SLED and Lua User’s Guide

Please note that the specifications contained in this document are preliminary and subject to change without prior notice.

This document explains how to use the SLED and Lua package developed by the WWS Tools and Technology Group. For information about installing and configuring SLED, see *Getting Started with SLED and Lua.*

Your feedback is important to us. If you have any comments or questions about this document, please post them on SLED Issues on GitHub or send email to sledpatsony@gmail.com.

Version	Revision Date	Author(s)	Comments
2.0.0	17-Oct-08	DT Friedman	Initial version
3.1.0	4-Sept-09	Risa Galant	Updates for 3.1.0
3.3.0	Feb-10	Risa Galant	Updates for 3.3.0
5.0.0	Apr-14	Gary Staas	Updates for 5.0.0. Update format. Update file paths. Update UI figures. Fix typos. Add info about new features, such as Lua Intelllisense, TTY filtering, source control, and changing skin. Add info about setting preferences.
5.1.0	Oct-14	Gary Staas	Update for 5.1.0. Update toolbar. Various clarifications.
5.1.2	Feb-15	Gary Staas	Open source version.

Table of Contents

1 **Overview** **4**

Prerequisites 4

2 **The SLED Interface** **5**

The SLED Toolbar 5

Customizing the Interface 6

Sorting Data 9

The TTY Window 10

3 **Creating, Editing, Compiling, and Managing Lua Scripts** **14**

Creating Script Files 14

Opening Script Files 15

Editing Script Files 15

Compiling Script Files 16

File Management Functions 20

Creating Projects 20

Managing Projects 23

Source Control 25

4 **Debugging** **27**

Controlling Script Execution 27

Breakpoints 29

Following Execution Flow 34

Monitoring Variables 35

Modifying Variables and Scripts 41

5 **Gauging Performance** **44**

Lua Profiler 44

Tracing Memory 47

6 **Find and Replace** **50**

Quick Find 51

Quick Replace 52

Find in Files 54

Replace in Files 55

1 Overview

SLED is a powerful, yet intuitive, IDE for editing, compiling, and debugging Lua and other scripts. Based on the TNT Authoring Tools Framework, it delivers an array of features normally associated with a full-fledged development environment, such as Microsoft Visual Studio.

In addition to this Overview, this guide comprises the following chapters:

- The SLED Interface: introduces key aspects of the SLED interface.
- Creating, Editing, Compiling, and Managing Lua Scripts: explains how to create, edit, compile, and manage script files and projects.
- Debugging: discusses script debugging.
- Gauging Performance: explains how to use SLED to gauge the performance of scripts.
- Find and Replace: explains how to use the SLED search and replace functionality.

Prerequisites

In order to debug or profile scripts, you must know how to:

- Configure SLED to connect to the target machine.
- Connect SLED to the target machine.

If you do not know how to perform these tasks, see *Getting Started with SLED and Lua* before attempting debugging or profiling.

2 The SLED Interface

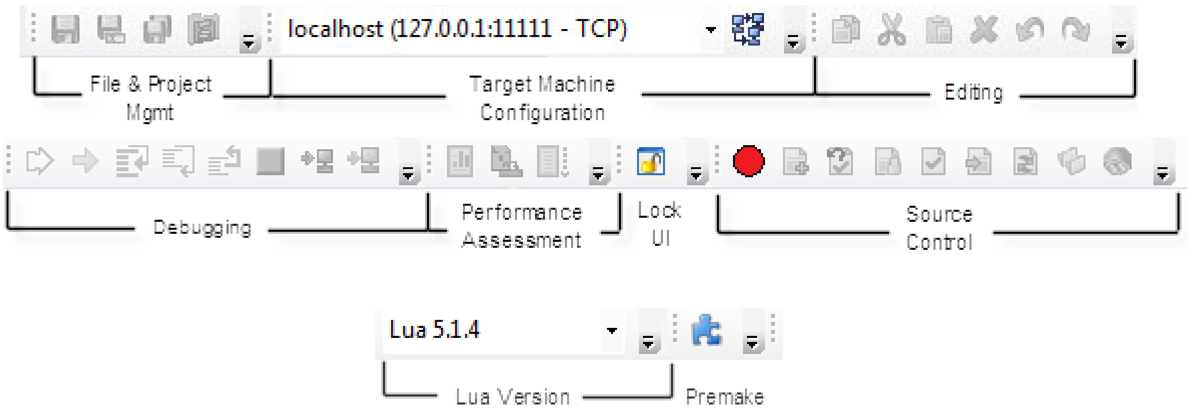
This chapter introduces aspects of the SLED interface which may be particularly helpful or not immediately intuitive to the new user. In particular, it discusses:

- The SLED Toolbar
- Customizing the Interface
- Sorting Data
- The TTY Window

The SLED Toolbar

The following figure shows the SLED toolbar, split in three pieces for the sake of illustration:

Figure 1 SLED Toolbar



localhost (127.0.0.1:11111 - TCP)

Lua 5.1.4

The tools are as follows:

- File and Project Management tools allow you to:
  - Save the script with which you're currently working.
  - Save all open scripts.
  - Save your entire project.
- Target Machine Configuration allows you to:
  - Select the machine to which to connect. For more information about target machine configurations and connecting, see *Getting Started with SLED and Lua*.

    - Manage remote targets.
- Editing tool functions are standard to most Windows applications: Copy, Paste, Undo, Redo, and so on.
- Debugging functions allow you to:
    - Start, pause, or resume script execution. For more information, see Pausing and Resuming Scripts.
    - Execute the next line of a paused script. For details on doing this, see Stepping Scripts.
    - Step over or out of a function. For details, see Stepping Scripts.
    - Connect to or disconnect from the selected target machine. For information on doing this, see *Getting Started with SLED and Lua*.
- Performance Assessment functions allow you to:
    - Toggle the Lua Profiler on and off. For details, see Lua Profiler.
    - Toggle memory tracing on and off. For more information, see Tracing Memory.
    - Compile Lua scripts.
- Lock UI allows you to freeze the user interface. For details on changing the user interface, see Manipulating Windows.
- Source Control functions allow you to manage Lua source changes.
- Lua Version allows you to select the Lua version you want to use. Support for Lua 5.2.3 is still experimental.
- Premake allows you to create a SLED project from a running Premake instance by reading shared memory set up by the SLED Lua C Library. This is an experimental facility to help debug generating scripts with Premake. As of this writing, the Premake executable with SLED debugging support added has not been released.

Customizing the Interface

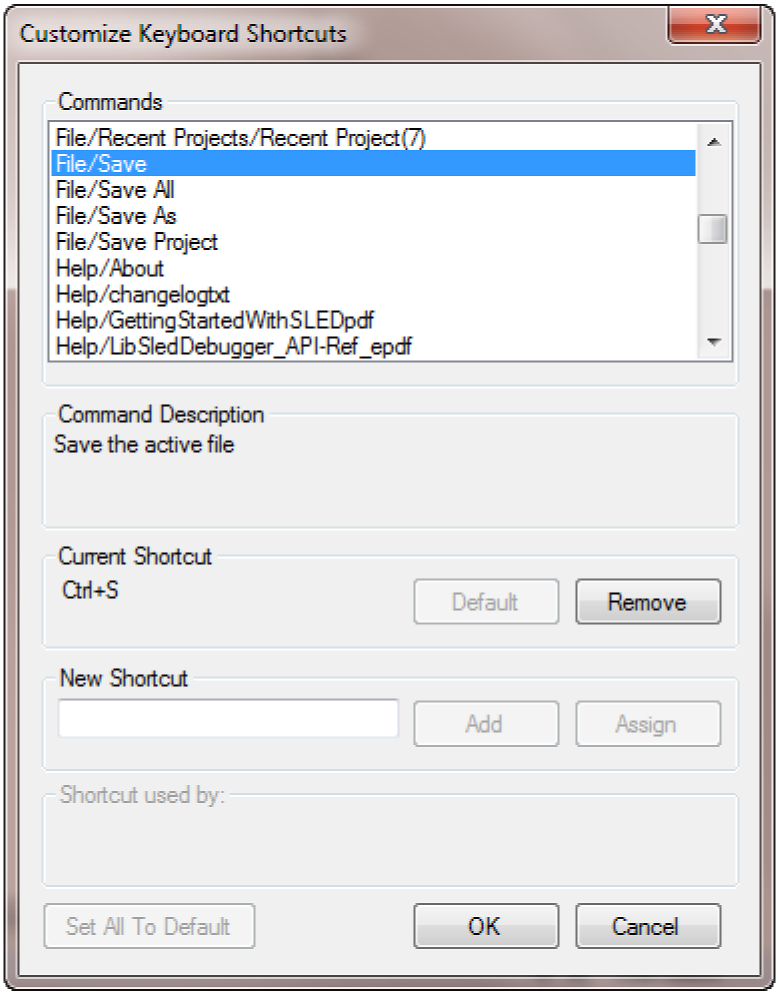
SLED allows you to customize your keyboard shortcuts and the layout of its windows.

Keyboard Shortcuts

To change or add keyboard shortcuts:

1. Select **Edit > Keyboard**. The **Customize Keyboard Shortcuts** window appears.

Figure 2 **Customize Keyboard Shortcuts Window**



Command Description: Save the active file

Current Shortcut: Ctrl+S

2. Select the command whose shortcut you wish to modify.
3. Press the desired key combination in the **New Shortcut** field, and click **Assign**.
4. Repeat this procedure for any other commands whose shortcuts you wish to modify.
5. Click **OK** to save your changes and close the window.

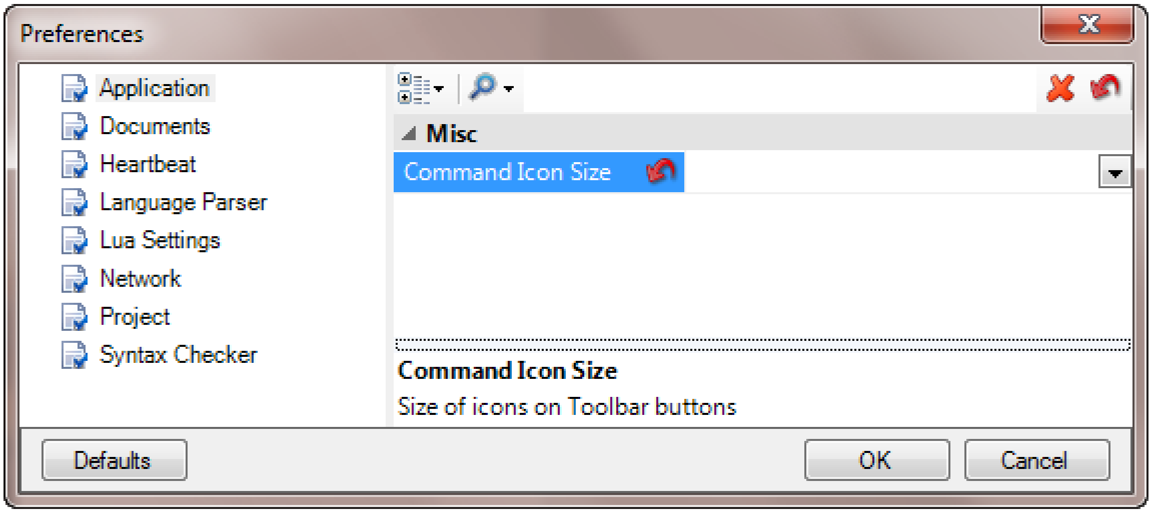
Note:
You do not need to use the **Remove** button unless you wish to leave a command without any keyboard shortcut.

Preferences

You can also change preferences, such as various project and Lua settings. For example, to change the size of the icons on the toolbar:

1. Select **Edit > Preferences**. The following window appears.

Figure 3 Resizing Command Icon (Command Icon Size Highlighted)



Command Icon Size

Size of icons on Toolbar buttons

2. Click the **Applications** item in the left pane and then click **Command Icon Size** line in the right pane. A dropdown menu appears on the right side of the line, allowing you to select from multiple icon sizes.
3. Select the desired size.
4. Click **OK** to save your changes and close the window. Click **Cancel** to close the dialog without changing preferences.

Selecting other categories in the left pane allows you to set other preferences.

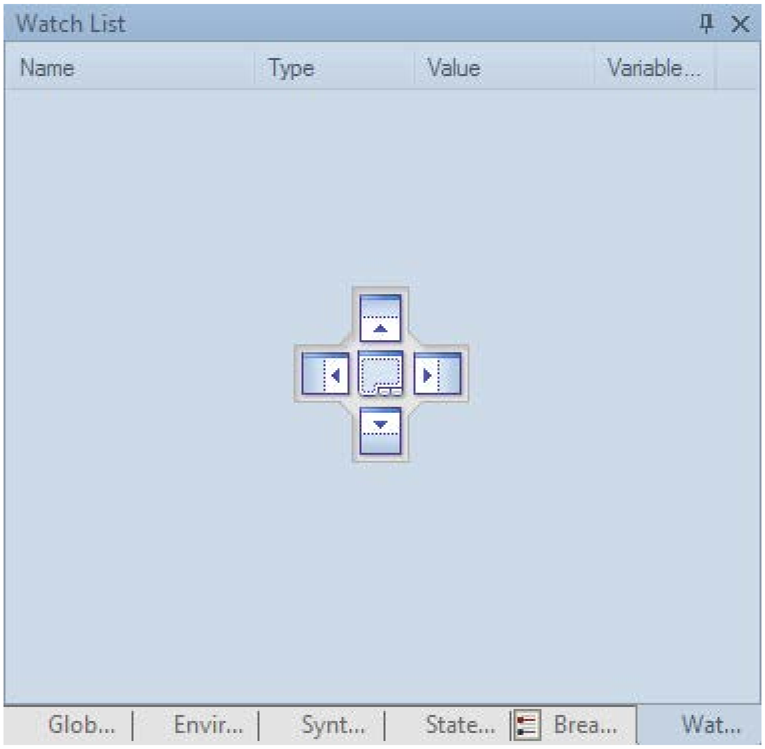
Manipulating Windows

You can undock the various SLED windows to place them anywhere on your screen you wish. You can also customize where to dock windows within the SLED application display.

To undock, double-click the title bar of a docked window. The window leaves its position and reappears as an independently movable and resizable window.

To dock a window at a specific position, drag the window, and align the pointer icon with one of the docking arrows shown in the following figure.

Figure 4 Docking Arrows



The left arrows dock the window vertically along the left side, the up arrows dock the window horizontally along the top, and so on. As you drag the window over other windows, arrows appear showing the possible docking positions. If you drag the window over an arrow, a highlighted region shows where the new window location would be if you released the mouse button there.

Changing the Theme

You can change SLED's "skin", that is, its overall color theme.

Select **View > Load Skin…** to specify the skin file. Skin files reside in the folder SLED.vs[version number]\Resources\Skins. To return to the default color scheme, select **View > Reset Skin to Default**.

You can use skin files from other applications.

Sorting Data

In windows showing quantitative data, you can select a column to sort on by clicking the header of the appropriate column. For example, you can sort the columns of the **Globals** and **Locals** windows, which show variable values.

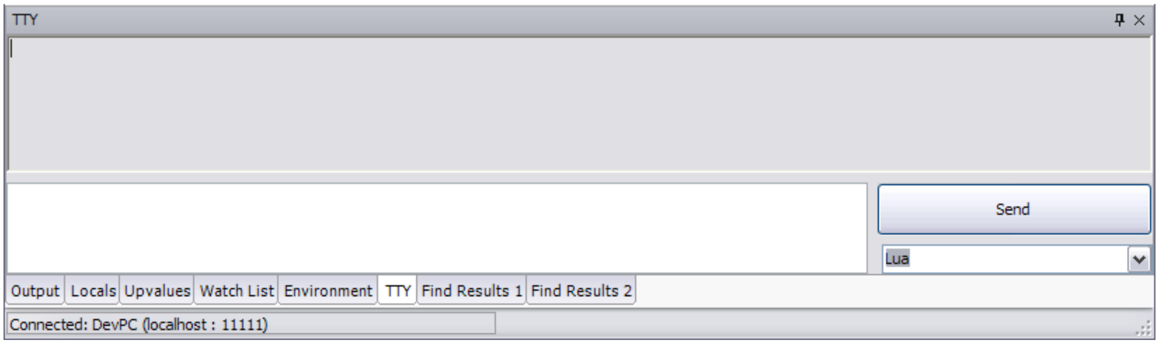
The TTY Window

SLED provides a **TTY** window that allows you to enter Lua commands on the command line while script execution is paused.

Plugins can also write messages to the **TTY** window. For more information on this, see *SLED Reference*, *LibSledDebugger API Reference*, and *LibSledLuaPlugin API Reference*.

To display the **TTY** window, click **TTY** at the top of the menu bar, or click the **TTY** tab at the bottom of the SLED window. The following window appears.

Figure 5 TTY Window



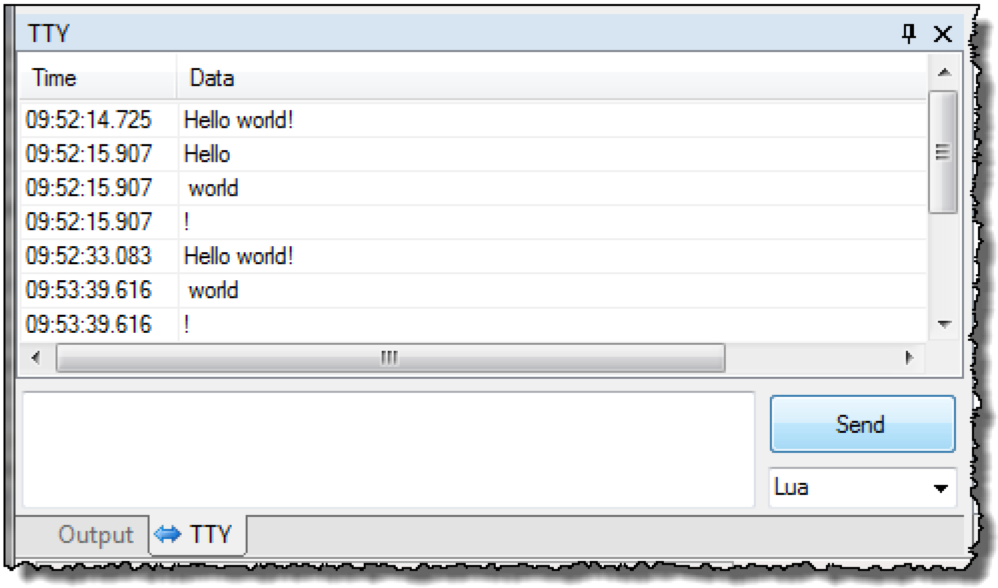
Connected: DevPC (localhost : 11111)

To send a Lua command to SLED through the **TTY** window:

1. Make sure **Lua** is selected in the drop-down menu below the **Send** button.
2. Type a valid Lua command into the window to the left of the **Send** button.
3. Click **Send**. Any return output appears in the wide upper window. You can enter **Control+Enter** as a shortcut, instead of clicking **Send**.

