Getting Started with Netscape JavaScript Debugger

Document last modified: 12/12/97.

This document gives a brief introduction to Netscape JavaScript Debugger 1.1. JavaScript is a cross-platform, object-based scripting language. This version of Netscape JavaScript Debugger allows you to debug client-side JavaScript code. The debugger cannot be used to debug server-side JavaScript, Java, or HTML.

Netscape JavaScript Debugger requires Netscape Communicator 4.02 or later. The debugger cannot be used with other browsers.

Note When you're stopped in Netscape JavaScript Debugger, you cannot use many features of Communicator. In particular, you cannot use Navigator to browse web pages. For this reason, it is recommended that you download the PDF version of this document before starting the debugger. You can then either print that document or view it online with Acrobat Reader.

What's in this document:

Starting the Debugger

The end of the installation process automatically runs the debugger. After that first time, you run the debugger by opening the appropriate page in Navigator. On Windows 95/NT, that page is:

```
[your Communicator install directory]\Program\JSDebug\JSDebugger.html
```

Here, *`[your Communicator install directory]`* is the directory in which Communicator is installed. For example, if you installed it in `c:\Program Files\Netscape\Communicator\`, then the start page for Netscape JavaScript Debugger would be:

```
c:\Program Files\Netscape\Communicator\Program\JSDebug\JSDebugger.html
```

On other platforms, that page is:

```
[your Communicator install directory]\JSDebug\JSDebugger.html
```

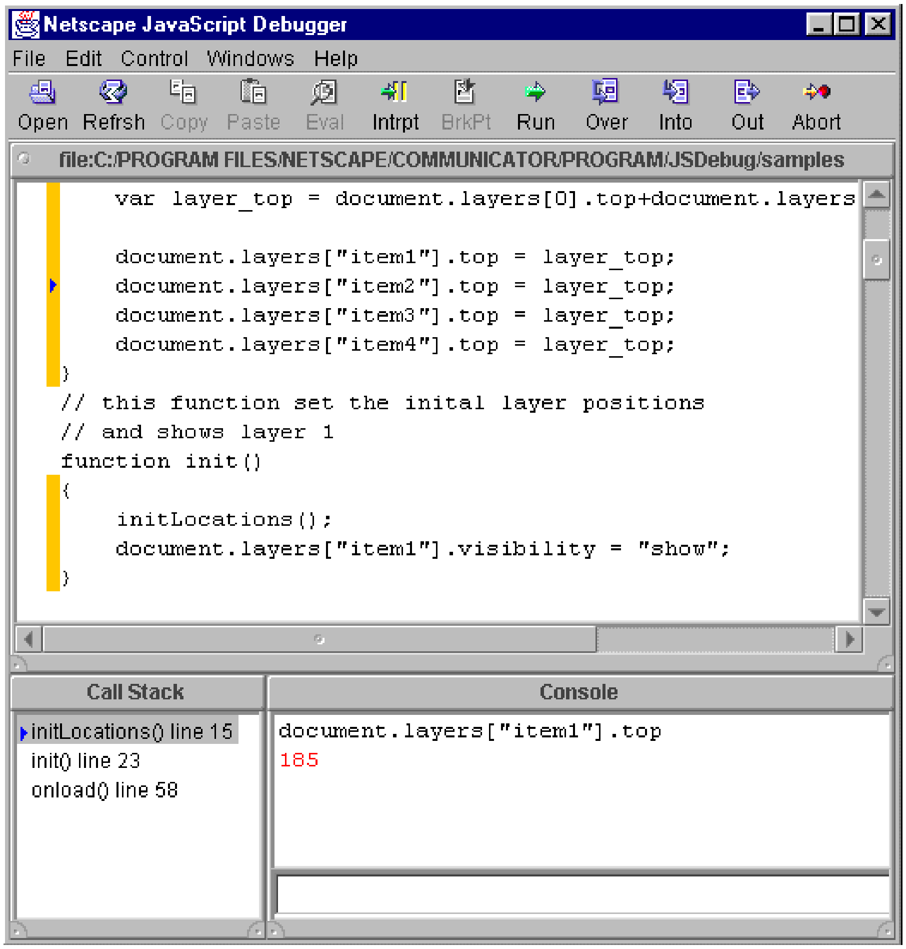
For future convenience, the first time you use the debugger, you should add the `JSDebugger.html` page to your bookmarks or to your personal toolbar in Navigator.

