceWord 0 autocompleteCaseSensitive 1 autocompleteThreshold 1 braceMatchMode sloppy caretLineBgColor #ccccff caretLineVisible 0 caretLineWidth 1 comment_highlight {{#009900} 0 0 0} enableAutocomplete 1 fixedFont {} instructionWord_highlight {{#0000ff} 1 0 0} matchedBraceBgColor #ffff00 modifier_highlight {{#cc6600} 0 0 0} number_highlight {{#ff9900} 0 0 0} selectionColor #cccccc showLineNumbers 1 string_highlight {{#000000} 1 0 0} text_highlight {{#000000} 0 0 0} unmatchedBraceBgColor #0000ff}`

Examples

```
default fwIDE_options {autocompleteThreshold 4 autoCompleteReplaceWord 1
 matchedBraceBgColor #00FF00}
default fwIDE_options {showLineNumbers 0}
```

default hotkeys

Sets the hotkeys for the Tcl Debugger.

Syntax

```
default hotkeys [action_hotkeys]
```

Returns

If you do not specify an argument, the command returns the current setting.

If you specify an argument, the command does not return anything.

Arguments

action_hotkeys

Specifies the actions and their associated keystrokes for hotkeys, such as "File->Save". This option requires you provide a list for *action_hotkeys*.

Description

Sets the hotkeys for the Tcl Debugger. After you specify the action and associated hotkey, the command sets the hotkey for the action subsequent use.

Examples

```
array set a[default hotkeys]
echo $b(File->Exit)
# Ctrl+Q
default hotkeys "File->Exit E"
array set b[default hotkeys]
echo $c(File->Exit)
# E
```

See Also

- default fwIDE_font_default
- default fwIDE_options

fwIDE breakpoint add

Adds a breakpoint at the specified line.

Syntax

```
fwIDE breakpoint add line_number [-file file]
```

Returns

Returns nothing.

Arguments

`line_number`

Specifies the line number where the breakpoint will be set.

`-file` *`file`*

Indicates the specific script file that is opened in the Debugger. If you do not specify `-file`, the Debugger uses the active script file. Use the file ID or the full file path.

Description

Sets a breakpoint at the specified line number in the active file if the `-file` option is not specified. Otherwise, a breakpoint in the specified file is set.

Examples

```
fwIDE file open foo.tcl

#-> 1

fwIDE breakpoint add 122

fwIDE breakpoint add 11
```

See Also

- fwIDE breakpoint delete
- default fwIDE_options
- fwIDE goto line

fwIDE breakpoint delete

Deletes breakpoints from opened script files.

Syntax

```
fwIDE breakpoint delete line_number [-file file]
```

Returns

Returns nothing.

Arguments

line_number

Specifies the line number where the breakpoint will be removed.

-file *file*

Indicates the specific script file that is opened in the Debugger. If you do not specify `-file`, the Debugger uses the active script file. Use the file ID or the full file path.

Description

Deletes a breakpoint at the specified line number in the active file if the `-file` option is not specified. Otherwise, a breakpoint in the specified file is deleted.

Examples

```
fwIDE file open foo.tcl

#-> 1

fwIDE breakpoint toggle 122 -file 1

fwIDE breakpoint delete 122
```

See Also

- fwIDE breakpoint add
- fwIDE breakpoint toggle
- fwIDE goto line

fwIDE breakpoint disable

Disables a breakpoint at the specified line.

Syntax

fwIDE breakpoint disable *line_number*|* [-file *file*]

Returns

Returns nothing.

Arguments

line_number

Specifies the line number where the breakpoint should be disabled. Use * to disable all breakpoints.

`-file` *`file`*

Indicates the source file in which the breakpoints should be disabled. If you do not specify `-file`, the active script file is used. Use the file ID or the full file path.

Description

Disables a breakpoint at the specified line.

Examples

```
fwIDE breakpoint disable 4
```

```
fwIDE breakpoint disable *
```

See Also

- fwIDE breakpoint delete
- fwIDE breakpoint enable
- fwIDE breakpoint toggle

---

fwIDE breakpoint enable

Enables a breakpoint at the specified line.

Syntax

```
fwIDE breakpoint enable line_number|* [-file file]
```

Returns

Returns nothing.

Arguments

`line_number`

Specifies the line number where the breakpoint should be enabled. Use * to enable all breakpoints.