The figure shows how Lua commands are displayed in the **TTY** window.

Figure 6 TTY Text Resulting from Lua Script



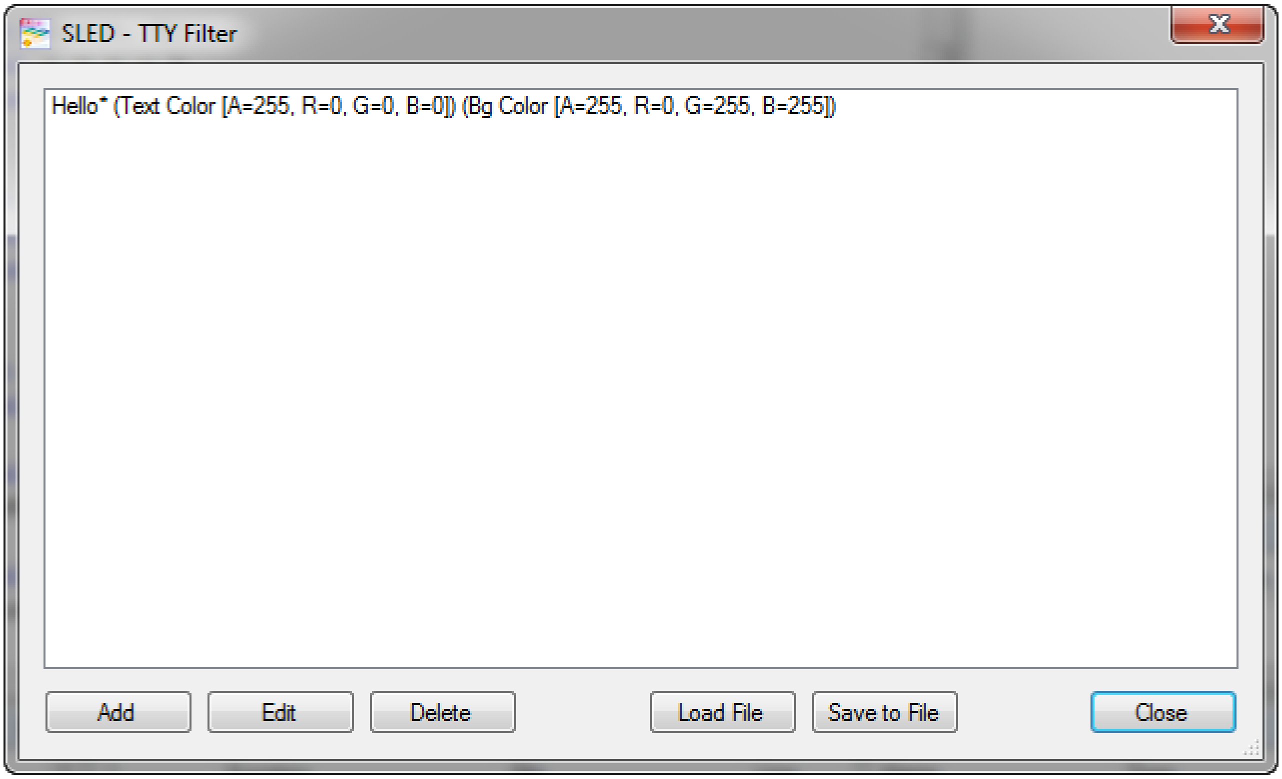
Time	Data
09:52:14.725	Hello world!
09:52:15.907	Hello
09:52:15.907	world
09:52:15.907	!
09:52:33.083	Hello world!
09:53:39.616	world
09:53:39.616	!

To clear the **TTY** window, select **TTY > Clear TTY Window**.

Filtering the TTY Window

You can filter text written to the **TTY** window. Select **TTY > Filter TTY Output** to display the **TTY Filter** dialog. It lists all the existing TTY filters:

Figure 7 TTY Filter List Window

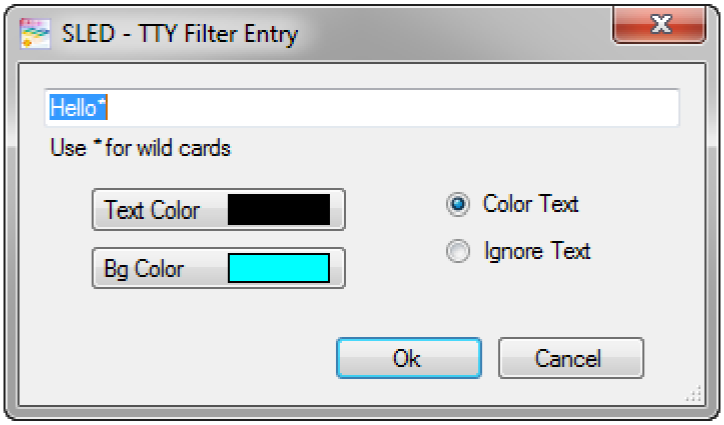


Hello* (Text Color [A=255, R=0, G=0, B=0]) (Bg Color [A=255, R=0, G=255, B=255])

Clicking the **Add** button displays the **TTY Filter Entry** dialog to create a filter. Selecting a filter and clicking **Edit** displays the **TTY Filter Entry** dialog with the values for the selected entry. **Delete** deletes selected filter entries. **Load File** displays a dialog to select a TTY filters file to load, previously created by clicking **Save to File** and saving a filter list.

In the **TTY Filter Entry** dialog, enter the text to filter, using "*" for a wild card. If you select **Color Text**, you can select colors for the text and background of text that matches the filter. If you select **Ignore Text**, text matching the filter is not written to the **TTY** window.

Figure 8 **TTY Filter Entry Window**


SLED - TTY Filter Entry

Hello*

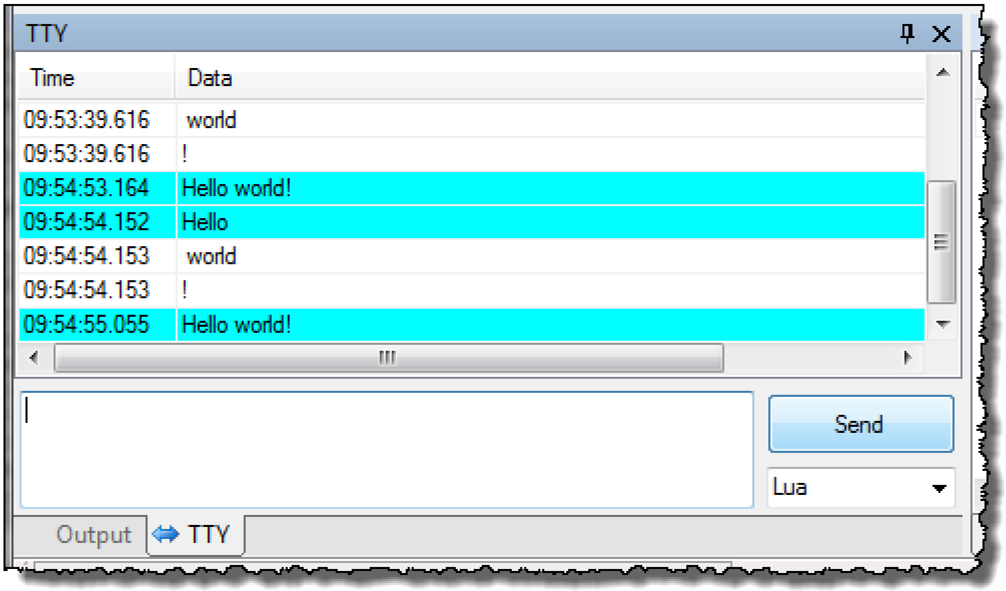
Use * for wild cards

● Color Text

○ Ignore Text

When you run a script with this filter, the lines matching the filter are highlighted with the selected color, as seen in the figure.

Figure 9 **Filtered TTY Window**


TTY

Time	Data
09:53:39.616	world
09:53:39.616	!
09:54:53.164	Hello world!
09:54:54.152	Hello
09:54:54.153	world
09:54:54.153	!
09:54:55.055	Hello world!

3 Creating, Editing, Compiling, and Managing Lua Scripts

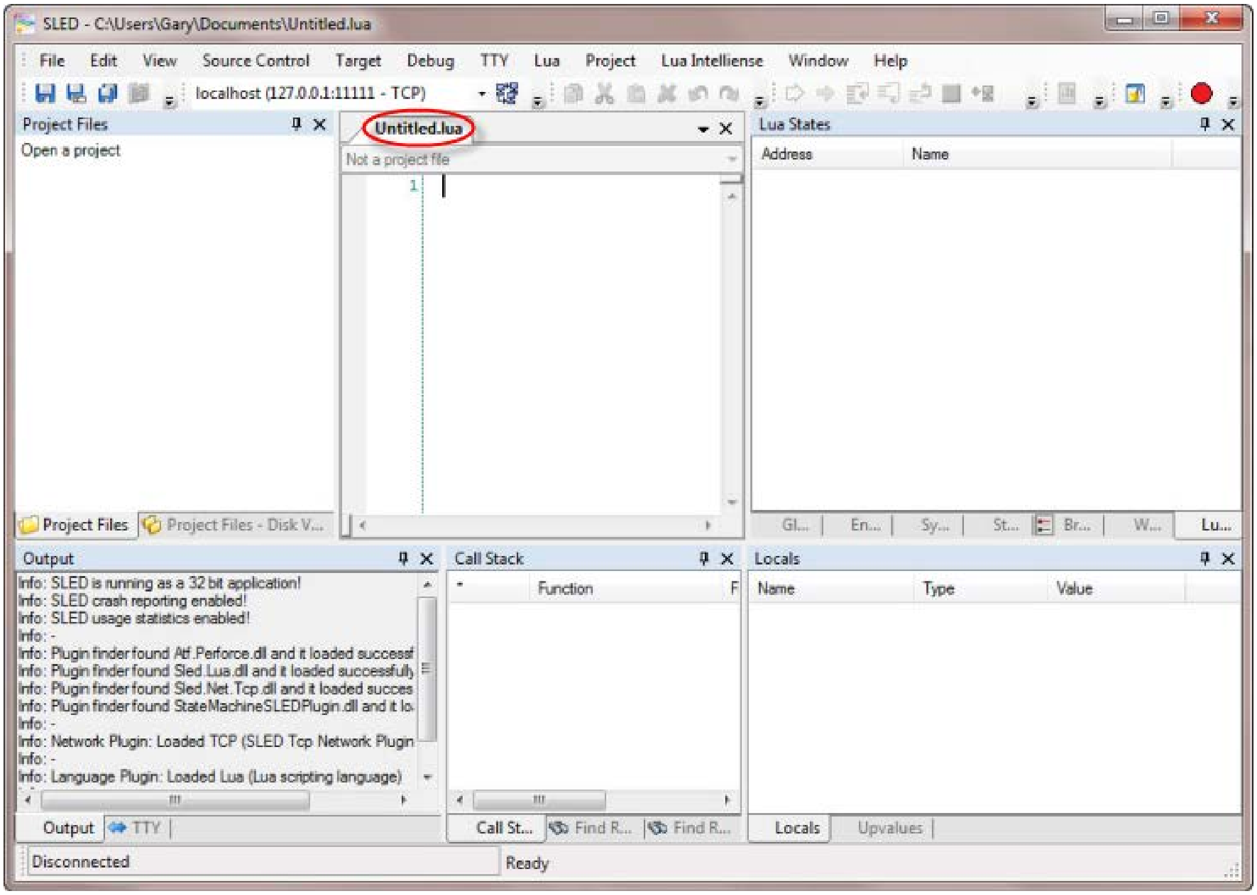
Each Lua script inhabits its own file. Lua scripts grouped together for a specific purpose constitute a project. This chapter explains:

- Creating Script Files
- Editing Script Files
- Compiling Script Files
- File Management Functions
- Creating Projects
- Managing Projects

Creating Script Files

SLED allows you to create Lua scripts from scratch. To create a Lua script, select **File > File > New > Lua**. A script-editing window appears, as shown in the figure. Note that the text “Not a project file” is displayed below the title to indicate the new file is not a part of the project. You can now enter a Lua script in the new window.

Figure 10 Empty Lua Script-Editing Window (Name Tab Circled)



```
Info: SLED is running as a 32 bit application!
Info: SLED crash reporting enabled!
Info: SLED usage statistics enabled!
Info: -
Info: Plugin finder found Atf.Perforce.dll and it loaded successf
Info: Plugin finder found Sled.Lua.dll and it loaded successfully
Info: Plugin finder found Sled.Net.Tcp.dll and it loaded succes
Info: Plugin finder found StateMachineSLEDPlugin.dll and it lo
Info: -
Info: Network Plugin: Loaded TCP (SLED Tcp Network Plugin
Info: -
Info: Language Plugin: Loaded Lua (Lua scripting language)
```

Opening Script Files

To open a file in the currently open project, double-click its name in the **Project Files** window. The file opens in a script-editing window.

To open any script file, select **File > File > Open** or press **Ctrl+O** to display an **Open** dialog. Navigate to and select the script file you want to open.

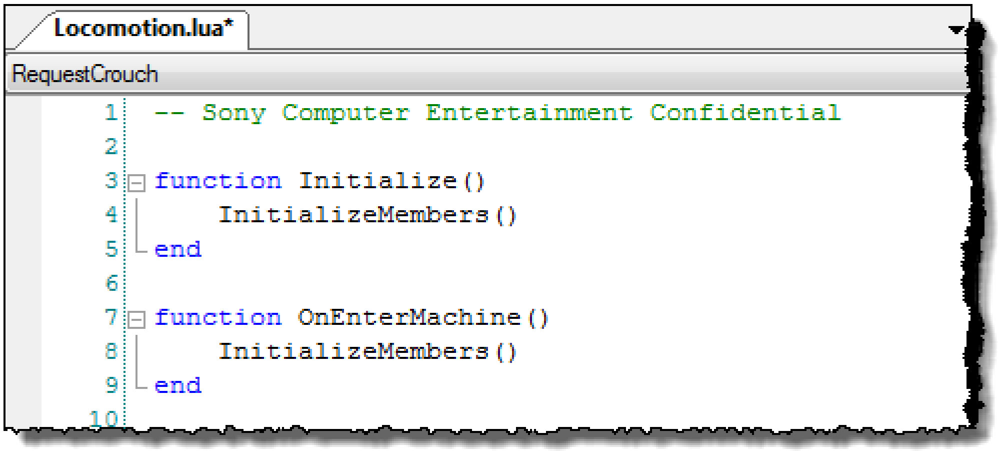
You can also select **File > Recent** to display a list of items for recently opened script files. Select the file you want to open from the list.

Editing Script Files

Whether you are creating a new Lua file, or working with an existing one, procedures for editing are identical.

As you type, SLED parses and highlights script syntax, making the script more easily readable. SLED also displays an asterisk next to the name of the script file when the content of the file has changed since the file was last saved.

Figure 11 **Syntax Highlighting and Asterisk Indicating File "Dirtiness"**



```
-- Sony Computer Entertainment Confidential

function Initialize()
    InitializeMembers()
end

function OnEnterMachine()
    InitializeMembers()
end
```

SLED also checks Lua syntax each time you save a script file. To view the **Syntax Errors** window, select the **Window** menu and then check **Syntax Errors** in the list. Double-clicking on an error takes you to its line in a script-editing window.

Note:
SLED can only check the syntax of a file that is included in a project. See Creating Projects for more information about including files in projects.

In addition to syntax highlighting and checking, SLED provides the standard Windows suite of editing commands: Copy, Paste, Undo, and so on.

Compiling Script Files

SLED includes a built in Lua compiler, allowing you to compile Lua scripts from within SLED for the supported platforms. You can also adjust the size (in bytes) of data types such as `int`, `size_t`, and `lua_Number`.

You can have multiple compiler configurations for a project. The built-in compiler performs validation as well.

Access the Lua compiler from the **Lua** menu, or by using the **Compiler** button in the toolbar.

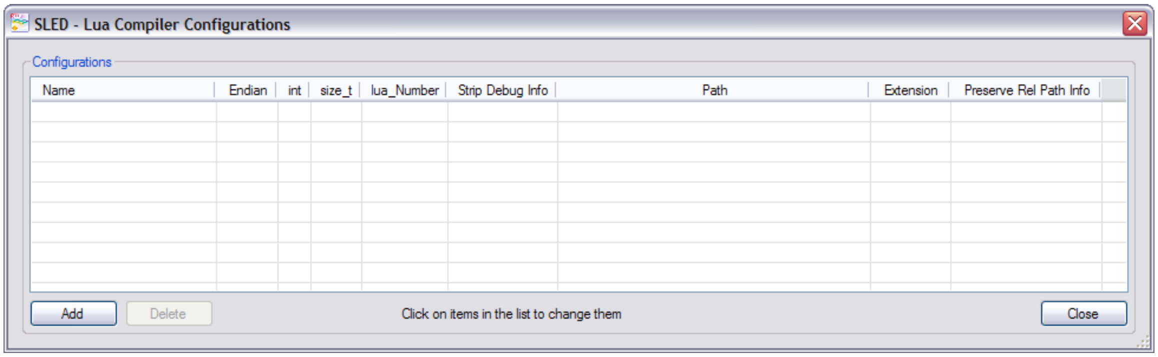
Note:
You must create at least one compiler configuration to be able to compile Lua scripts.

To access the SLED Lua compiler functionality, open a project. The **Compile** toolbar button and **Lua > Compile** are both enabled.

Add a Compiler Configuration

As noted, you must first add a compiler configuration before compiling a Lua script. To do this, select **Lua > Compiler Settings**. The **Lua Compiler Configurations** window appears, similar to the following figure.