Note Several security dialog boxes are displayed the first time you run the debugger. You must grant the security privileges the debugger requests in order for it to function correctly.

Understanding the Windows

The primary configuration of the JavaScript debugger windows is shown in Figure 1.

Figure 1 JavaScript debugger configuration


Netscape JavaScript Debugger
File Edit Control Windows Help
Open Refrsh Copy Paste Eval Intrpt BrkPt Run Over Into Out Abort
file:C:/PROGRAM FILES/NETSCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples
Call Stack
Console

```
    var layer_top = document.layers[0].top+document.layers

    document.layers["item1"].top = layer_top;
    document.layers["item2"].top = layer_top;
    document.layers["item3"].top = layer_top;
    document.layers["item4"].top = layer_top;
}
// this function set the inital layer positions
// and shows layer 1
function init()
{
    initLocations();
    document.layers["item1"].visibility = "show";
}
```

```
initLocations() line 15
init() line 23
onload() line 58
```

```
document.layers["item1"].top
185
```

The debugger window has a menu bar, a toolbar and (once you've opened an HTML page) three panes. A Source View window is at the top; the Call Stack is below the Source View and on the left; and the Console is below the Source View and on the right.

While stopped in the debugger, you can use commands from the Windows menu open the Object Inspector, Breakpoint, and Watches windows.

Where To Find Commands

There are three common places to access the debugger's commands: the menu bar, the toolbar, and the right-click menu in a Source View window.

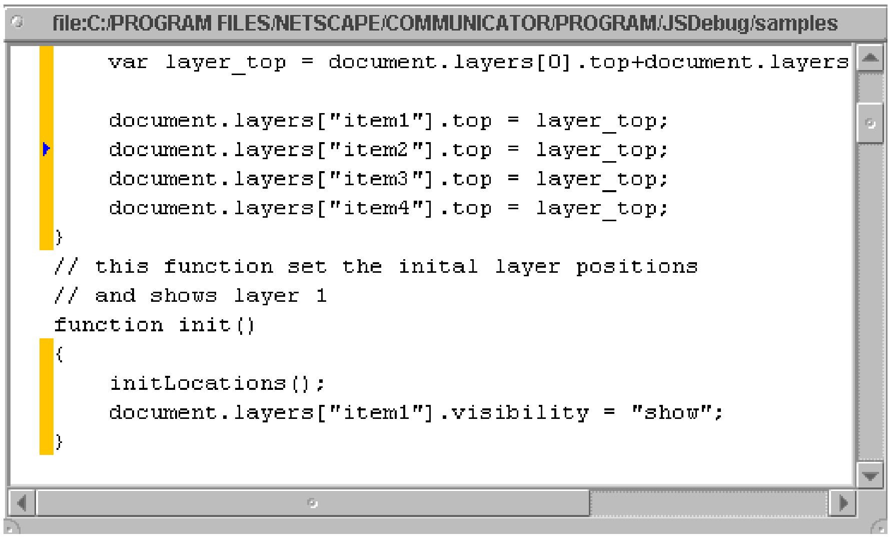
Most of the items that occur on menus also occur on the toolbar. In addition, if you right-click in a Source View window over selected text, a menu of common commands appears. These commands are also available on the menus and some are on the toolbar.

Where applicable, the instructions in this document tell you how to use the toolbar and the individual panes to access the debugger's functionality. You can also choose menu items (either from the menu bar or from the right-click menu) for these tasks. When there is no button for a task, the menu item on the menu bar is described.