`-file` *`file`*

Indicates the source file in which the breakpoints should be enabled. If you do not specify `-file`, the active script file is used. Use the file ID or the full file path.

Description

Enables breakpoint at the specified line. If no breakpoint is found at the specified line, a new breakpoint is added at that line.

Examples

```
fwIDE breakpoint enable 4
```

```
fwIDE breakpoint enable *
```

See Also

- fwIDE breakpoint add
- fwIDE breakpoint disable
- fwIDE breakpoint toggle

fwIDE breakpoint toggle

Sets or resets a breakpoint at the specified line.

Syntax

```
fwIDE breakpoint toggle line_number [-file file]
```

Returns

Returns nothing.

Arguments

`line_number`

Specifies the line number where the breakpoint will be set or removed.

`-file` *`file`*

Indicates the specific script file opened in the TCL Debugger. If you do not specify `-file`, the Debugger uses the active script file. Use the file ID or the full file path.

Description

Sets or resets a breakpoint at the specified line in the specified file.

Examples

```
fwIDE file open foo.tcl
```

```
#-> 1
```

```
fwIDE breakpoint toggle 122 -file 1
```

See Also

- fwIDE breakpoint add
- fwIDE breakpoint delete
- fwIDE goto line

fwIDE dbg in

Executes a single instruction on the target in the TCL Debugger.

Syntax

```
fwIDE dbg in
```

Returns

Returns nothing.

Description

Executes a single instruction on the target in the TCL Debugger. If the instruction is a procedure call, the Debugger steps into the procedure.

Examples

```
fwIDE dbg in
```

See Also

- fwIDE dbg next
- fwIDE dbg out
- fwIDE goto line
- fwIDE dbg stop

fwIDE dbg next

Sets the next statement to execute in the TCL Debugger.

Syntax

```
fwIDE dbg next
```

Returns

Returns nothing.

Description

Sets the next statement to execute in the TCL Debugger.

Examples

```
fwIDE dbg next
```

See Also

- fwIDE dbg in
- fwIDE dbg out
- fwIDE goto line
- fwIDE dbg stop

fwIDE dbg out

Executes all remaining instructions in the current scope and then moves up one level.

Syntax

```
fwIDE dbg out
```

Returns

Returns nothing.

Description

Executes all remaining instructions in the current scope in the TCL Debugger and then moves up one level.

Examples

```
fwIDE dbg out
```

See Also

- fwIDE dbg in
- fwIDE dbg next
- fwIDE goto line
- fwIDE dbg stop

fwIDE dbg pause

Pauses a debugging session in the Tcl Debugger.

Syntax

```
fwIDE dbg pause
```

Returns

Returns nothing.

Description

Pauses a debugging session in the Tcl Debugger.

Examples

```
fwIDE dbg run 1
fwIDE dbg pause
```

See Also

- fwIDE dbg stop

fwIDE dbg run

Starts a debugging session in the TCL Debugger.

Syntax

```
fwIDE dbg run [file]
```

Returns

Returns nothing.

Arguments

file

Specifies the opened file index or the full file path to start debugging. If you do not specify a file, the Debugger debugs the active file.

Description

Starts a debugging session in the TCL Debugger.

Examples

```
fwIDE dbg run 1
```

See Also

- fwIDE dbg in
- fwIDE dbg next
- fwIDE dbg out
- fwIDE dbg stop

fwIDE dbg stop

Stops a debugging session in the TCL Debugger.

Syntax

```
fwIDE dbg stop
```

Returns

Returns nothing.

Description

Stops a debugging session in the TCL Debugger.

Examples

```
fwIDE dbg stop
```

See Also

- fwIDE dbg in
- fwIDE dbg next
- fwIDE dbg out
- fwIDE dbg run

fwIDE file active

Activates the specified file in the Tcl Debugger.

Syntax

```
fwIDE file active [file]
```

Returns

If you do not specify an argument, returns the ID of the active file. Otherwise, returns nothing.

Arguments

file

Specifies the file to be activated in the Tcl Debugger. *file* can be the file name, the file's ID, or empty. The file name and ID are displayed on the file's tab in the editor.

Description

Makes the specified file active and displays its tab in the Tcl Debugger editor.