Figure 12 Lua Compiler Configuration Window

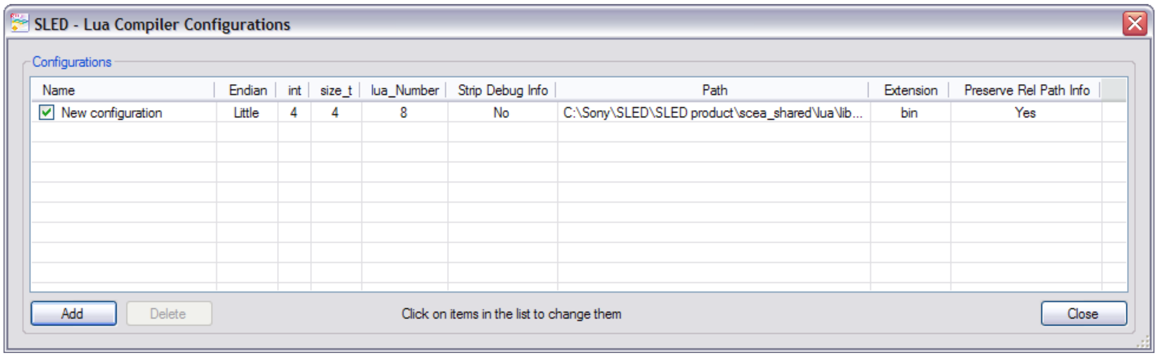


Name	Endian	int	size_t	lua_Number	Strip Debug Info	Path	Extension	Preserve Rel Path Info

Click on items in the list to change them

To add a compiler configuration, click **Add**. A default Lua compiler configuration appears in the **Configurations** window, similar to the following figure:

Figure 13 Default Lua Compiler Configuration



SLED - Lua Compiler Configurations
Configurations

Name	Endian	int	size_t	lua_Number	Strip Debug Info	Path	Extension	Preserve Rel Path Info
☑ New configuration	Little	4	4	8	No	C:\Sony\SLED\SLED product\scea_shared\lua\lib...	bin	Yes

Add
Delete
Click on items in the list to change them
Close

To change any value, click the field to select it, then enter or select the desired value. You type in new values for fields such as **Name** and **Extension**, and select from allowed values for fields such as **Endian**, **int**, **size_t**, **lua_Number**, and **Strip Debug Info**.

Note:
If you strip Debug information from anything, you can't debug it with SLED.

Path is the directory in which to place compiled Lua script output. By default, this is the Asset Directory. You can click the **Path** field to display a folder browser window that allows you to select another path.

The **Preserve Rel Path Info** field controls preserving relative path information. For details on using this field, see Preserve Rel Path Info.

Check the name of the configuration to choose it for Lua compilation.

When you are satisfied with the configuration, click **Close**. SLED saves the configuration.

Preserve Rel Path Info

The **Preserve Rel Path Info** field allows you to set whether or not SLED preserves relative path information when saving the output of compiled Lua scripts.

Note:
If paths do not match between SLED and the runtime, breakpoints are not hit.

By default, SLED sets the Asset Directory as the directory in which to place compiled Lua script output. This is the **Path** field in the compiler configuration, described in Add a Compiler Configuration. You can set a different output directory for your project's compiled Lua scripts at any time.

If you set **Preserve Rel Path Info** to **Yes**, SLED preserves the relative path information (such as sub-directories) and places the compiled Lua script output in a directory relative to the output directory you set. This is the default. If you set **Preserve Rel Path Info** to **No**, SLED does not preserve relative path information and places the compiled Lua script output directly in the specified output directory.

For example, suppose that your asset directory is this path:

```
C:\scea\games\ratchet_and_clank\scripts\
```

And suppose that you have the following Lua script assets:

```
C:\scea\games\ratchet_and_clank\scripts\weapons\player\rynov.lua
C:\scea\games\ratchet_and_clank\scripts\weapons\player\some\other\gun\other.lua
```

By default, SLED sees your Lua script files with relative paths:

```
weapons\player\rynov.lua
weapons\player\some\other\gun\other.lua
```

Suppose you set up a compiler configuration using `.bin` as the extension of the compiled script file, the Asset Directory as the compiled Lua script output directory, and **Preserve Rel Path Info** set to **Yes**. The resulting `.bin` files are at these locations:

```
C:\scea\games\ratchet_and_clank\scripts\weapons\player\rynov.bin
C:\scea\games\ratchet_and_clank\scripts\weapons\player\some\other\gun\other.bin
```

If you set **Preserve Rel Path Info** to **No**, the resulting `.bin` files are here:

```
C:\scea\games\ratchet_and_clank\scripts\rynov.bin
C:\scea\games\ratchet_and_clank\scripts\other.bin
```

Suppose that you change the compiled output directory to something else, for example:

```
C:\MyGames\Lua\Bin
```

And suppose that you set **Preserve Rel Path Info** to **Yes**. The resulting `.bin` files are here:

```
C:\MyGames\Lua\Bin\weapons\player\rynov.bin
C:\MyGames\Lua\Bin\weapons\player\some\other\gun\other.bin
```

If you set **Preserve Rel Path Info** to **No** and recompile, the resulting `.bin` files are in these locations:

```
C:\MyGames\Lua\Bin\rynov.bin
C:\MyGames\Lua\Bin\other.bin
```

Select and Update Compiler Configurations

You can select the configuration to use, or change, add, or delete Lua compiler configurations at any time.

Select **Lua > Lua Compiler Settings** to display the saved configurations in the **Lua Compiler Configurations** window shown in Figure 13.

- To select a configuration to use for compilation, check the box next to the configuration name.
- To change the value of any configuration fields, click the field to select it, and then enter or select the desired value.
- To add a new configuration, click **Add** and proceed as described in Add a Compiler Configuration.
- To delete a configuration, click the configuration entry to select the entire entry, and then click **Delete**.

Click **Close** to save your changes.

Compiling a Script

To compile a script, select **Lua > Compile** or click the **Compile** toolbar button. Click the **Output** window to view the compilation output.

Clear the **Output** window by selecting **Project > Clear Output Window**.

If you have not yet created a compiler configuration, SLED prompts you to do so: a message window appears, explaining that you must set up a compiler configuration. Click **OK** to display the **Lua Compiler Configurations** window. Follow the directions described in Add a Compiler Configuration.

File Management Functions

Right-click on a script file in the **Project Files** window to access the file management functions, including:

- Open the script file in a SLED script-editing window or in another application.
- Rename the file.
- Add other files to the project.
- Explore the folder containing the file in Windows Explorer.
- Remove the file from the project.

Note:
Removing a script file from the current project does not delete it; the file remains intact in its folder.

Creating Projects

A project is a group of Lua script files that, collectively, control an element of game logic and flow. Such an element may be small, like a mini-game, or large, such as an entire game. Although grouping Lua script files into a project is not mandatory, you must do so in order to take advantage of SLED's debugging features.

Project files reside in the project directory, and are the heart of a project; they define its scope and characteristics. The SLED project file stores names and locations of the files comprising the project, with the following Information stored in a temporary user settings file:

- Functions used in the project.
- Breakpoint locations and conditions.

The default project file extension is `.spf`. The SLED User Settings file extension is `.sus`.

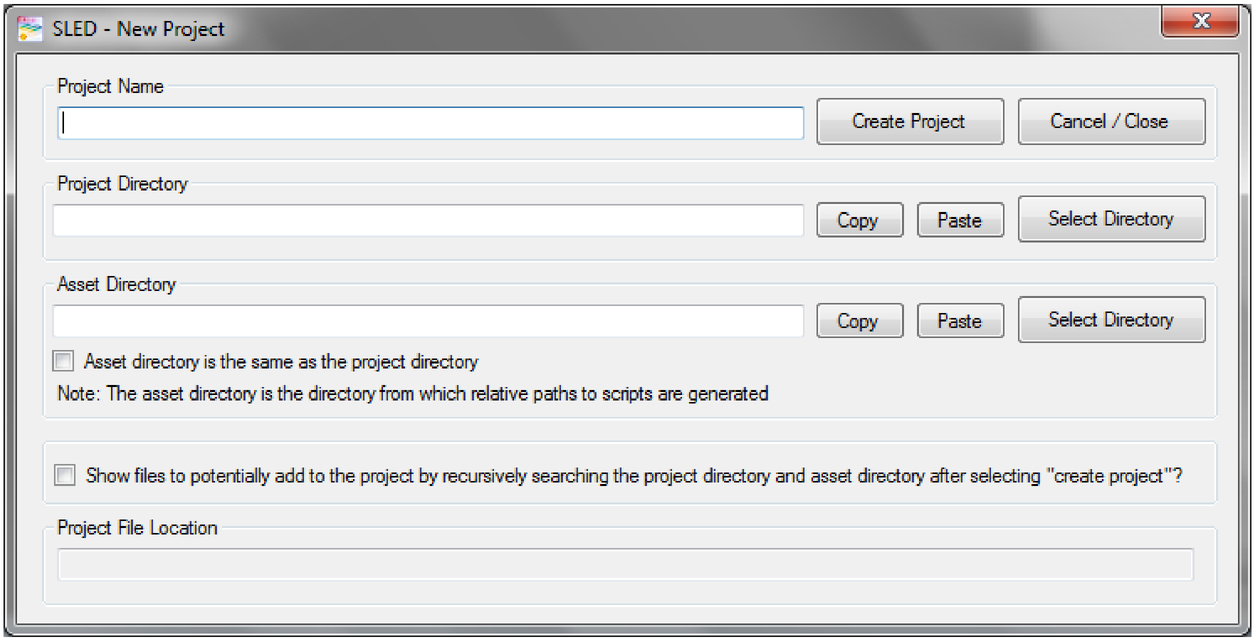
A project's Lua script files may reside anywhere.

Note:
Although the Lua script files do not have to reside in the asset directory, their paths must match up between the target machine and SLED to enable breakpoints to be hit.

To create a Lua project and project files:

1. Select **File > Project > New**. The **New Project** dialog box appears.

Figure 14 New Project Dialog Box



□ Asset directory is the same as the project directory

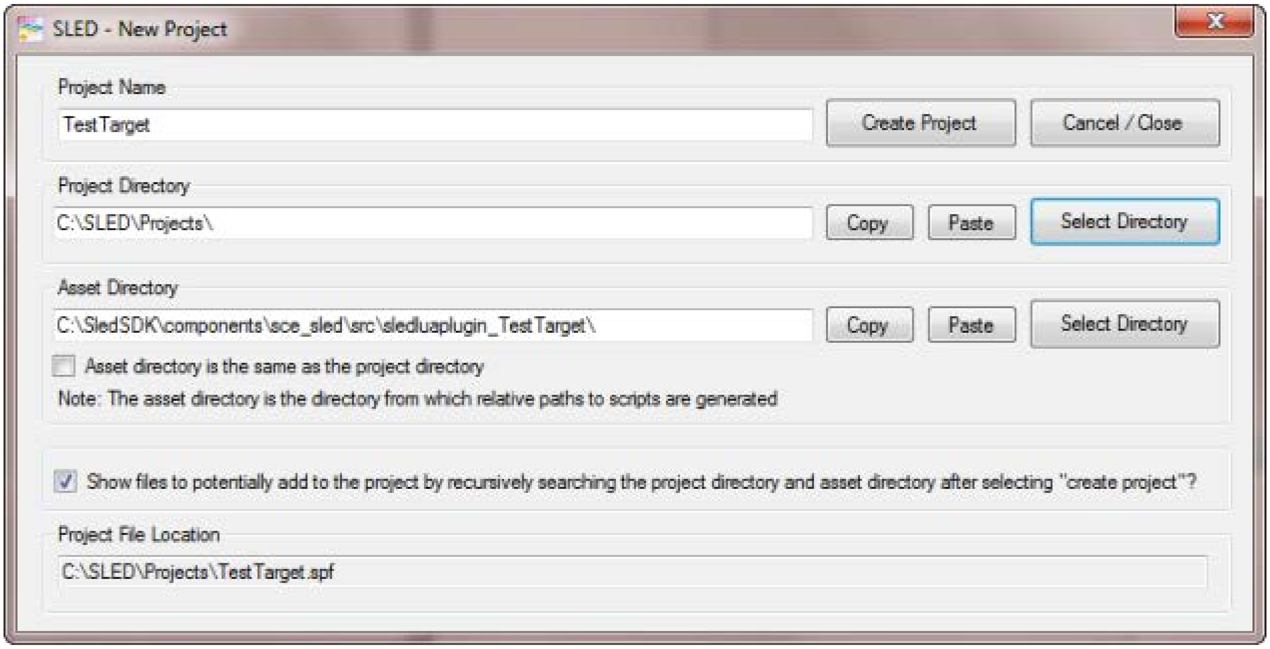
Note: The asset directory is the directory from which relative paths to scripts are generated

□ Show files to potentially add to the project by recursively searching the project directory and asset directory after selecting "create project"?

2. In the **Project Name** field, specify a name for your project. The specified name, with the extension `.spf` appended to it, serves as the name of the project file.

3. Click **Select Directory** in to the **Project Directory** pane. The **Browse For Folder** dialog box appears.
4. Select the folder where you store your project file, and then click **OK**. The **New Project** dialog box reappears.
5. Click **Select Directory** in the **Asset Directory** pane. The **Browse For Folder** dialog box appears.
6. Select the folder that contains the Lua scripts, and click **OK**. The **New Project** dialog box reappears.

Figure 15 **New Project Dialog Box With Directory Information Specified**



Project Name: TestTarget

Project Directory: C:\SLED\Projects\

Asset Directory: C:\SledSDK\components\sce_sled\src\sledluaplugin_TestTarget\

☐ Asset directory is the same as the project directory

Note: The asset directory is the directory from which relative paths to scripts are generated

☑ Show files to potentially add to the project by recursively searching the project directory and asset directory after selecting "create project"?

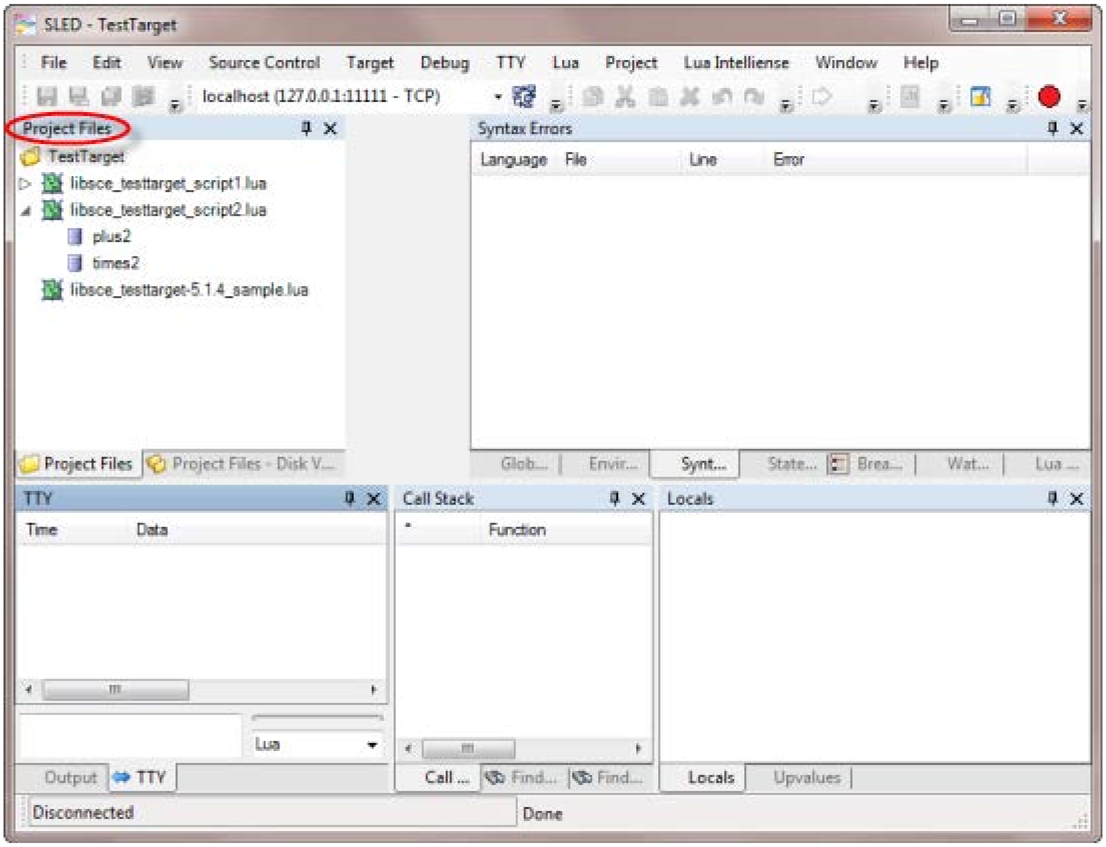
Project File Location: C:\SLED\Projects\TestTarget.spf

7. Click **Create Project** in the **Project Name** pane. The **New Project** dialog box closes, and the project file is created.

To display the **Project Files** window, select **Window > Project Files**. The **Project Files** window shows, in tree form:

- Project name.
- Script files associated with the project.
- Functions contained in the script files.

Figure 16 Project Files Window (Title Circled)


SLED - TestTarget
File Edit View Source Control Target Debug TTY Lua Project Lua Intelliense Window Help
localhost (127.0.0.1:11111 - TCP)
Project Files
TestTarget
libsce_testtarget_script1.lua
libsce_testtarget_script2.lua
plus2
times2
libsce_testtarget-5.1.4_sample.lua
Project Files
Project Files - Disk V...
Syntax Errors
Language File Line Error
Glob... Envir... Synt... State... Brea... Wat... Lua ...
TTY
Time Data
Lua
Output TTY
Call Stack
Function
Call ... Find... Find...
Locals
Locals Upvalues
Disconnected
Done

At any time, you can check your directory paths or change your asset directory:

- To check directory paths, select **Project > View Project Paths** to display the **Project Paths** dialog, listing paths of all asset files.
- To change the asset directory, select **Project > Change Asset Directory** to display a folder browser, and select the directory that serves as the new asset directory.

Managing Projects

You can open projects, add existing Lua script files to your project, or you can add new scripts to the project as you create them.

Opening a Project

To open a project, select **File > Project > Open** or press **Ctrl+Shift+O** to display an **Open** dialog that allows you to choose the project you wish to open.