Source View Windows

The JavaScript debugger creates a separate Source View window, as shown in Figure 2, for each HTML page you open in the debugger. (See "Opening a page to debug" on page 13 for information on how you can open a page.)

Figure 2 Source View window



file:C:/PROGRAM FILES/NETSCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples

```
    var layer_top = document.layers[0].top+document.layers

    document.layers["item1"].top = layer_top;
    document.layers["item2"].top = layer_top;
    document.layers["item3"].top = layer_top;
    document.layers["item4"].top = layer_top;
}
// this function set the inital layer positions
// and shows layer 1
function init()
{
    initLocations();
    document.layers["item1"].visibility = "show";
}
```

Each Source View window contains the source of an HTML page. When execution stops in the debugger, the Source View window containing the currently executing HTML page is brought to the surface.

Source View windows use colors and icons in the left margin to indicate the following information:

- A dark arrow indicates a currently executing line
- Orange bars to the left indicate function bodies
- Yellow bars to the left indicate top-level scripts
- Red dots indicates lines for which unconditional breakpoints have been set
- Orange dots indicate lines for which conditional breakpoints have been set

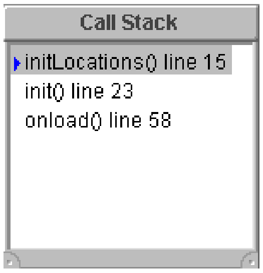
When execution is stopped, you can perform these actions in Source View windows:

- Select text in a line in the standard way
- Highlight a word by double-clicking on the text
- Evaluate an expression. On Unix and Windows platforms, you do so by right-clicking on the selection and choosing from the menu. On Macintosh PowerPC, you do so by command-clicking on the selection and choosing from the menu.
- Open the Object Inspector. This action is on the right-click or command menu.
- Add or remove a breakpoint from the current line. This action is on the right-click or command menu.
- Add the current selection to the Watch window. This action is on the right-click or command menu.
- Copy the current selection to the clipboard. This action is on the right-click or command menu.
- To refresh the Source View windows, from the File menu, choose Refresh All. This also refreshes the Page List displayed when you select Open. (The system automatically refreshes this window; you rarely need to refresh manually.)

Call Stack

The Call Stack, shown in Figure 3, displays the current execution location.

Figure 3 Call Stack window


Call Stack

initLocations() line 15
init() line 23
onload() line 58

If this window is not empty, you are currently stopped in the debugger.

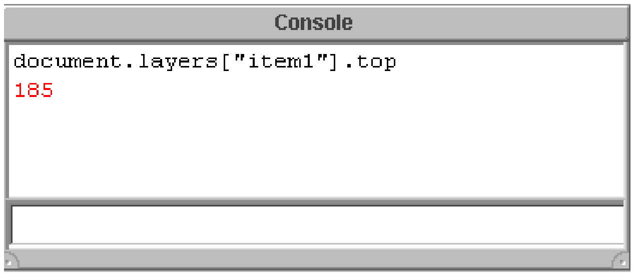
- The most recent stack frame is displayed at the top of the Call Stack. Its caller appears below it, and so on.
- A dark arrow in the margin points to the currently active stack frame.
- Select a line in the Call Stack to surface the Source View window for that source and to set the active frame to that line.

Important If the Call Stack is not empty, that is, if you are stopped in the debugger, you cannot use many features of Communicator. For example, while stopped in the debugger you cannot open a page in Navigator.

Console

The console, shown in Figure 4, is active only while JavaScript execution is stopped.