Examples

```
fwIDE file active New3
fwIDE file active 5
fwIDE file active
#-> 5
```

See Also

- fwIDE file new
- fwIDE file close
- fwIDE file save

fwIDE file close

Closes the file(s) opened in the TCL Debugger.

Syntax

```
fwIDE file close [filenames]
```

Returns

Returns nothing.

Arguments

filenames

Specifies the files to close. You can use "*" to indicate all opened files. If you do not specify the filename, the application closes the active file. Use the file ID or the full file path.

Description

Closes the specified file in the TCL Debugger.

Examples

```
fwIDE file new foo.tcl
#->12
fwIDE file new foo.tcl
#->13
fwIDE file close {12 13}
```

See Also

- fwIDE file new
- fwIDE file open
- fwIDE file save

fwIDE file new

Creates a new file in the TCL Debugger.

Syntax

```
fwIDE file new filename
```

Returns

Returns the ID of the newly created file.

Arguments

`filename`

Specifies the name of the new file to be created in the TCL Debugger. *`filename`* must be the name of the tab created in the editor. *tabname* must be alphanumeric and cannot exceed 127 characters.

Description

Creates a new empty file with the specified name. The file then becomes the active one. There is no specific file type associated with the file until it is saved

Examples

```
fwIDE file new New5

#-> 1
```

Feedback

See Also

- fwIDE file close
- fwIDE file open
- fwIDE file save

fwIDE file open

Opens the specified file in the TCL Debugger.

Syntax

```
fwIDE file open filename [-format format] [-reload]
```

Returns

Returns the ID of the opened file.

Arguments

filename

Specifies the file name to open in the TCL Debugger. If it is already open, the application will make the Debugger visible and show the specified file as the active file in the Debugger.

[-*format format*]

Controls whether to load the data as tcl or mac or another format supported by the Debugger (python, perl, etc). Text highlighting is set according to this argument value.

-*reload*

Reloads the file if the latter was changed/modified outside the TCL Debugger.

Description

Opens script file in the TCL Debugger.

Examples

```
fwIDE file open foo.tcl
```

```
#-> 1
```

See Also

- fwIDE file close
- fwIDE file new
- fwIDE file save

fwIDE file save

Saves the specified file in the TCL Debugger.

Syntax

```
 fwIDE file save [file] -rename filename [-format format]
```

Returns

Returns nothing.

Arguments

file

Specifies the file to save. If you do not specify a file, the application saves the active file.

-rename filename

Renames the file you are saving to a new file name.

-format format

Specifies the format for saving the file.

Description

Saves the specified file in the TCL Debugger.

Examples

```
fwIDE file save foo.tcl
```

```
fwIDE file save foo.tcl -rename tmp.tcl
```

See Also

- fwIDE file new
- fwIDE file close
- fwIDE file open

fwIDE goto line

Moves the cursor position to the specified line.

Syntax

`fwIDE goto line` *`line_number`* `[-file` *`file`*`]`

Returns

Returns nothing.

Arguments

`line_number`

Specifies the line number where you want to move the cursor.

`-file` *`file`*

Indicates the specific script file opened in the TCL Debugger. If you do not specify `-file`, the Debugger uses the active script file. Use the file ID or the full file path.

Description

Moves the cursor position to the specified line number in the specified script file.

Examples

```
fwIDE file open foo.tcl

#-> 1

fwIDE goto line 5
```

See Also

- fwIDE breakpoint add
- fwIDE breakpoint delete
- fwIDE breakpoint toggle

fwIDE profiler run

Measures the execution time of each line of code in the script.

Syntax

`fwIDE profiler run` *`scriptPath resultsFilename`*

Returns

Returns nothing.

Arguments

scriptPath

Specifies the script file path to be profiled.

resultsFilename

Specifies the output file where the profiling results should be stored.

Description

Measures the execution time of each line of code in the provided script and provides information about how many times the line was called. The results are stored in a text file that you provide.

Examples

```
fwIDE profiler run my_script.tcl result.txt
```

fwIDE profiler save

Saves the result of the profiling script's timing in a file.

Syntax

```
fwIDE profiler save filename
```

Returns

Returns nothing.

Arguments

filename

Specifies the file path where the profiling results should be stored.

Description

Saves the result of the profiling script's timing in a file.

Examples

```
fwIDE profiler save result.txt
```

See Also

- fwIDE profiler run