You can also select **File > Recent Projects** to display a list of items for recently opened projects. Select the project you want to open from the list.

Opening a project closes the currently opened project, if any.

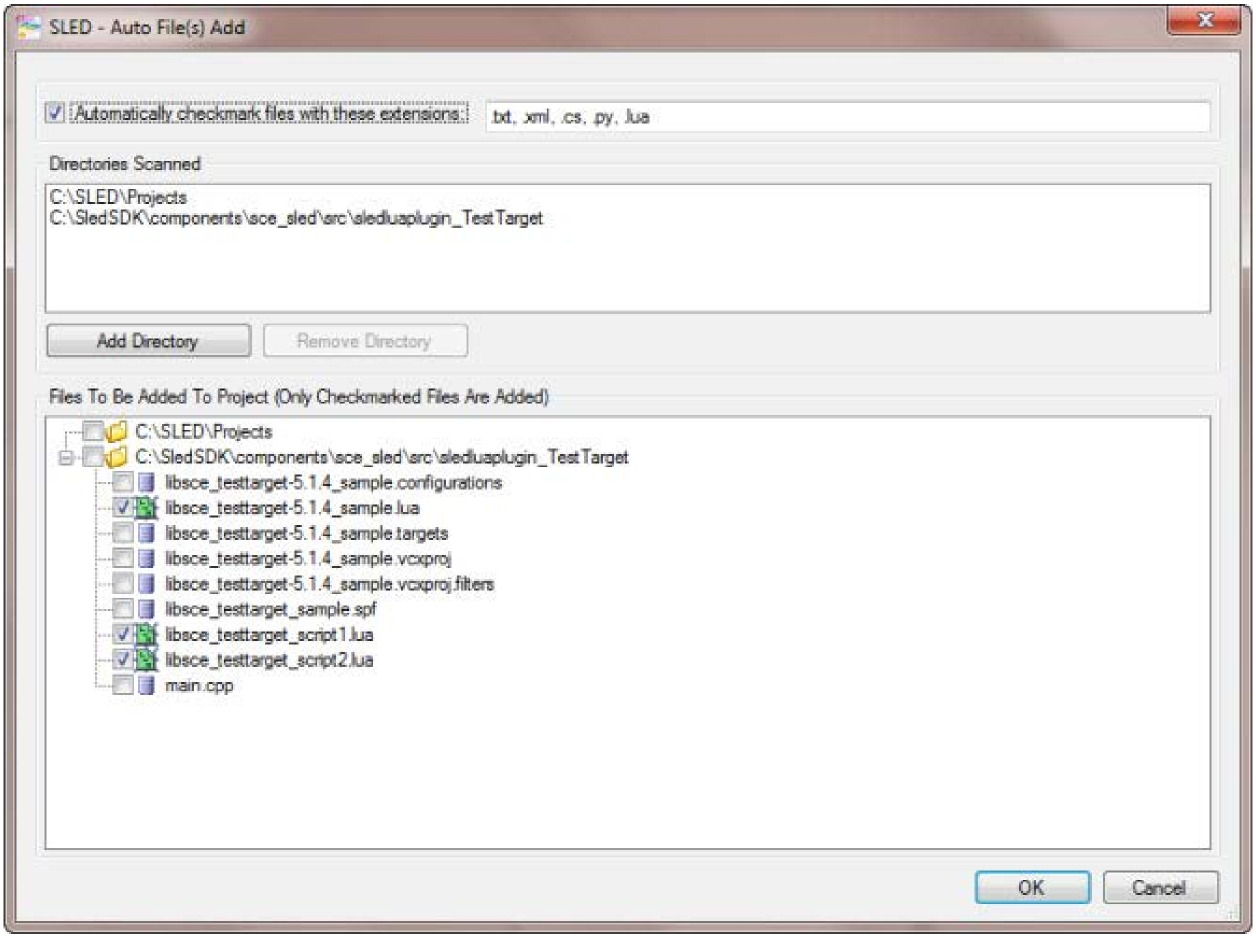
Adding Existing Scripts

There are two ways to add an existing script to your project:

- You can drag the desired files from their folder in Windows Explorer to the **Project Files** window in SLED.
- You can use the **Project > Add Files…** command to add Lua scripts to your project.

Select **Project > Add Files…** to display the **Auto File(s) Add** window.

Figure 17 Auto File(s) Add Window



☑ Automatically checkmark files with these extensions: .txt, .xml, .cs, .py, .lua

Directories Scanned

C:\SLED\Projects
C:\SledSDK\components\sce_sled\src\sledluaplugin_TestTarget

Files To Be Added To Project (Only Checkmarked Files Are Added)

- □ C:\SLED\Projects
- □ C:\SledSDK\components\sce_sled\src\sledluaplugin_TestTarget
  - □ libsce_testtarget-5.1.4_sample.configurations
  - ☑ libsce_testtarget-5.1.4_sample.lua
  - □ libsce_testtarget-5.1.4_sample.targets
  - □ libsce_testtarget-5.1.4_sample.vcxproj
  - □ libsce_testtarget-5.1.4_sample.vcxproj.filters
  - □ libsce_testtarget_sample.spf
  - ☑ libsce_testtarget_script1.lua
  - ☑ libsce_testtarget_script2.lua
  - □ main.cpp

The first field, **Automatically checkmark files with these extensions**, has a list of extensions. You can specify multiple extensions, using commas as delimiters. When checked, the dialog automatically checks all files with these extensions in the **Files To Be Added** list at the bottom of the display.

The **Directories Scanned** field shows the project and assets directories. You can add additional directories by clicking the **Add Directory** button. The **Files To Be Added to Project** field lists all files in these directories.

If you change your mind about directories or file extensions, you can re-enter the values of the first two fields.

Click **OK** to close the **Auto File(s) Add** dialog box and to add the checked files to the project. The **Project Files** window lists the files you just added to the project.

Adding Newly Created Scripts

To add a script that you created, save the file (remembering to give it the `.lua` extension), and select **File > Project > Add File**. The new script joins the others in the **Project Files** window.

Other Project Management Functions

Double-click a function name to go directly to that function in the script-editing window. Or right-click the function name and select **Goto** in the context menu.

Right-click the project icon in the Project Files window to display a context menu that allows you to:

- View project paths in the **Project Paths** dialog.
- Explore the folder that contains the project in Windows Explorer.
- Add a folder to the project.

Source Control

Source control systems can be integrated into SLED. Perforce support is provided.

To use Perforce source control, do the following:

- Ensure that the Perforce source control plugin, `Atf.Perforce.dll`, is in the SLED `Plugins` folder. Plugins reside in the `bin\sce_sled\SLED.[VS Version]\Plugins` folder of the SLED installation, where `[VS Version]` is the version of Visual Studio used to build SLED.
- Start SLED.
- Open a SLED Editor project under Perforce source control.
- Select **Source Control > Source Control Disabled – Click To Enable**.
- Select **Source Control > Source Control > Open Connection…** to display a **Connections** dialog to open a connection to the Perforce server. You must

have the proper credentials for the server. Enter the required information and click **Ok** to open the connection.

You can now use the source control commands under the **Source Control > Source Control** menu once any item in the project is selected. Commands are also available under the context menu you get by right-clicking on any script file node in the **Project Files** window.

The default file check-out behavior is to check out files as “non-exclusive”, that is, the file does not become locked to others. Files in the project tree view display a question mark icon when the source control status of the associated file is being evaluated.

This table describes source control menu items.

Table 1 Source Control Menu

Command	Use	
Source Control	Submenu for source control commands. The command menu items are enabled or disabled as appropriate.	
	Add	Add files to source control.
	Check Out	Check out files under source control. The default check-out behavior is to check out files as “non-exclusive”, that is, the file does not become locked to others.
	Check In	Check in your version of files under source control.
	History	Show file history.
	Get Latest Version	Get latest version of files under source control.
	Revert	Revert a file to its last version under source control.
	Refresh Status	Refresh the status of files under source control. Files in the project tree view display a question mark icon when the source control status of the associated file is being evaluated.
	Open Connection...	Open a connection to the Perforce server. You must have the proper credentials for the server.

4 Debugging

SLED provides an array of powerful and flexible tools to help you debug scripts. This chapter explains these topics:

- Controlling Script Execution
- Breakpoints
- Monitoring Variables
- Modifying Variables and Scripts

Before performing any of these debugging operations, you must:

- Connect the developer computer to the target machine.
- Start the executable on the target machine.
- Connect SLED to the target machine.
- Open the SLED project associated with the executable.

Controlling Script Execution

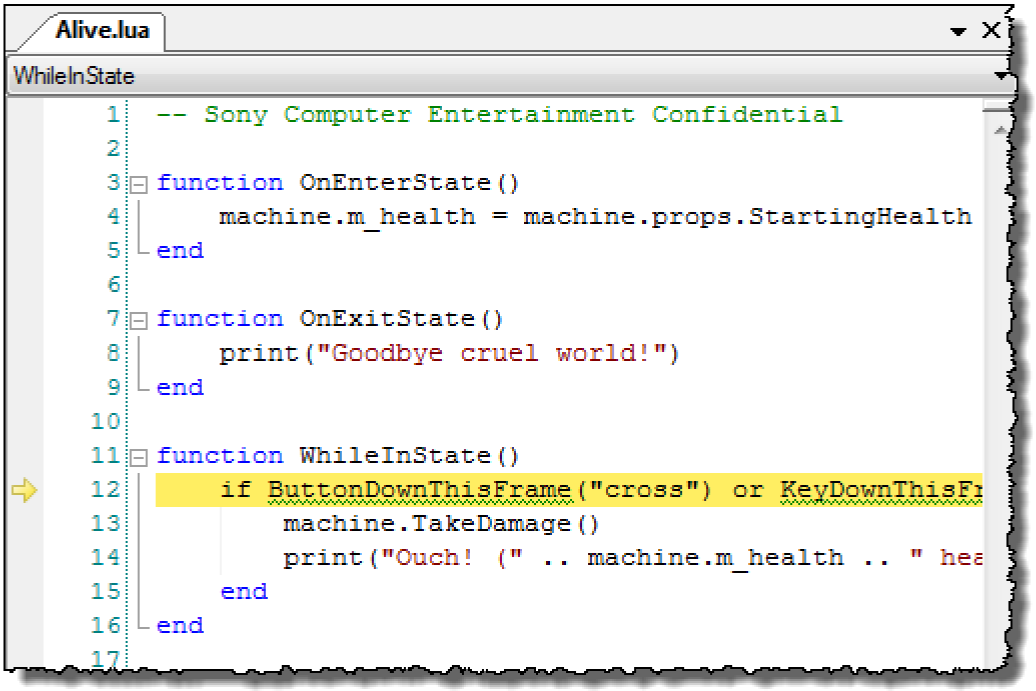
Pausing and Resuming Scripts

SLED allows you to pause and resume script execution.

To pause script execution, select **Debug > Stop**.

When you pause execution, SLED opens the script file in which execution halted, and highlights the line where the pause occurred, as shown in Figure 18.

Figure 18 **Script-Editing Window Showing Pause Location**



```
-- Sony Computer Entertainment Confidential

function OnEnterState()
    machine.m_health = machine.props.StartingHealth
end

function OnExitState()
    print("Goodbye cruel world!")
end

function WhileInState()
    if ButtonDownThisFrame("cross") or KeyDownThisFr
        machine.TakeDamage()
        print("Ouch! (" .. machine.m_health .. " hea
    end
end
```

To resume script execution, select **Debug > Start**.

To go to the statement where the script is stopped, select **Debug > Current Statement**.

Stepping Scripts

When execution is paused, you can tell SLED to continue executing the script in certain increments.

- **Debug > Step Into** advances script execution by one line each time you select it. SLED follows the script wherever it goes, entering functions when they are called, and returning from them when they end. You can also click the **Step Into** button in the toolbar's **Debug** command group.
- **Debug > Step Over** tells SLED to advance script execution by one line each time you select it. Instead of following the script when it calls a function, however, SLED steps over the function, executing it without entering its script. You can also click the **Step Over** button in the toolbar's **Debug** command group.
- **Debug > Step Out** tells SLED to finish executing the current function, return from it, and pause execution. You can also click the **Step Out** button in the toolbar's **Debug** command group.

For information about keeping track of nested and executing functions, see Following Execution Flow.

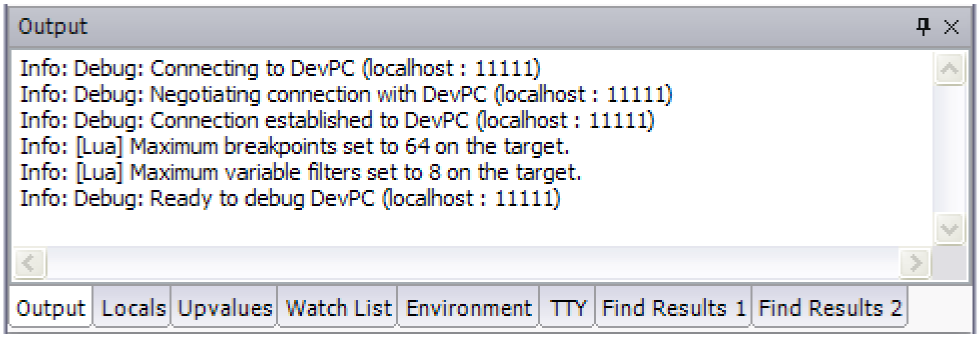
Breakpoints

You can set a breakpoint to pause script execution on a certain line. Setting a breakpoint assures that execution always pauses on that line, but you can also specify a condition so that execution pauses only when the condition is met.

When you connect to the target machine, the maximum number of breakpoints that you can set appears in the Output tab. The executable on the target machine hits the breakpoints only if SLED is connected to the target machine.

The following figure shows the maximum breakpoints message in the **Output** tab.

Figure 19 Maximum Breakpoints Message In The Output Tab

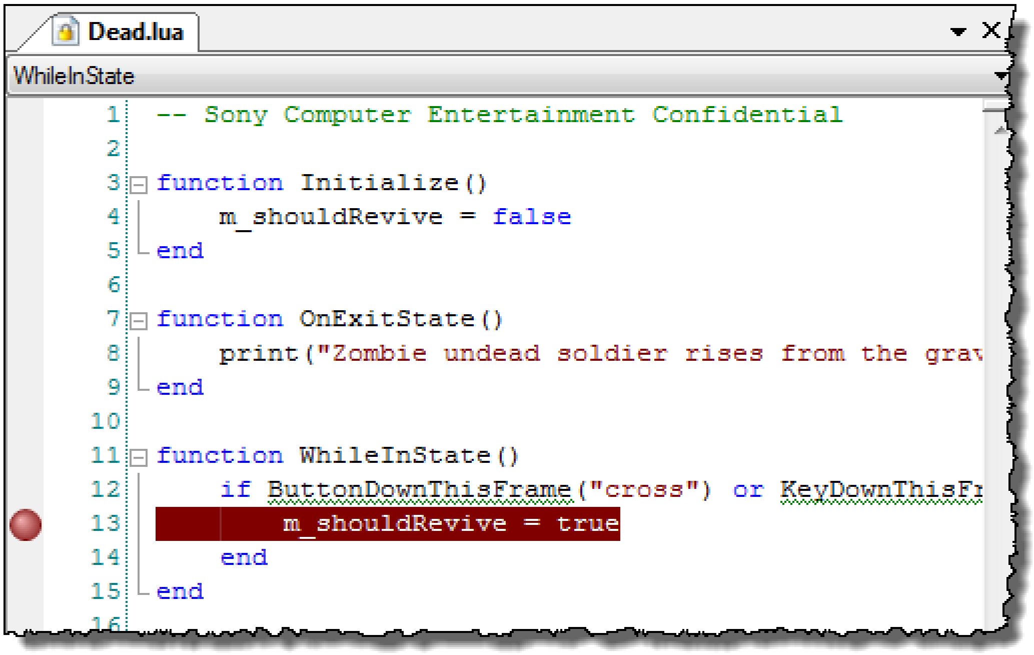


```
Info: Debug: Connecting to DevPC (localhost : 11111)
Info: Debug: Negotiating connection with DevPC (localhost : 11111)
Info: Debug: Connection established to DevPC (localhost : 11111)
Info: [Lua] Maximum breakpoints set to 64 on the target.
Info: [Lua] Maximum variable filters set to 8 on the target.
Info: Debug: Ready to debug DevPC (localhost : 11111)
```

Setting a Breakpoint

To set a breakpoint, point the cursor at the gray-shaded, vertical margin (the breakpoint bar) on the left side of the script-editing window. Align the cursor with the line on which to set the breakpoint, and click. In the figure, clicking at the location indicated by the arrow places a breakpoint at line 13. A red circle appears where you clicked, indicating that a breakpoint now exists at that line, as seen in Figure 20.

Figure 20 **Breakpoint Icon Indicating That a Breakpoint Has Been Successfully Set**



```
-- Sony Computer Entertainment Confidential

function Initialize()
    m_shouldRevive = false
end

function OnExitState()
    print("Zombie undead soldier rises from the grav
end

function WhileInState()
    if ButtonDownThisFrame("cross") or KeyDownThisFr
        m_shouldRevive = true
    end
end
```

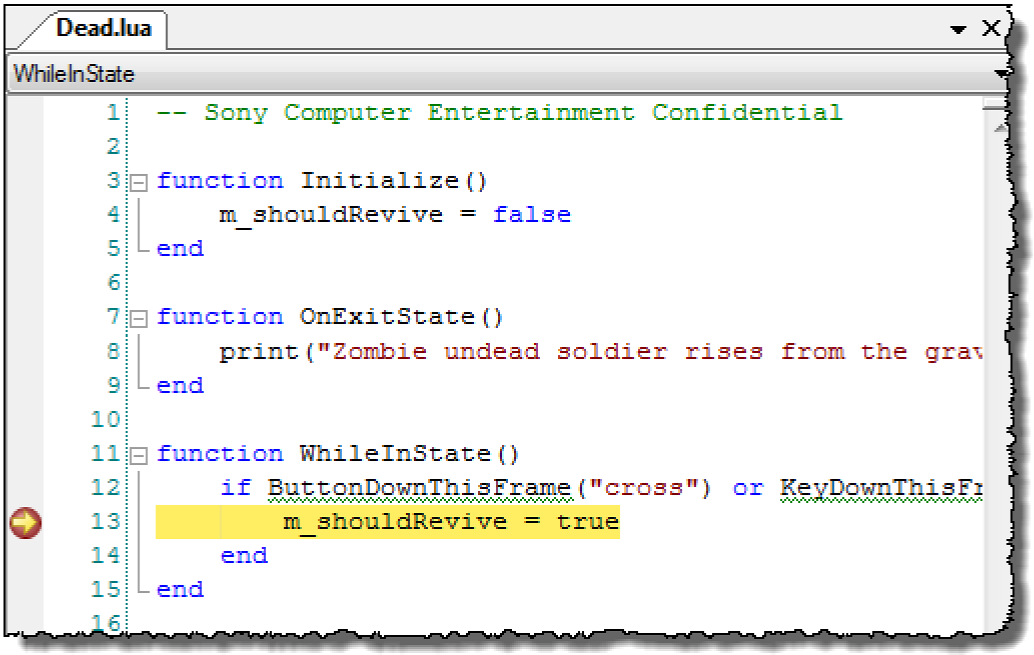
To toggle the breakpoint off, click the breakpoint icon. You can also remove all breakpoints by right-clicking anywhere in the breakpoint bar, and selecting **Remove all Breakpoints** in the context menu. For information about removing multiple, but not all, breakpoints, see Viewing and Managing Breakpoints.