Figure 4 Console window



```
document.layers["item1"].top
185
```

The console is divided into two parts:

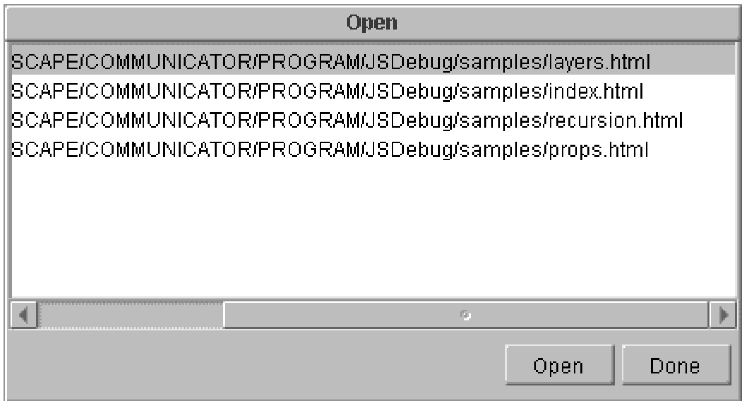
- The top part shows information displayed by the watch mechanism and the result of expressions you evaluate while stopped in the debugger.
- The bottom part is an input window in which you can type JavaScript expressions. These expressions are evaluated in the context of the currently active frame.

When execution is stopped, you can have the debugger evaluate the current selection in a Source View window. To do so, on Windows and Unix you right-click on the selection. On Macintosh PowerPC, you command-click on it. On any system, a menu appears from which you can choose Evaluate. The result is displayed in the Console window. You can also click the Eval button on the toolbar to evaluate the current selection.

Page List

The Page List, shown in Figure 5, appears when you click the Open button. It contains a list of all pages you have opened in Navigator since starting the debugger. The pages are sorted with the most recently opened listed first.

Figure 5 Page List



```
SCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples/layers.html
SCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples/index.html
SCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples/recursion.html
SCAPE/COMMUNICATOR/PROGRAM/JSDebug/samples/props.html
```

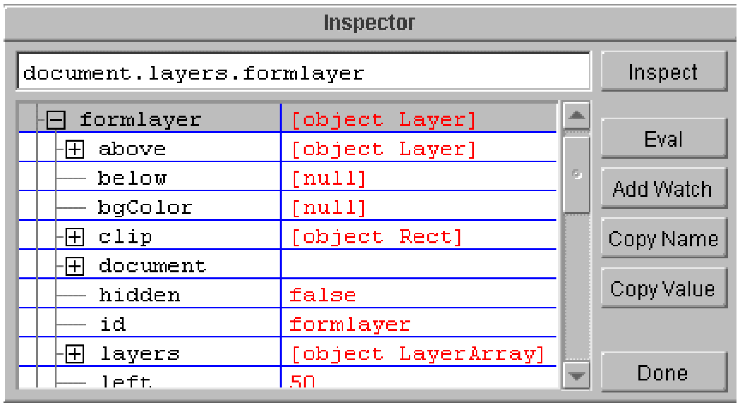
If you open the Page List while execution is stopped in the debugger, the page being debugged is highlighted in the list. If execution is not stopped, the highlighted page is not significant.

To refresh the Page List, from the File menu, choose Refresh All. This also refreshes Source View windows.

Object Inspector

The Object Inspector, shown in Figure 6, lets you inspect and work with the property values of an object. If a property value is itself an object, the inspector lets you see that object's property values and so on.

Figure 6 Object Inspector



```
document.layers.formlayer
```

⊟ formlayer	[object Layer]
⊞ above	[object Layer]
below	[null]
bgColor	[null]
⊞ clip	[object Rect]
⊞ document	
hidden	false
id	formlayer
⊞ layers	[object LayerArray]
left	50

The Object Inspector is divided into three sections:

- The top line initially shows the object being inspected.

  This area is similar to the input line in the console. You can type JavaScript expressions on this line. When you press Return or choose Inspect, JavaScript evaluates the expression and inspects the result, updating the main table.

- The main table shows the properties and values for expression being inspected.

  A plus sign ⊞ to the left of a property name indicates that the value is an object and that the object's properties are not currently shown. A minus sign ⊟ to the left of a property name indicates that the value is an object and that the object's properties are currently shown. You can click the plus and minus icons to expand and contract the property values.

  If you double-click on a line in this table, the property value on that line becomes the new value being inspected. For example, in Figure 6, if you double-click on the line for the `clip` property, the top line becomes `document.layers.formlayer.clip` and that object is inspected.

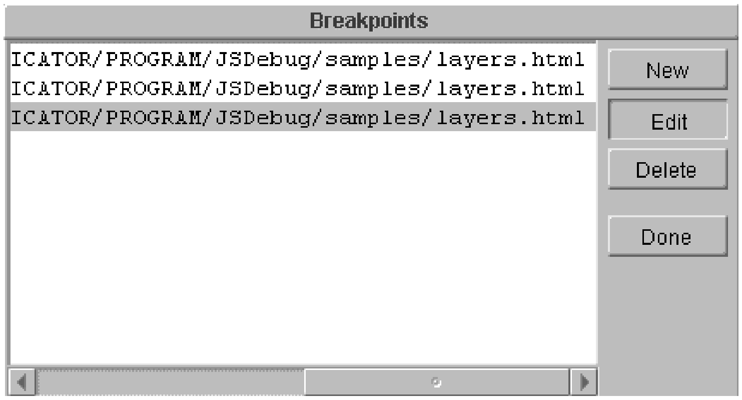
- The command buttons to the right allow you to perform various actions.
  - Inspect evaluates the expression currently in the top line and inspects the result.
  - Eval evaluates the expression currently in the top line without changing what is being inspected.
  - Add Watch adds the expression currently in the top line to the Watch window. For information on the Watch window, see “Working with Watches” on page 19.
  - Copy Name copies to the clipboard the property name of the line selected in the main table. This lets you copy this information between parts of the debugger or to other applications.
  - Copy Value copies to the clipboard the property value of the line selected in the main table.
  - Done dismisses the Object Inspector.

The Object Inspector is a powerful tool for examining the current state of your JavaScript objects.

Breakpoint Window

The Breakpoint window, shown in Figure 7, allows you to manipulate breakpoints. For information on breakpoints, see “Working with Breakpoints” on page 15.

Figure 7 Breakpoint window

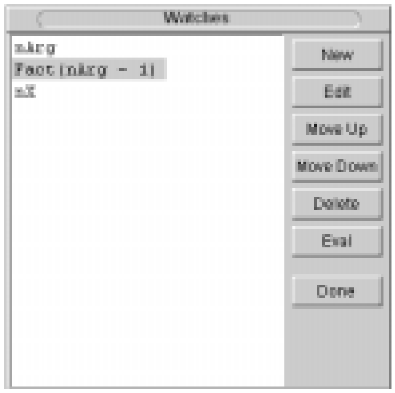


```
ICATOR/PROGRAM/JSDebug/samples/layers.html
ICATOR/PROGRAM/JSDebug/samples/layers.html
ICATOR/PROGRAM/JSDebug/samples/layers.html
```

Watches Window

The Watches window, shown in Figure 8, lets you manipulate watches. For information on watches, see “Working with Watches” on page 19.

Figure 8 The Watches window.



```
nArg
Fact(nArg - 1)
nX
```

Opening a page to debug

There are two ways to open a page you want to debug. The simplest method is as follows:

1. In the debugger, click the Intrpt button.

   This sets an interrupt. (See "Working with Interrupts" on page 14 for information about interrupts.)

2. In Navigator, load the page you want to debug.

   Navigator starts loading the page. As soon as it encounters JavaScript code on the page, it stops execution and opens a Source View window containing the HTML source for that page.

Alternatively, you can do the following:

1. In Navigator, open the page you want to debug.

2. Click the Open button on the debugger toolbar.

   The Page List window appears containing a list of the pages you have opened in Navigator since starting the debugger. The pages are sorted with the most recently opened listed first.

3. Select the page you want to open in the debugger and click Open to display the page in a Source View window.

4. Set a breakpoint in the code where you want the interpreter to stop and activate the debugger. (See "Working with Breakpoints" on page 15 for information on setting breakpoints.)