You can leave a breakpoint in place—but inactive—with one of the following methods:

- Right-click the breakpoint icon, and select **Disable Breakpoint** in the context menu.
- Select **Window > Breakpoints** to display the **Breakpoints** window, right-click the breakpoint you want to disable, and select **Disable** from the context menu.

When a breakpoint halts script execution, SLED opens the script file and indicates the line where the breakpoint occurred, as shown in the following figure.

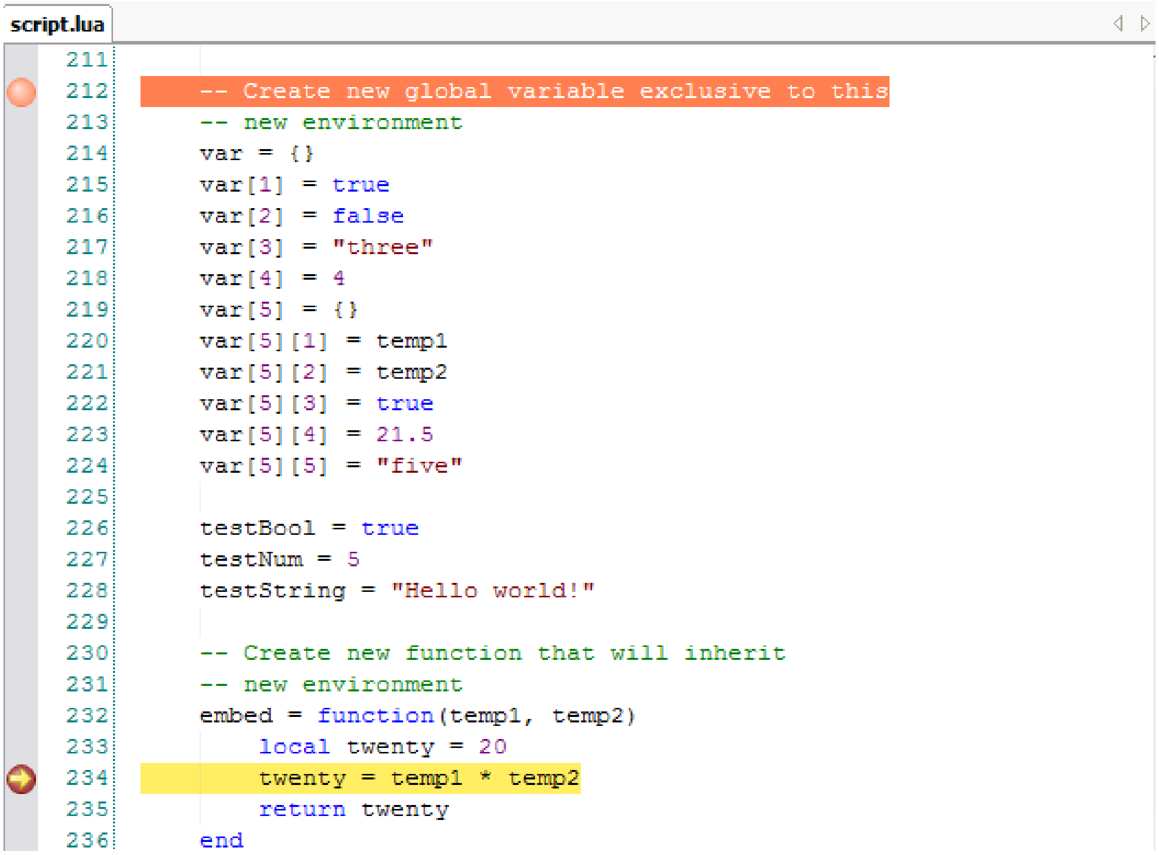
Figure 21 **Display Indicating Breakpoint Occurrence and Location**



```
-- Sony Computer Entertainment Confidential

function Initialize()
    m_shouldRevive = false
end

function OnExitState()
    print("Zombie undead soldier rises from the grav
end

function WhileInState()
    if ButtonDownThisFrame("cross") or KeyDownThisFr
        m_shouldRevive = true
    end
end
```

Some breakpoints are not hit, depending on where you place them. Those breakpoints are colored differently when SLED connects to the target machine. For example, a breakpoint on a comment line is never hit. The following figure shows a breakpoint that is not hit because it is set on a comment, and a breakpoint that has been hit. These colors are used when a script is running.

Figure 22 **Difference in Colors Between a Breakpoint that is Not Hit and a Breakpoint that is Hit**



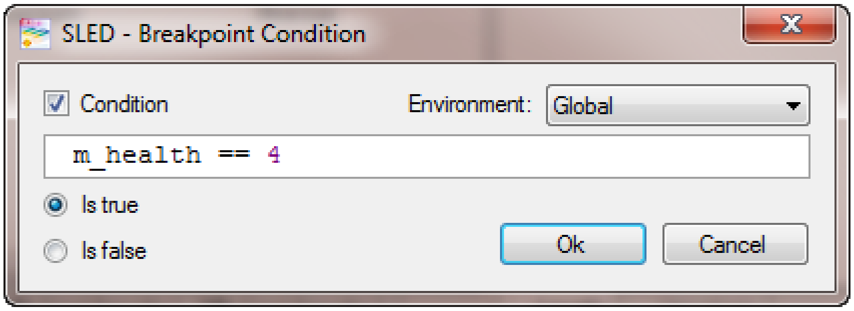
```
-- Create new global variable exclusive to this
-- new environment
var = {}
var[1] = true
var[2] = false
var[3] = "three"
var[4] = 4
var[5] = {}
var[5][1] = temp1
var[5][2] = temp2
var[5][3] = true
var[5][4] = 21.5
var[5][5] = "five"

testBool = true
testNum = 5
testString = "Hello world!"

-- Create new function that will inherit
-- new environment
embed = function(temp1, temp2)
    local twenty = 20
    twenty = temp1 * temp2
    return twenty
end
```

Setting a Breakpoint With Conditions

To set a breakpoint so that it halts script execution only if certain conditions are met:

1. Set a breakpoint, as explained in Setting a Breakpoint.
2. Right-click the breakpoint icon. A context menu appears.
3. Select **Condition** and the **Breakpoint Condition** dialog box appears.
4. Make sure **Condition** is checked, and enter a valid Lua expression to evaluate. Its syntax is highlighted as you enter the Lua expression. Select the environment of the variable in the **Environment** drop down. Select either **Is true** or **Is false**.
5. Click **Ok**. The **Breakpoint Condition** dialog box disappears, and the condition is set.

Figure 23 **Breakpoint Condition Dialog Box with Syntax Highlighted**



SLED - Breakpoint Condition

☑ Condition

Environment: Global

```
m_health == 4
```

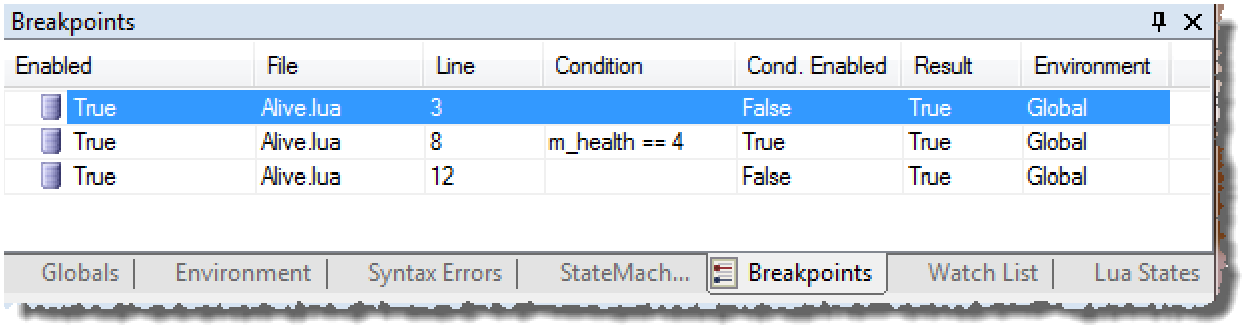
● Is true

○ Is false

Viewing and Managing Breakpoints

The **Breakpoints** window allows you to view all breakpoints and their attributes. To view this window, select **Window > Breakpoints**.

Figure 24 **Breakpoints Window**

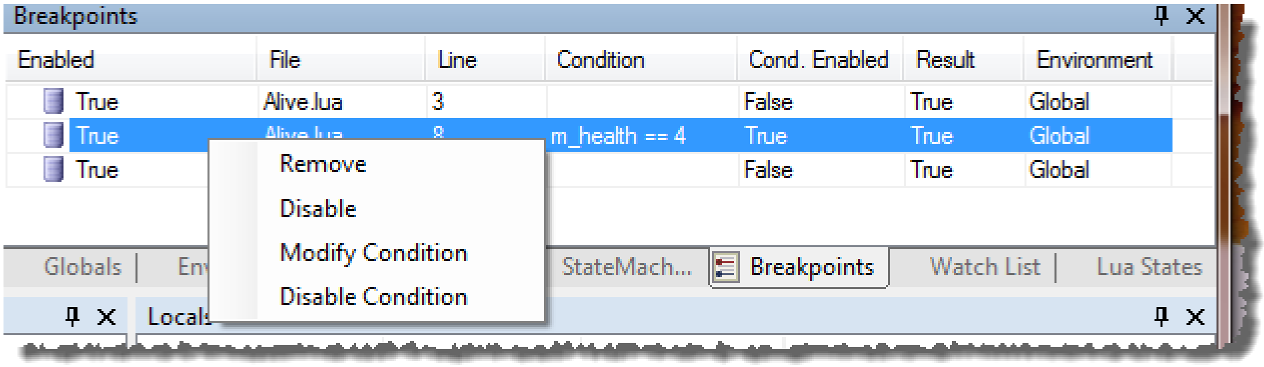


Breakpoints

Enabled	File	Line	Condition	Cond. Enabled	Result	Environment
True	Alive.lua	3		False	True	Global
True	Alive.lua	8	m_health == 4	True	True	Global
True	Alive.lua	12		False	True	Global

Right-click on any breakpoint to display a context menu that allows you to remove the breakpoint, disable it, modify the breakpoint condition, or enable the breakpoint condition, as shown in the following figure:

Figure 25 **Breakpoint Context Window**



Enabled	File	Line	Condition	Cond. Enabled	Result	Environment
True	Alive.lua	3		False	True	Global
True	Alive.lua	8	m_health == 4	True	True	Global
True				False	True	Global

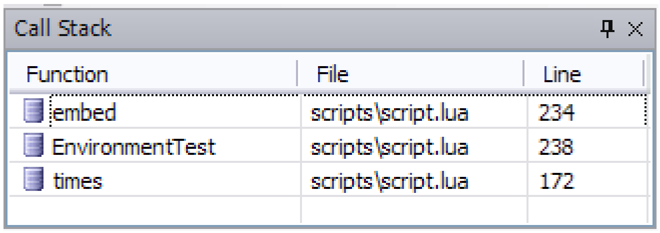
This context menu allows you to manage breakpoints across multiple files. You can also select a subset of breakpoints for removal.

You can use the standard Windows conventions to select multiple breakpoints for removal: **Shift**+click for contiguous breakpoints, and **Ctrl**+click for noncontiguous ones. Remove the selected breakpoints by right-clicking on any of them, and selecting **Remove All Selected** from the context menu.

Following Execution Flow

When execution pauses due to a breakpoint or **Stop** command, the **Call Stack** window tells you exactly where the pause occurred. This window also shows all nested function calls.

Figure 26 **Lua Call Stack Window**



Function	File	Line
embed	scripts\script.lua	234
EnvironmentTest	scripts\script.lua	238
times	scripts\script.lua	172

This call stack display shows the following nesting of function calls. Nesting sequence is shown in declining order of depth:

`times` (in the file `scripts\script.lua`) calls
    `EnvironmentTest` (in the file `scripts\script.lua`) calls
        `embed` (in the file `scripts\script.lua`), where execution has paused.

Monitoring Variables

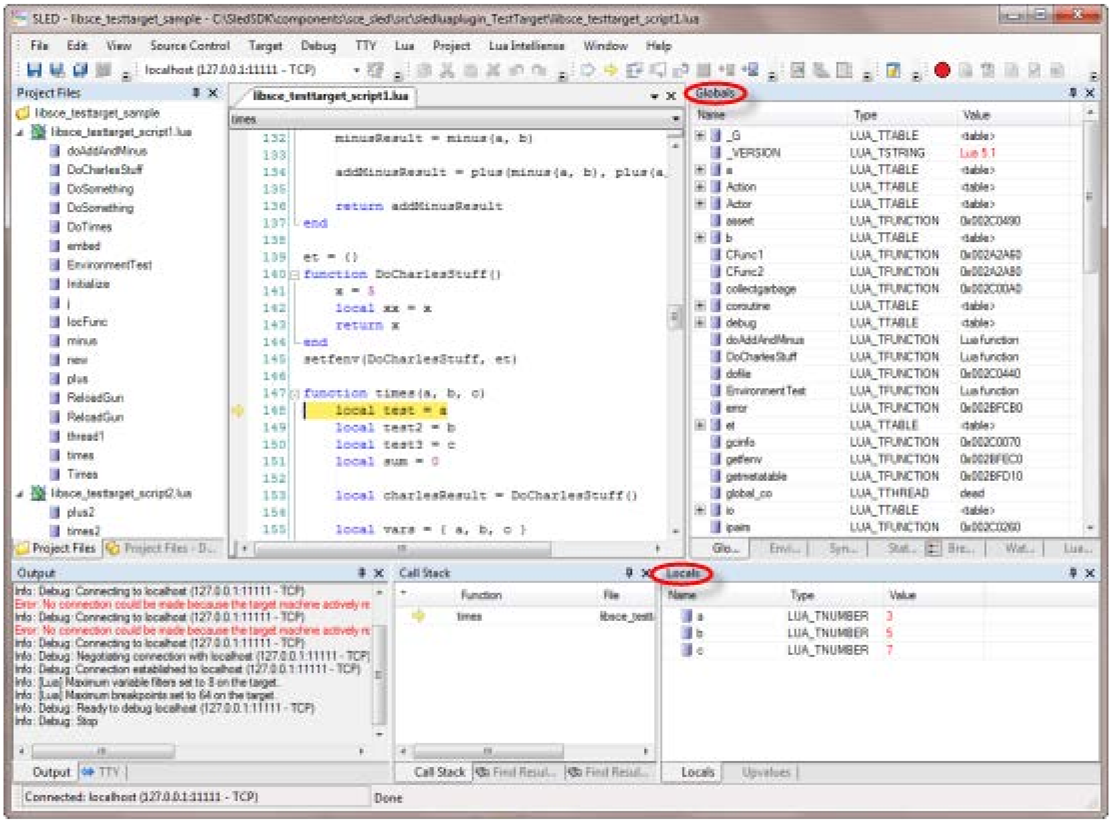
SLED provides the ability to monitor runtime values of local, upvalue, and global variables. It also allows you to set watches on variables of particular interest.

Note:
Script execution must be paused in order for SLED to provide runtime information about variables.

Viewing Variables

To view the values of variables, select **Window > Globals**, **Window > Locals**, **Window > Upvalues**, or click the **Globals**, **Locals**, or **Upvalues** tabs if these windows are already marked visible in the **Windows** menu. As the following figure demonstrates, you can display more than one of these windows at a time.

Figure 27 **Globals and Locals Windows, Both Open (Window Names Circled)**



```
    minusResult = minus(a, b)

    addMinusResult = plus(minus(a, b), plus(a_

    return addMinusResult
end

et = {}
function DoCharlesStuff()
    x = 5
    local xx = x
    return x
end
setfenv(DoCharlesStuff, et)

function times(a, b, c)
    local test = a
    local test2 = b
    local test3 = c
    local sum = 0

    local charlesResult = DoCharlesStuff()

    local vars = { a, b, c }
```

Name	Type	Value
_G	LUA_TTABLE	<table>
_VERSION	LUA_TSTRING	Lua 5.1
a	LUA_TTABLE	<table>
Action	LUA_TTABLE	<table>
Actor	LUA_TTABLE	<table>
assert	LUA_TFUNCTION	0x002C0490
b	LUA_TTABLE	<table>
CFunc1	LUA_TFUNCTION	0x002A2A60
CFunc2	LUA_TFUNCTION	0x002A2A80
collectgarbage	LUA_TFUNCTION	0x002C00A0
coroutine	LUA_TTABLE	<table>
debug	LUA_TTABLE	<table>
doAddAndMinus	LUA_TFUNCTION	Lua function
DoCharlesStuff	LUA_TFUNCTION	Lua function
dofile	LUA_TFUNCTION	0x002C0440
EnvironmentTest	LUA_TFUNCTION	Lua function
error	LUA_TFUNCTION	0x002BFCB0
et	LUA_TTABLE	<table>
gcinfo	LUA_TFUNCTION	0x002C0070
getfenv	LUA_TFUNCTION	0x002BFEC0
getmetatable	LUA_TFUNCTION	0x002BFD10
global_co	LUA_TTHREAD	dead
io	LUA_TTABLE	<table>
ipairs	LUA_TFUNCTION	0x002C0260

```
Info: Debug: Connecting to localhost (127.0.0.1:11111 - TCP)
Error: No connection could be made because the target machine actively re
Info: Debug: Connecting to localhost (127.0.0.1:11111 - TCP)
Error: No connection could be made because the target machine actively re
Info: Debug: Connecting to localhost (127.0.0.1:11111 - TCP)
Info: Debug: Negotiating connection with localhost (127.0.0.1:11111 - TCP)
Info: Debug: Connection established to localhost (127.0.0.1:11111 - TCP)
Info: [Lua] Maximum variable filters set to 8 on the target.
Info: [Lua] Maximum breakpoints set to 64 on the target.
Info: Debug: Ready to debug localhost (127.0.0.1:11111 - TCP)
Info: Debug: Stop
```

Function	File
times	libsce_testt...