5. In Navigator, reload the page to start debugging.

Working with Interrupts

Setting an interrupt tells the JavaScript interpreter to halt execution as soon as it can. If JavaScript code is running when you set the interrupt, the interpreter immediately halts execution and activates the debugger. If JavaScript code is not running when you set the interrupt, the interpreter waits until JavaScript code executes. At that time, the interpreter halts and activates the debugger.

You can set an interrupt for two major purposes:

- Initiate debugging

  Set an interrupt when JavaScript code is not running and then load a page in Navigator that contains JavaScript. This is the simplest way to start debugging a particular page containing JavaScript code.

- Debug running code

  Set an interrupt while the interpreter is executing JavaScript code. The interpreter immediately stops. For example, if you've somehow managed to get an infinite loop into your code, setting an interrupt while executing the code could let you see where the problem is.

To toggle an interrupt

- Click the Intrpt button on the toolbar.

  If the Intrpt button is depressed, then an interrupt has been set. If the button is not depressed, an interrupt has not been set. Clicking the button toggles between these states.

Working with Breakpoints

A breakpoint is a specific location at which program execution is halted so that you can review the program's status up to that point. You set a breakpoint on a particular line of code. When execution reaches that line of code, the JavaScript interpreter stops and activates the debugger.

Breakpoints can either be unconditional or conditional. With an *unconditional* breakpoint, when the interpreter reaches the line of code, it always stops and activates the debugger. With a *conditional* breakpoint, when the interpreter reaches the line of code, it evaluates the condition (a JavaScript expression). If that expression returns `true`, then the interpreter activates the debugger. If the expression returns any value other than `true`, the interpreter does not activate the debugger.

As you select items in the Source View, the state of the BrkPt button changes.

- If there is no selection, the button is grayed.
- If there is a breakpoint set for the current selection currently, the button is depressed.
- If there is not a breakpoint set for the current selection currently, the button is not depressed.

You can manipulate breakpoints in the Breakpoint window, shown in Figure 7.

The breakpoints you set in one debugger session are saved with the debugger. The next time you start the debugger, the same breakpoints are automatically in effect.

To toggle a breakpoint

Do one of the following:

- In the Source View window, click to the left of the line (in the whitespace) where you want to set or remove a breakpoint.
- Select text in the Source View window and toggle the BrkPt button on the toolbar or use the right-button menu.

- In the Breakpoint window, choose New or Delete to manually enter or remove breakpoints.

When you set a breakpoint, a red dot appears by the line in the margin of the Source View window, indicating an unconditional breakpoint. When you remove a breakpoint, the dot disappears.

You can set a breakpoint either when execution is stopped or while JavaScript code is running. If execution is stopped when you set a breakpoint, you can restart execution by clicking the Run button in the debugger.

To manually add a breakpoint

You may wish to add a breakpoint to a line of code without first opening the file in a Source View window. In general, because you do not have to manually type the URL and the line number, it is much easier to add breakpoints through a Source View window. However, if you know precisely where the breakpoint should occur, follow these steps:

1. From the Windows menu, choose Breakpoints.

   The Breakpoint window appears.

2. Choose New to create a new breakpoint.

   The Edit Breakpoint dialog, shown in Figure 9, appears.

3. In the line text box, enter the line number of the breakpoint.

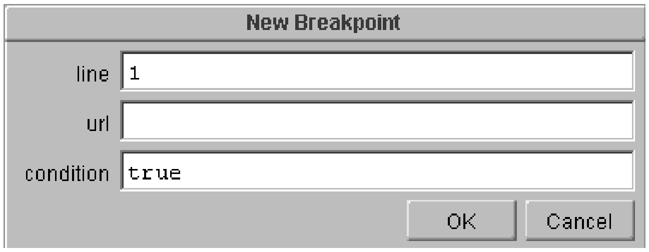
4. In the url text box, enter the complete URL of the file.