Name	Type	Value
a	LUA_TNUMBER	3
b	LUA_TNUMBER	5
c	LUA_TNUMBER	7

Connected: localhost (127.0.0.1:11111 - TCP)
Done

These variable windows are in list views that allow you to sort the columns by clicking on the column label.

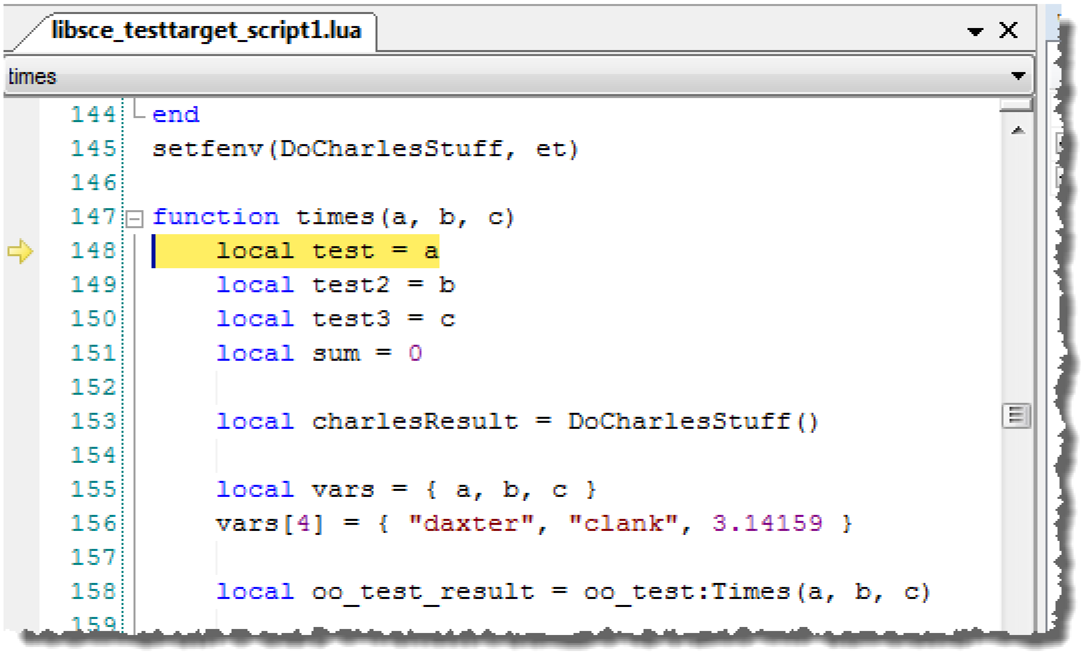
For example, you can use a variable tab to view *upvalues*. The **Upvalues** window displays the upvalues for the current context when the script is paused. In Lua, you can use a variable from some outer scope inside some inner scope, and it is called an upvalue in the inner scope. For example:

```
function Test1()
    local test1_value = 5

    local embedded_func = function()
        return 3.14159 * test1_value
    end

    print("embedded_func value" .. embedded_func())
end
```

In the function `embedded_func()`, `test1_value` from the function `Test1()` is referenced. From inside `embedded_func()`, `test1_value` is an upvalue, because it was defined in an outer scope.

Consider this example of an upvalue from the TestTarget project described in “Confirming Successful Installation and Configuration” in *Getting Started with SLED and Lua*. If script execution is stopped, the script window looks something like this:

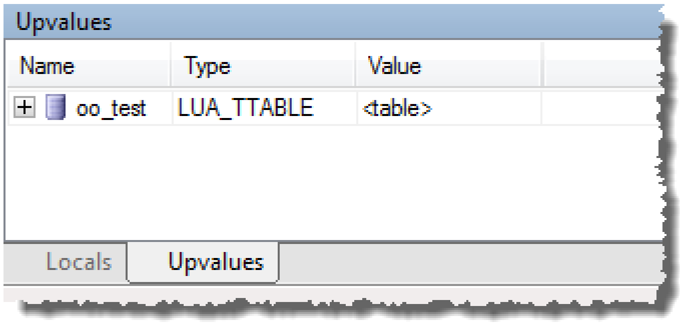
Figure 28 Script Window in TestTarget Project



```
end
setfenv(DoCharlesStuff, et)

function times(a, b, c)
    local test = a
    local test2 = b
    local test3 = c
    local sum = 0

    local charlesResult = DoCharlesStuff()

    local vars = { a, b, c }
    vars[4] = { "daxter", "clank", 3.14159 }

    local oo_test_result = oo_test:Times(a, b, c)
```

The **Upvalues** window displays this:

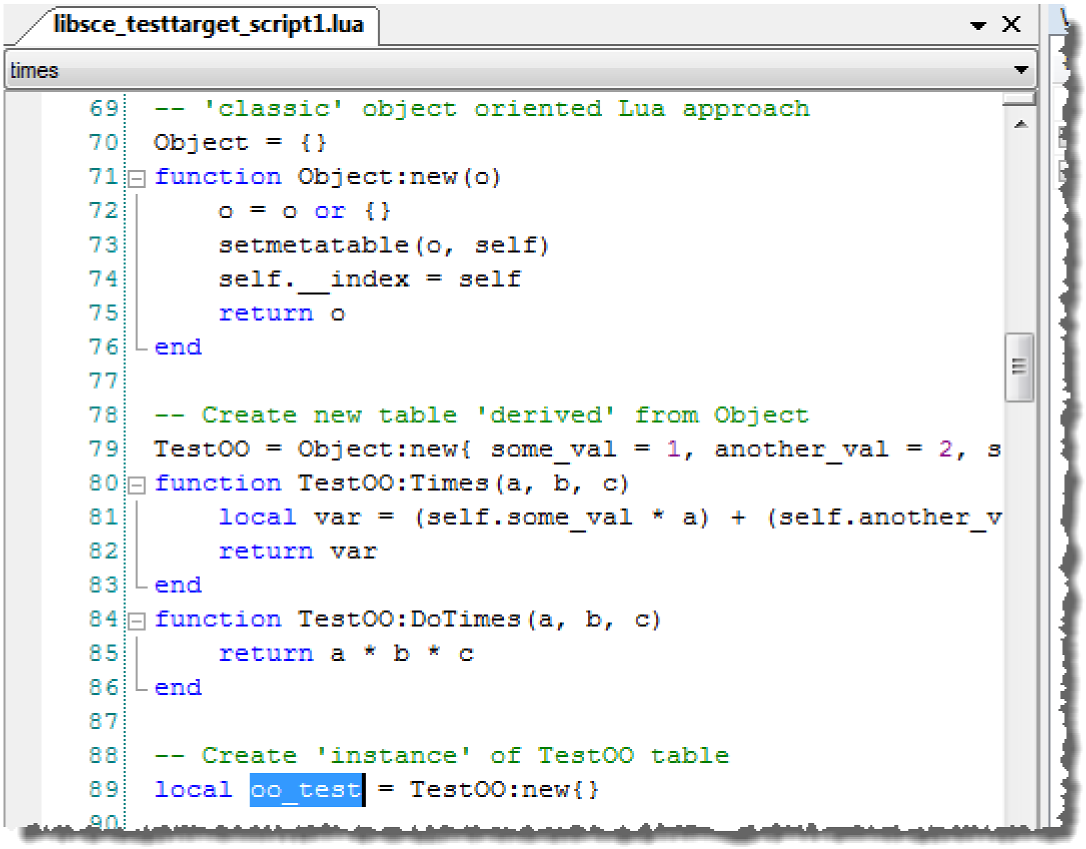
Figure 29 Upvalues Window in TestTarget



Name	Type	Value
⊞ oo_test	LUA_TTABLE	<table>

The variable `oo_test` is a local table in the script, an upvalue relative to the times function:

Figure 30 TestTarget Script Table Variable



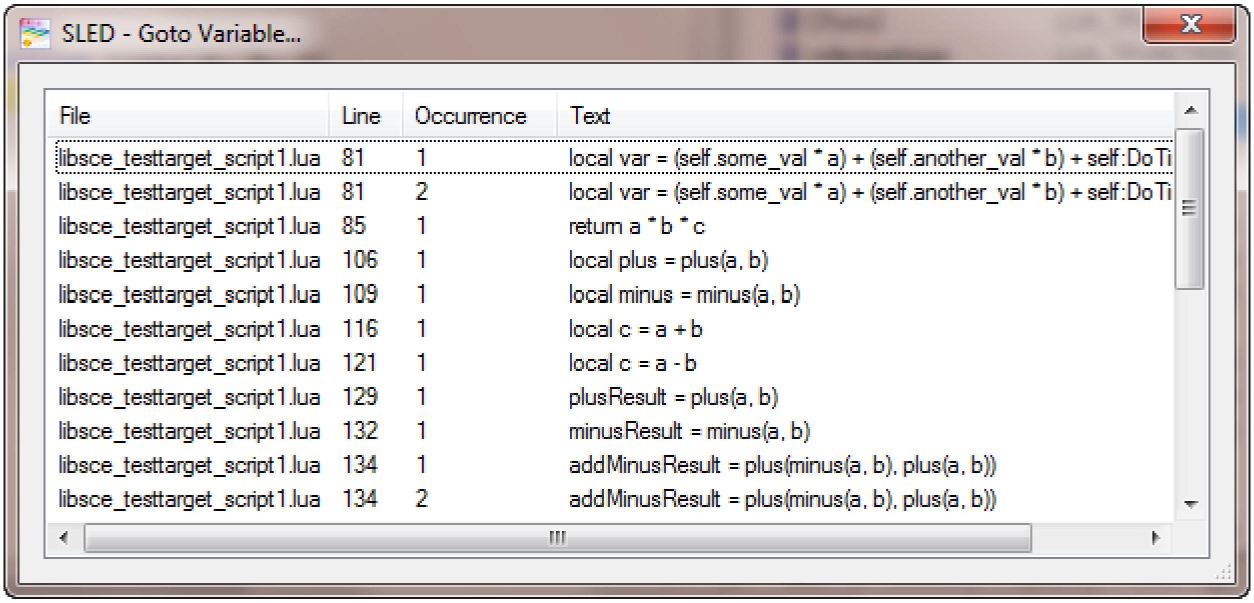
```
-- 'classic' object oriented Lua approach
Object = {}
function Object:new(o)
    o = o or {}
    setmetatable(o, self)
    self.__index = self
    return o
end

-- Create new table 'derived' from Object
TestOO = Object:new{ some_val = 1, another_val = 2, s
function TestOO:Times(a, b, c)
    local var = (self.some_val * a) + (self.another_v
    return var
end
function TestOO:DoTimes(a, b, c)
    return a * b * c
end

-- Create 'instance' of TestOO table
local oo_test = TestOO:new{}
```

Going to, Watching, and Copying Variables

You can do several things with variables of particular interest to you. Right-click on the variable name in the **Globals**, **Locals**, or **Upvalues** variable observation windows. The context menu that appears provides these options:

- **Goto**: Display a dialog listing all references to the variable. Click on an entry to go to that reference in the script file.

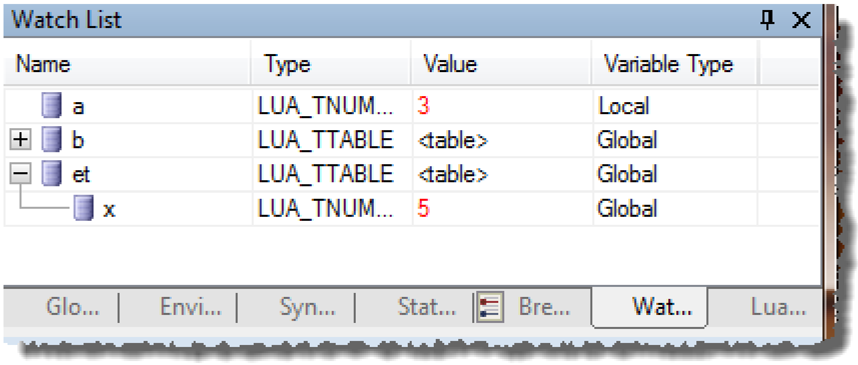
Figure 31 Goto Variable Dialog



File	Line	Occurrence	Text
libsce_testtarget_script1.lua	81	1	local var = (self.some_val * a) + (self.another_val * b) + self:DoTi
libsce_testtarget_script1.lua	81	2	local var = (self.some_val * a) + (self.another_val * b) + self:DoTi
libsce_testtarget_script1.lua	85	1	return a * b * c
libsce_testtarget_script1.lua	106	1	local plus = plus(a, b)
libsce_testtarget_script1.lua	109	1	local minus = minus(a, b)
libsce_testtarget_script1.lua	116	1	local c = a + b
libsce_testtarget_script1.lua	121	1	local c = a - b
libsce_testtarget_script1.lua	129	1	plusResult = plus(a, b)
libsce_testtarget_script1.lua	132	1	minusResult = minus(a, b)
libsce_testtarget_script1.lua	134	1	addMinusResult = plus(minus(a, b), plus(a, b))
libsce_testtarget_script1.lua	134	2	addMinusResult = plus(minus(a, b), plus(a, b))

- **Add Watch**: Add the variable to the **Watch List** window. It is then displayed in the **Watch List** window, showing the same information as in the variable window. You can right-click on variables in the **Watch List** window and the resulting context menu has the same options as in the variable window except for one: the **Add Watch** item is replaced by **Remove Watch** to remove the variable from the **Watch List** window. You can sort column data in the **Watch List** window by clicking on the column label. The **Watch List** persists between SLED sessions.

Figure 32 Watch Window



Watch List

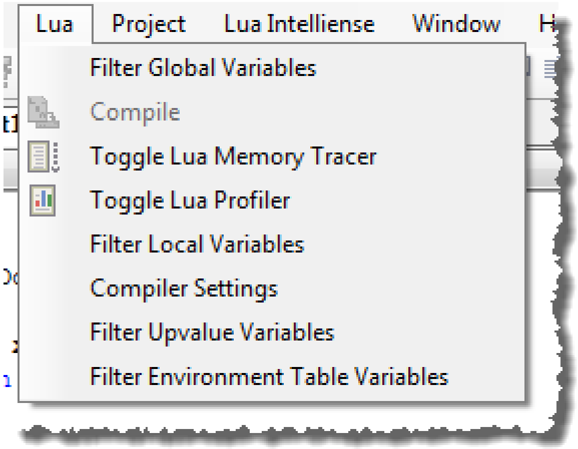
Name	Type	Value	Variable Type
a	LUA_TNUM...	3	Local
⊞ b	LUA_TTABLE	<table>	Global
⊟ et	LUA_TTABLE	<table>	Global
x	LUA_TNUM...	5	Global

- **Copy**: Copy the line of text from the window, copying this text, for example:
  - et.x LUA_TNUMBER 5 Global
- **Copy Name**: Copy the variable name, copying this text:
  - et.x
- **Copy Value**: Copy the variable value, copying this text:
  - 5

Filtering Variables

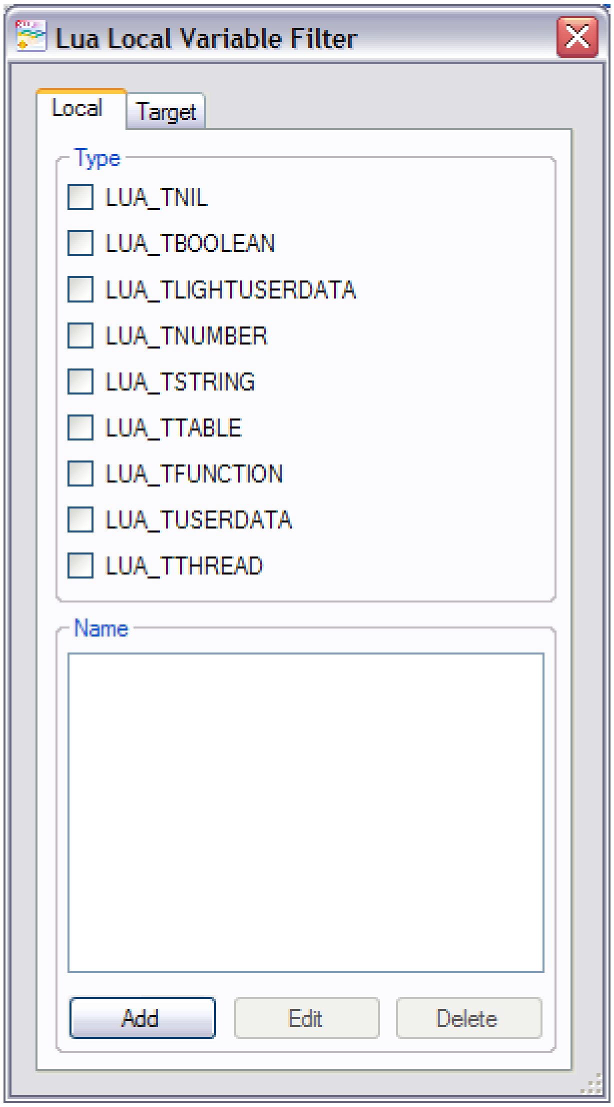
You can filter out variables of no interest to you based on type (Global, Local, Upvalue, or Environment Table) and their local and target side values. Filtering data on the target side costs slightly less network bandwidth than doing so locally. Click the **Lua** menu to select the type of filtering you want to do:

Figure 33 Lua Menu with Variable Filter Items



You can filter out variables according to type, name, or both. For example, when you select **Lua > Filter Local Variables**, a window similar to the following appears.