   You need to enter the URL exactly.

5. If the breakpoint is to be conditional, add the condition in the condition text box.

   Conditional breakpoints are discussed in "To edit a breakpoint's condition" on page 17.

Figure 9 New Breakpoint dialog box


New Breakpoint

line: 1

url:

condition: true

To edit a breakpoint's condition

Any breakpoint can be made into a conditional breakpoint by editing its condition as follows:

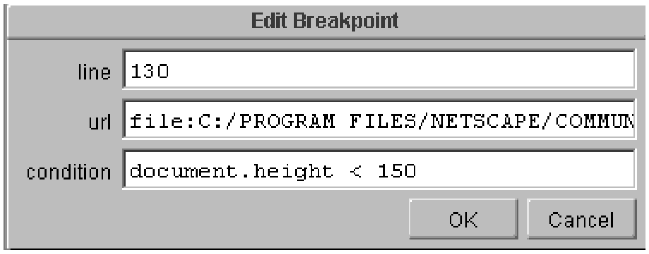
1. Select the line in the Source View window which holds the breakpoint.
2. From the Control menu, choose Edit Breakpoint.

   A dialog box appears in which you can edit the condition expression. When this dialog first appears, the condition is `true`. This means "always stop". You can enter any JavaScript expression as the condition.

Instead of using a Source View window to select the breakpoint to edit, you can select the breakpoint in the Breakpoint window and choose Edit.

Whichever method you choose, the same dialog box, shown in Figure 10, appears.

Figure 10 Editing a breakpoint


Edit Breakpoint

line: 130

url: file:C:/PROGRAM FILES/NETSCAPE/COMMUN

condition: document.height < 150

When you make a breakpoint conditional, the red dot by the line in the margin of the Source View window becomes orange. When you remove the breakpoint, the dot disappears.

When the interpreter evaluates the condition expression, it provides no automatic feedback. Any errors in the condition expression are ignored and discarded by the interpreter.

The interpreter stops execution and activates the debugger only when the expression evaluates to `true`.

Note The condition expression must literally evaluate to `true`. If it evaluates to anything else (such as 0, `false`, 1, or `"Success"`), the interpreter does not activate the debugger.

Conditional Breakpoint Examples

The following are simple examples of condition expressions.

- Halt if the value of the variable `x` is either greater than or equal to 5 or its value is `-1`:

  ```
  x >= 5 || x == -1
  ```

- Halt when a particular URL is reached:

  ```
  document.URL == "http://warp/index.html"
  ```

- Halt if `my_var` has a value:

  ```
  my_var != null
  ```

You can use conditional breakpoints in other situations. For example, you may want to log information to the Java console each time a line of code is executed, but not stop on that line. To do so, use an expression similar to the following:

```
(java.lang.System.out.println(“myFunction called with x = " + x)) &&
false
```

With this condition, the debugger always prints to the console, but does not stop since the entire expression returns false.

A similar situation would be to log information every time the code is executed but activate the debugger only if the value of `prop.length` is 5. To do so, use this expression:

```
(java.lang.System.out.println(“myFunction called with x = " + x)),
prop.length == 5
```

You can also use a conditional breakpoint to change values at runtime. Consider the following expression:

```
(x == -1) && (x = 0) && false
```

When the interpreter executes the line of code that contains that condition, it first checks to see if the value of `x` is -1. If so, it sets the value of `x` to 0. The interpreter does not stop for this condition, since it always returns `false`.

Working with Watches

A watch is an expression you enter that the interpreter automatically evaluates each time the interpreter stops and activates the debugger. The evaluation occurs for breakpoints, interrupts, or steps. You use the Watches window, shown in Figure 8, to manipulate watches. The Watches window displays the current list of watch expressions and allows you to manipulate the list and edit the items in the list.

You can set as many watch expressions as you like. The interpreter evaluates them in the order in which they occur in the Watches window. The interpreter displays each watch expression and its output in the Console.