Figure 34 Lua Local Variable Filter Dialog

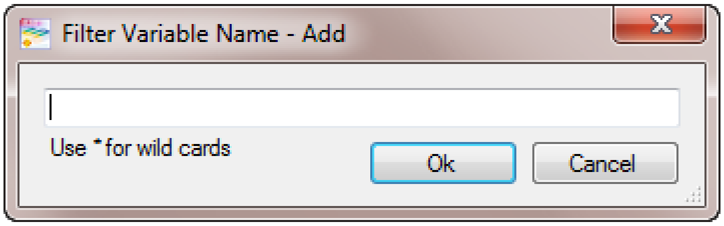


□ LUA_TNIL
□ LUA_TBOOLEAN
□ LUA_TLIGHTUSERDATA
□ LUA_TNUMBER
□ LUA_TSTRING
□ LUA_TTABLE
□ LUA_TFUNCTION
□ LUA_TUSERDATA
□ LUA_TTHREAD

To exclude a variable type, check the box next to the type.

In addition to excluding types, you can add specific variable names, or partial names, to exclude. To add a name, click the **Add** button. In the **Filter Variable Name – Add** dialog box that appears, enter the name of the variable that you wish to exclude.

Figure 35 **Filter Variable Name – Add Dialog**



You can use the wildcard symbol (*) in the name to exclude any variable that begins with a specific letter or letter combination. For example, to exclude everything that starts with B, specify “B*”. To specify everything that starts with “BA”, but no other “B”s, specify “BA*”. These names are case sensitive. Use the **Edit** and **Delete** buttons on the **Filter Variable Name – Add** dialog to change or remove the selected name filters.

When you are done excluding variable types and names, click the close button on the dialog’s top right to activate filtering.

Exclusion criteria are independent of one another. If you check **LUA_TNUMBER** and also exclude “BA*”, all variables of type **LUA_TNUMBER** are excluded regardless of name, and all variables whose names start with “BA” are excluded, regardless of type.

Filter settings do *not* persist across Lua sessions.

Modifying Variables and Scripts

When script execution is paused, SLED allows you to change variable values or even the script itself. You can see the effects of your changes immediately upon resuming execution. You do not need to restart the executable.

You can modify any value displayed in red in the variable observation windows.

Modifying a Variable

To modify a variable, double-click its existing value, and enter a new one in its place. Press **Enter** to save the change. When execution resumes, the script uses the new value.

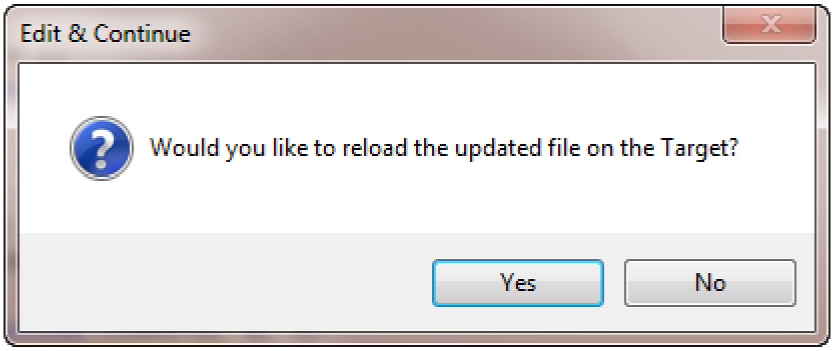
Note:
Values entered are superseded by any assignments specified in the script itself. For example, if you enter 10 for **anotherTemp**, and the script subsequently hits a line **anotherTemp = anotherTemp + 5**, the value is incremented by 5. A value change can only persist if it the change is written into the script. Use the edit-and-continue feature to assure persistence of changes.

To Modify a Script

The edit-and-continue feature allows you to make farther reaching, more persistent changes to the script that take effect right away, without having to restart the executable.

To perform edit-and-continue, make changes to the script when execution is paused, and then resume execution. When SLED asks if you wish to reload the updated file on the target machine, click **Yes**.

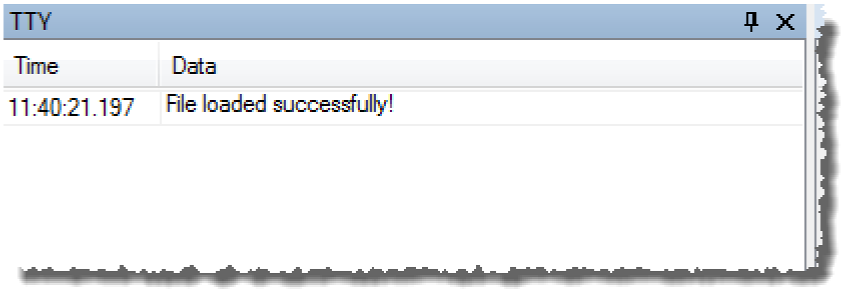
Figure 36 Edit and Continue Dialog



Would you like to reload the updated file on the Target?

SLED displays a message in the **TTY** tab that indicates whether or not the target machine successfully reloaded the script. The script would not load if there were syntax errors, for instance.

Figure 37 TTY File Loaded Successfully Message



Time	Data
11:40:21.197	File loaded successfully!

Note:
When you tell SLED to reload the updated file, the application automatically saves the changes that you made to the script. Make sure that you have a backup or some other way to revert the script to its previous state. You may also be able to undo the change if you are in the same debugging session.

5 Gauging Performance

SLED provides features that allow you to assess the performance of scripts. This chapter discusses:

- Lua Profiler
- Tracing Memory

Lua Profiler

Running in the background as a script executes, Lua Profiler records detailed information about the functions executed during runtime. This information simplifies the task of identifying problems with the flow of your script's execution.

Lua Profiler provides two types of function data: a tree view of the functions called during runtime, and quantitative data about those functions.

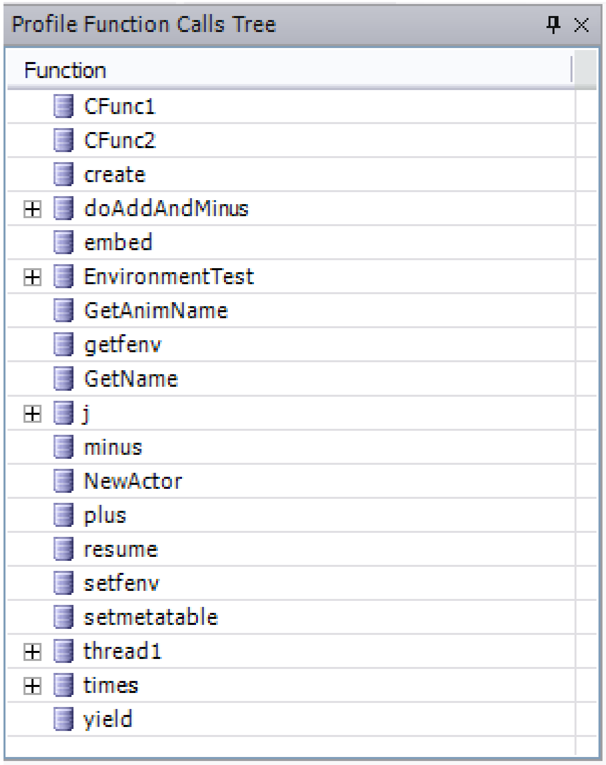
To start the Lua Profiler, select **Lua > Toggle Lua Profiler**. When you resume script execution, Lua Profiler begins recording data.

Note:
Lua Profiler causes a slight degradation of performance. Turn it on only when you wish to record performance data.

To View the Function Calls Tree

To view called functions, select **Window > Profile Function Calls Tree**. The **Profile Function Calls Tree** window appears, similar to the following figure.

Figure 38 Profile Function Calls Tree Window



Profile Function Calls Tree

Function

CFunc1
CFunc2
create
⊞ doAddAndMinus
embed
⊞ EnvironmentTest
GetAnimName
getfenv
GetName
⊞ j
minus
NewActor
plus
resume
setfenv
setmetatable
⊞ thread1
⊞ times
yield

This window shows all functions that were called while the script was executing. A plus sign indicates that a function has called other functions. Click the plus sign to reveal those other functions, listed under the calling function. Double-clicking any of the function names takes you to that function in a script editing window.

To clear data shown in this window, toggle the profiler off and then on again.

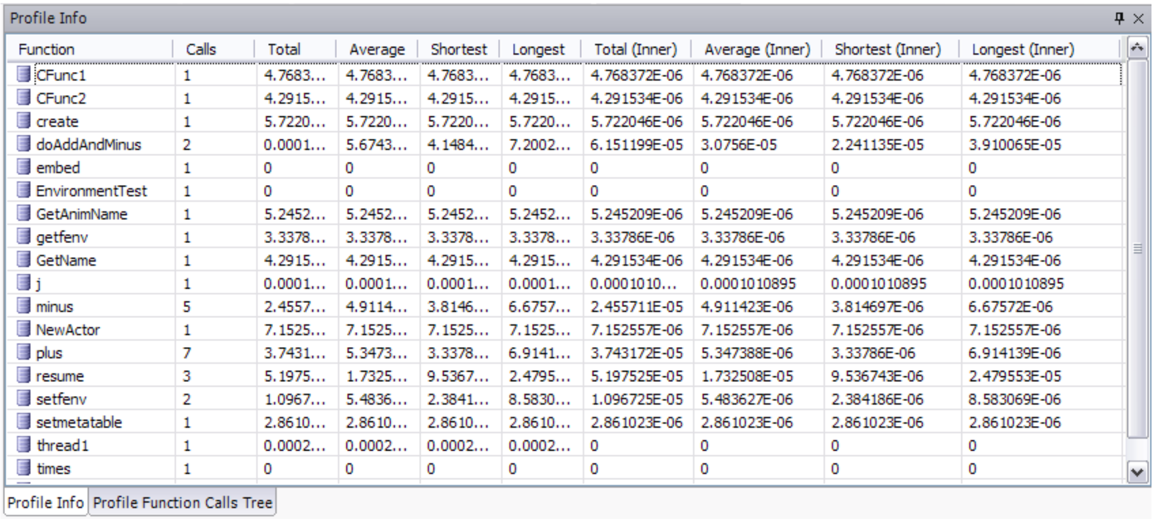
Note:
You must stop script execution to see the function calls list.
Called functions are shown in alphabetical—not chronological—order.

To View Quantitative Data

The **Profile Info** window shows, for each function, how many times it was called, along with a variety of time-related metrics.

To display this window, select **Window > Profile Info**. The **Profile Info** window appears, similar to the following figure.

Figure 39 The Profile Info Window



Function	Calls	Total	Average	Shortest	Longest	Total (Inner)	Average (Inner)	Shortest (Inner)	Longest (Inner)
CFunc1	1	4.7683...	4.7683...	4.7683...	4.7683...	4.768372E-06	4.768372E-06	4.768372E-06	4.768372E-06
CFunc2	1	4.2915...	4.2915...	4.2915...	4.2915...	4.291534E-06	4.291534E-06	4.291534E-06	4.291534E-06
create	1	5.7220...	5.7220...	5.7220...	5.7220...	5.722046E-06	5.722046E-06	5.722046E-06	5.722046E-06
doAddAndMinus	2	0.0001...	5.6743...	4.1484...	7.2002...	6.151199E-05	3.0756E-05	2.241135E-05	3.910065E-05
embed	1	0	0	0	0	0	0	0	0
EnvironmentTest	1	0	0	0	0	0	0	0	0
GetAnimName	1	5.2452...	5.2452...	5.2452...	5.2452...	5.245209E-06	5.245209E-06	5.245209E-06	5.245209E-06
getfenv	1	3.3378...	3.3378...	3.3378...	3.3378...	3.33786E-06	3.33786E-06	3.33786E-06	3.33786E-06
GetName	1	4.2915...	4.2915...	4.2915...	4.2915...	4.291534E-06	4.291534E-06	4.291534E-06	4.291534E-06
j	1	0.0001...	0.0001...	0.0001...	0.0001...	0.0001010...	0.0001010895	0.0001010895	0.0001010895
minus	5	2.4557...	4.9114...	3.8146...	6.6757...	2.455711E-05	4.911423E-06	3.814697E-06	6.67572E-06
NewActor	1	7.1525...	7.1525...	7.1525...	7.1525...	7.152557E-06	7.152557E-06	7.152557E-06	7.152557E-06
plus	7	3.7431...	5.3473...	3.3378...	6.9141...	3.743172E-05	5.347388E-06	3.33786E-06	6.914139E-06
resume	3	5.1975...	1.7325...	9.5367...	2.4795...	5.197525E-05	1.732508E-05	9.536743E-06	2.479553E-05
setfenv	2	1.0967...	5.4836...	2.3841...	8.5830...	1.096725E-05	5.483627E-06	2.384186E-06	8.583069E-06
setmetatable	1	2.8610...	2.8610...	2.8610...	2.8610...	2.861023E-06	2.861023E-06	2.861023E-06	2.861023E-06
thread1	1	0.0002...	0.0002...	0.0002...	0.0002...	0	0	0	0
times	1	0	0	0	0	0	0	0	0

Note:
You must stop script execution to see profile data.

The following table lists and describes each column. All times are in seconds.

Table 2 Lua Profile Info Column Names and Meanings

Column name	Meaning
Function	Function name
Calls	Number of times the function was called
Total	Total time spent in the function
Average	Average time spent in the function
Shortest	Shortest time spent in the function
Longest	Longest time spent in the function
Total (Inner)	Total time spent in the function, excluding time spent in those nested within it
Average (Inner)	Average time spent in the function, excluding time spent in those nested within it
Shortest (Inner)	Shortest time spent in the function, excluding time spent in those nested within it
Longest (Inner)	Longest time spent in the function, excluding time spent in those nested within it

Double-clicking any of the function names takes you to that function in a script-editing window.

To clear the data shown in this window, toggle the profiler off and then on again.

Tracing Memory

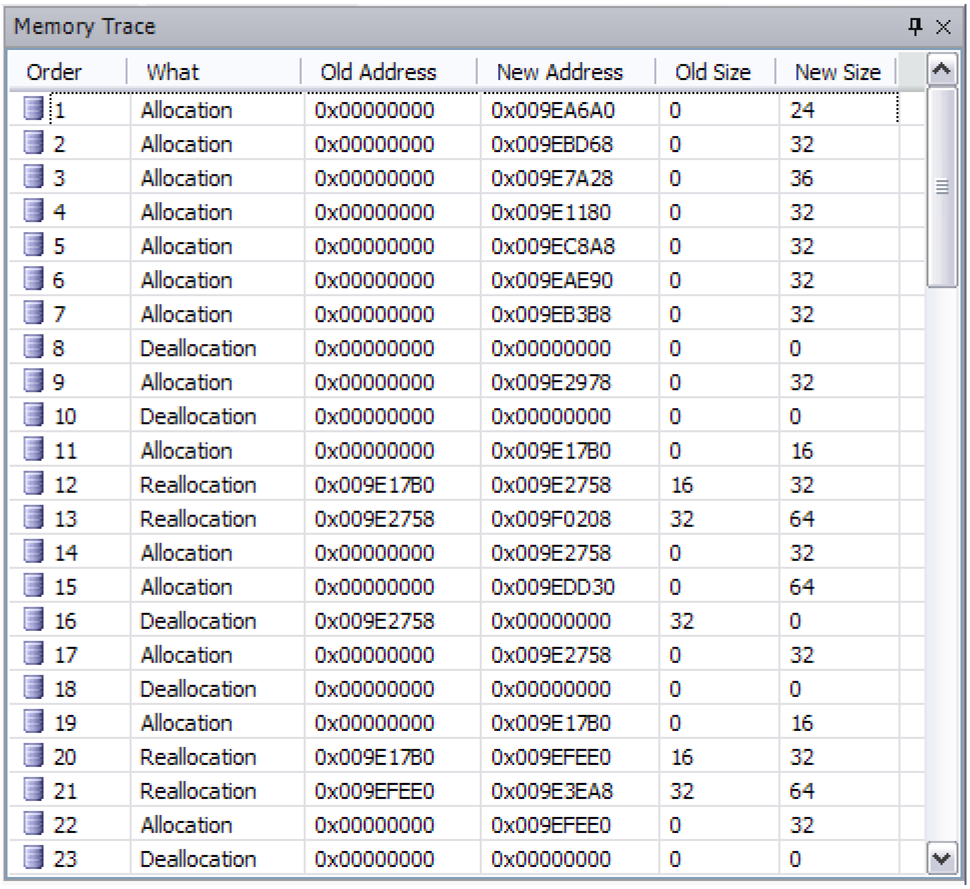
Running in the background as a script executes, Lua Memory Tracer records detailed information about sizes and locations of memory allocations and deallocations. This information simplifies memory management and troubleshooting connected with script execution.

To start Lua Memory Tracer, select **Lua > Toggle Lua Memory Tracer**. When you resume script execution, Lua Memory Tracer begins recording data.

Note:
You must stop script execution to see memory data.
Lua Memory Tracer causes a slight degradation of performance. Turn it on only when you wish to record memory data.

To view the recorded memory data, select **Window > Memory Trace**. The **Memory Trace** window appears, similar to the following figure.

Figure 40 Memory Trace Window



Memory Trace

Order	What	Old Address	New Address	Old Size	New Size
1	Allocation	0x00000000	0x009EA6A0	0	24
2	Allocation	0x00000000	0x009EBD68	0	32
3	Allocation	0x00000000	0x009E7A28	0	36
4	Allocation	0x00000000	0x009E1180	0	32
5	Allocation	0x00000000	0x009EC8A8	0	32
6	Allocation	0x00000000	0x009EAE90	0	32
7	Allocation	0x00000000	0x009EB3B8	0	32
8	Deallocation	0x00000000	0x00000000	0	0
9	Allocation	0x00000000	0x009E2978	0	32
10	Deallocation	0x00000000	0x00000000	0	0
11	Allocation	0x00000000	0x009E17B0	0	16
12	Reallocation	0x009E17B0	0x009E2758	16	32
13	Reallocation	0x009E2758	0x009F0208	32	64
14	Allocation	0x00000000	0x009E2758	0	32
15	Allocation	0x00000000	0x009EDD30	0	64
16	Deallocation	0x009E2758	0x00000000	32	0
17	Allocation	0x00000000	0x009E2758	0	32
18	Deallocation	0x00000000	0x00000000	0	0
19	Allocation	0x00000000	0x009E17B0	0	16
20	Reallocation	0x009E17B0	0x009EFEE0	16	32
21	Reallocation	0x009EFEE0	0x009E3EA8	32	64
22	Allocation	0x00000000	0x009EFEE0	0	32
23	Deallocation	0x00000000	0x00000000	0	0

The following table lists and describes each column. All sizes are in bytes. The columns are not sortable.

Table 3 Lua Profile Info Column Names and Meanings

Column name	Meaning
Order	Sequence of the memory operation, to indicate order.
What	Kind of memory operation: Allocation, Deallocation, or Reallocation.
Old Address	Previous address. For Allocation, always 0x00000000. For Deallocation, the address being deallocated. For Reallocation, the address that was deallocated before reallocation.
New Address	New address. For Allocation, the address of memory allocated. For Deallocation, always 0x00000000. For Reallocation, the address memory relocated to.

Column name	Meaning
Old Size	Previous memory size. For Allocation, always 0. For Deallocation, the size being deallocated. For Reallocation, the size that was deallocated before reallocation.
New Size	New memory size. For Allocation, the size of memory. For Deallocation, always 0. For Reallocation, the new size of memory allocated, typically larger than **Old Size**.

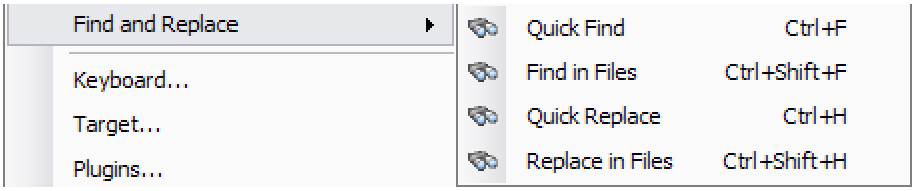
6 Find and Replace

SLED provides find and replace functionality similar to that provided in programs like Microsoft Visual Studio. You can search for any string or use wildcards to match partial strings in the current file, open files, files in the project, or you can specify the files to search. You can access these functions in four ways:

- Quick Find
- Quick Replace
- Find in Files
- Replace in Files

To access Find and Replace, select **Edit > Find and Replace**. The following menu items appear:

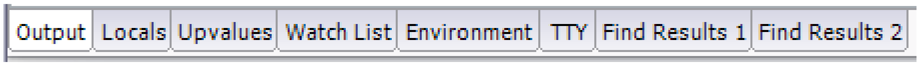
Figure 41 Find and Replace Menu Operations



Select one of the following types of Find and Replace operations:

- **Quick Find**: match a string or expression in files open for editing or in the current project.
- **Find in Files**: match a string or expression in any accessible files, or in the entire solution. You can specify which **Find Results** window in which to place results. **Find Results 1** is the default results window.
- **Quick Replace**: replace a string or expression in files open for editing or in the current project.
- **Replace in Files**: replace a string or expression in any accessible files, or in the entire solution. Results appear in one of the two Find results windows, accessible using the tabs at the bottom of the SLED window. You can specify the **Find Results** window in which to place results. **Find Results 1** is the default results window. The following figure shows the **Find Results** tabs.

Figure 42 Find Results Tabs

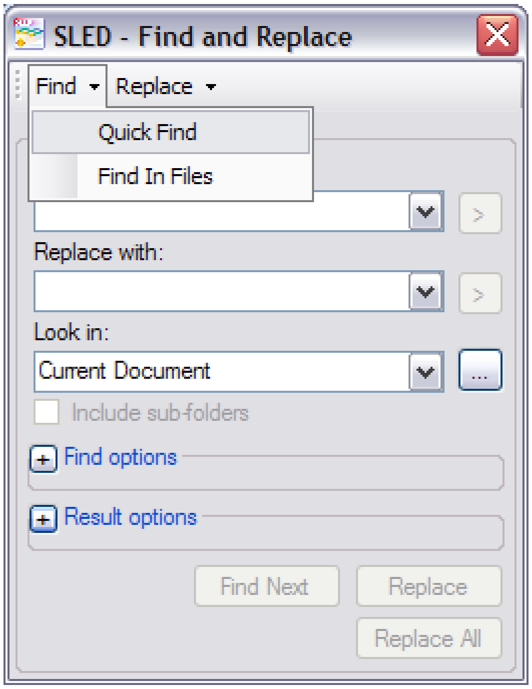


When you specify where to search for the Find in Files and Replace in Files operations, you can optionally add directories to search using the ellipsis button .... When you click ..., an add/remove window appears. Click **Add** to browse for and add directories to search, or click **Remove** to remove directories.

Use the **F3** function key to repeat the last find or replace. Use **Ctrl+F3** to search for the next instance of the currently-selected text.

All of the Find and Replace operations are accessible from any of the **Find and Replace** windows, because all the **Find** menu items display a dialog that can be transformed into all the other dialog forms. For example, you can access Find in Files, Quick Replace, and Replace in Files from the Quick Find window, and you can access Quick Find from any of the other operations' windows, using the **Find and Replace** drop-down menus. The following figure shows how to access **Quick Find** from the **Replace in Files** window.

Figure 43 **Accessing Quick Find from the Replace in Files Dialog**

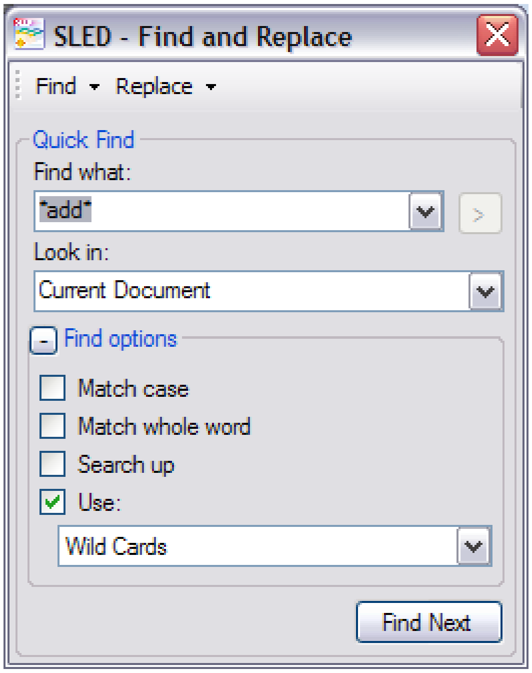


Look in: Current Document

Quick Find

The following figure shows the **Quick Find** window.

Figure 44 Quick Find Dialog



Find what: *add*

Look in: Current Document

□ Match case

□ Match whole word

□ Search up

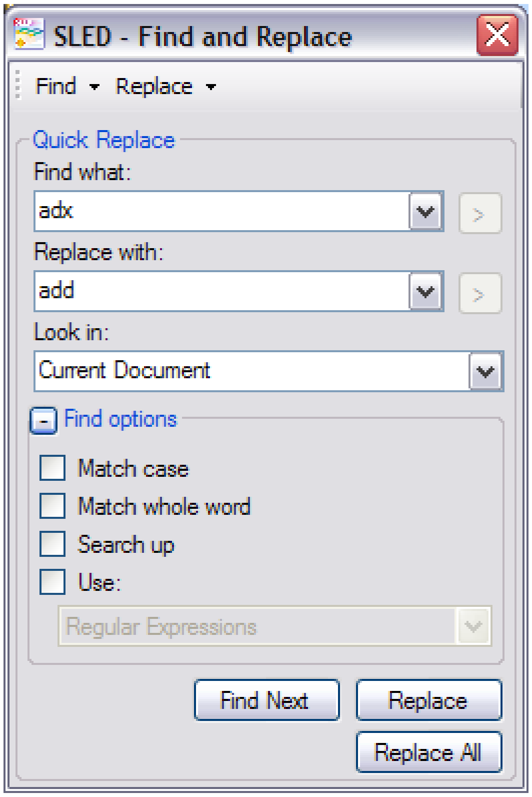
☑ Use: Wild Cards

- **Find what**: Specify the character or string to search for. To use wildcards in your search, check the **Use:** checkbox and select **Wild Cards** from the drop-down menu. To use regular expressions in your search, check the **Use:** checkbox and select **Regular Expressions** from the drop-down menu.
- **Look in**: Use the drop-down menu to specify where to search for the character or string: **Current Document**, **All Open Documents**, **Current Project**, or **Entire Solution**.
- **Find options**: Check the boxes to specify that you want to match case exactly, match a whole word exactly, or search backwards in the document. Use the drop-down **Use:** menu to search using wild cards or to search using regular expressions.
- **Find Next** button: Click to find the next occurrence of the character or string. If you select **Search up**, SLED searches for the previous instance of the character or string.

Quick Replace

The following figure shows the **Quick Replace** window.

Figure 45 Quick Replace Dialog



Find what: adx

Replace with: add

Look in: Current Document

□ Match case

□ Match whole word

□ Search up

□ Use:

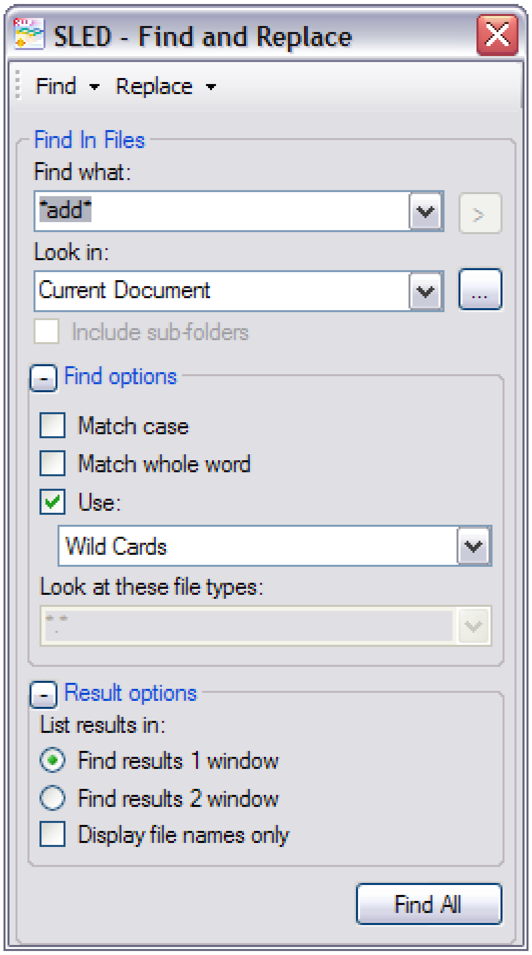
Regular Expressions

- **Find what**: Specify the character or string to search for. To use wildcards in your search, check the **Use:** checkbox and select **Wild Cards** from the drop-down menu. To use regular expressions in your search, check the **Use:** checkbox and select **Regular Expressions** from the drop-down menu.
- **Replace with**: Specify the replacement character or string.
- **Look in**: Use the drop-down menu to specify where to search for the character or string: **Current Document**, **All Open Documents**, **Current Project**, or **Entire Solution**.
- **Find options**: Check the boxes to specify that you want to match case exactly, match a whole word exactly, or search backwards in the document. Use the drop-down **Use:** menu to search using wild cards or to search using regular expressions.
- Click the **Find Next** button to find the next occurrence of the character or string. Click **Replace** to replace the character or string specified in the **Find what:** field with the current occurrence of the character or string specified in the **Replace with:** field, or click the **Replace All** button to replace all occurrences in the document or set of documents specified in the **Look in:** field.

Find in Files

The following figure shows the **Find in Files** window.

Figure 46 Find In Files Dialog



Find what: *add*

Look in: Current Document

□ Include sub-folders

☐ Match case

☐ Match whole word

☑ Use:

Wild Cards

Look at these file types:

.

List results in:

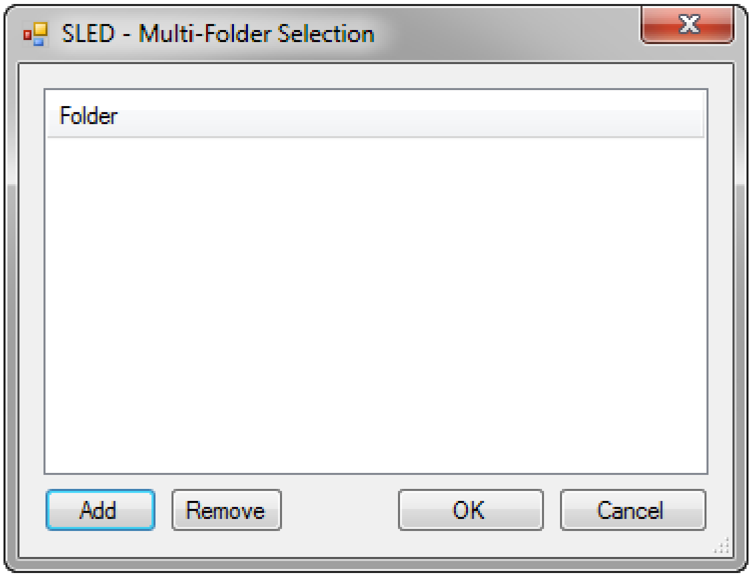
● Find results 1 window

○ Find results 2 window

☐ Display file names only

- **Find what**: Specify the character or string to search for. To use wildcards in your search, check the **Use:** checkbox and select **Wild Cards** from the drop-down menu. To use regular expressions in your search, check the **Use:** checkbox and select **Regular Expressions** from the drop-down menu.
- **Look in**: Use the drop-down menu to specify where to search for the character or string: **Current Document**, **All Open Documents**, **Current Project**, **Entire Solution**, or **Custom**. Click **Custom** or the ... button to add or remove directories to search using the **Multi-Folder Selection** dialog, shown in the illustration.

Figure 47 Multi-Folder Selection Dialog

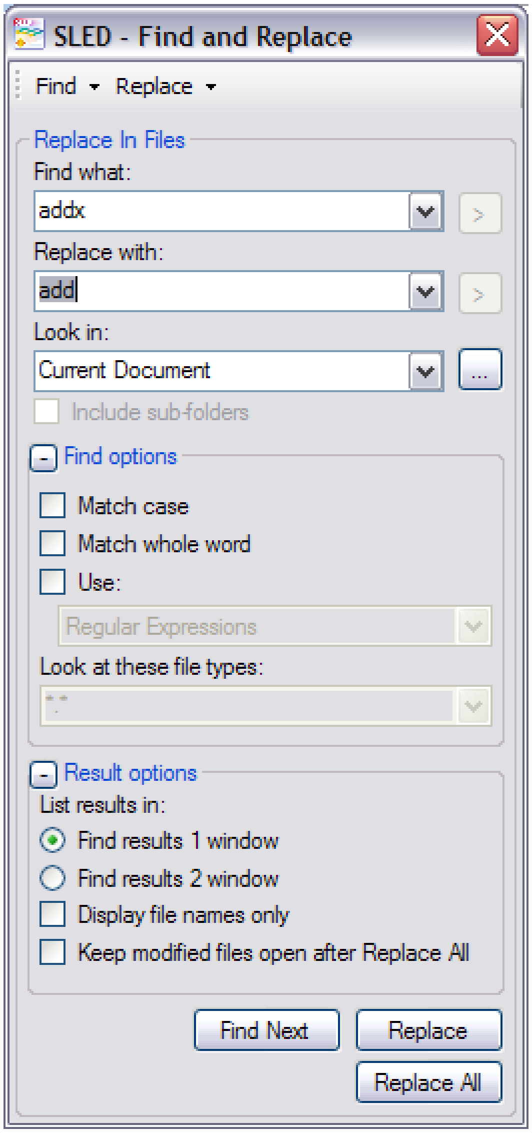


- **Find options**: Click the checkboxes to specify that you want to match case exactly or match a whole word exactly. Use the drop-down **Use:** menu to search using wild cards or to search using regular expressions.
- **Look at these file types**: If you select **Custom** in the **Look in:** field, or you click the ... button, this field becomes active. Specify the types of files to search. The default is all files, "*.*", and several other options are provided in this drop-down list.
- **Result options**: Check the **Find results 1 window** or **Find results 2 window** boxes to specify the results window in which to place search results. Note the **Find Results 1** and **Find Results 2** tabs along the bottom row of tabs in the SLED window. If you want to see only the list of file names where the string was found, check the **Display file names only** box.
- **Find All** button: Click to find all occurrences of the character or string in the document set specified in the **Look in:** field.

Replace in Files

The following figure shows the **Replace in Files** window.

Figure 48 Replace In Files Dialog



Find what: addx

Replace with: add

Look in: Current Document

□ Include sub-folders

□ Match case

□ Match whole word

□ Use: Regular Expressions

Look at these file types: *.*

List results in:

● Find results 1 window

○ Find results 2 window

□ Display file names only

□ Keep modified files open after Replace All

- **Find what**: Specify the character or string to search for. To use wildcards in your search, check the **Use:** checkbox and select **Wild Cards** from the drop-down menu. To use regular expressions in your search, check the **Use:** checkbox and select **Regular Expressions** from the drop-down menu.
- **Replace with**: Specify the replacement character or string.
- **Look in**: Use the drop-down menu to specify where to search for the character or string: **Current Document**, **All Open Documents**, **Current Project**, **Entire Solution**, or **Custom**. Click **Custom** or the ... button to add or remove directories to search using the **Multi-Folder Selection** dialog.

- **Find options**: Check the boxes to specify that you want to match case exactly or match a whole word exactly. Use the drop-down **Use:** menu to search using wild cards or to search using regular expressions.
- **Look at these file types**: if you select **Custom** in the **Look in:** field, or you click the ... button, this field becomes active. Specify the types of files to search. The default is all files, "*.*", and several other options are provided in this drop-down list.
- **Result options**: Check the **Find results 1 window** or **Find results 2 window** boxes to specify the results window in which to place search results. Note the **Find Results 1** and **Find Results 2** tabs along the bottom row of tabs in the SLED window. If you want to see to see only the list of file names where the string was replaced, check the **Display file names only** box. Check **Keep modified files open after Replace All** to enable easy double-checks of replacements after a **Replace All** operation.
- Click the **Find Next** button to find the next occurrence of the character or string. Click **Replace** to replace the character or string specified in the **Find what:** field with the current occurrence of the character or string specified in the **Replace with:** field, or click the **Replace All** button to replace all occurrences in the document or set of documents specified in the **Look in:** field.