Watch expressions are not specific to a particular page. They are evaluated every time the debugger is activated for any page, regardless of the page from which you set the watch.

The watches you set in one debugger session are saved with the debugger. The next time you start the debugger, the same watches are automatically in effect.

To add a new watch

You can add a new watch to the list in three ways.

To add text from a Source View window:

1. In a Source View window, select the text you want to add.
2. From the Edit menu, choose Copy to Watch.

   This adds the selected text as the last item in the Watches window. It also causes the interpreter to evaluate all items in the Watches window. Choosing Copy to Watch does not, however, bring the Watches window to the surface.
3. From the Windows menu, choose Watches.

   The Watches window appears. From there you can perform other operations on the watch expressions.

To add text from the Object Inspector:

1. Choose Add Watch.

   This adds the text currently in the top line of the inspector as the last item in the Watches widow. It also causes the interpreter to evaluate all items in the Watches window. Choosing Copy to Watch does not, however, bring the Watches window to the surface.
2. From the Windows menu, choose Watches.

   The Watches window appears. From there you can perform other operations on the watch expressions.

If you want to create a watch expression from scratch, do the following:

1. From the Windows menu, choose Watches.

   The Watches window appears.
2. Click the New button.

   The New Watch dialog appears.

3. Enter an expression in the New Watch dialog and click OK.

   Clicking OK confirms the change; if you want to discard the new watch, choose Cancel. If you clicked OK, the new watch expression is added as the last expression in the Watches window.

4. Click the Done button to indicate that you are finished making changes to the watches.

   The Watch window disappears.

To edit a watch expression

1. From the Windows menu, choose Watches.

   The Watches window appears.

2. In the Watches window, select the watch expression you want to edit.

3. Click the Edit button.

   The Edit Watch dialog appears.

4. Make the necessary changes and click OK to confirm the changes or Cancel to discard them.

5. Click the Done button to indicate that you are finished making changes to the watches.

   The Watch window disappears.

To delete a watch

1. From the Windows menu, choose Watches.

   The Watches window appears.

2. In the Watches window, select the watch expression you want to delete.

3. Click the Delete button.

   The selected expression disappears from the Watch window. There is no undelete.

4. Click the Done button to indicate that you are finished making changes to the watches.

   The Watch window disappears.

To change the evaluation order

The expressions in the Watch window are evaluated from top to bottom. To move a watch expression, do the following:

1. From the Windows menu, choose Watches.

   The Watches window appears.

2. In the Watches window, select the watch expression you want to move.

3. Click the Move Up button to move the expression earlier in the order. Click the Move Down button to move the expression later in the order.

4. Click the Done button to indicate that you are finished making changes to the watches.

   The Watch window disappears.

To evaluate all watches

To immediately evaluate all watches:

1. From the Windows menu, choose Watches.

   The Watches window appears.

2. Click the Evaluate button.

   The interpreter evaluates all watch expressions, displaying the expression and its result in the Console.

3. Click the Done button to indicate that you are finished working with the watches.

   The Watch window disappears.

Executing Code

The debugger provides several commands for executing sections of code. When stopped in the debugger, you have several options for how you execute code:

- You can continue execution and stop only when a breakpoint is reached.
- If the statement at which you're stopped is a function call, you can execute the individual statements of the function. (This is *stepping into* the function.)
- If the statement at which you're stopped is a function call, you can execute the entire function at once and stop after it returns. (This is *stepping over* the function.)
- If the statement at which you're stopped is *inside* a function call (instead of at the top level of the script), you can execute the rest of the containing function at once and stop after it has been executed. (This is *stepping out of* the function.)
- If the statement at which you're stopped is not a function call, stepping into and stepping over both simply execute that single statement. Stepping out continues execution until the next breakpoint.
- You can choose to skip executing the rest of the script and continue loading the HTML page after that script. (This is *aborting* execution.)

The following sections tell you how to perform these actions.

To continue execution

When stopped in the debugger, you may want to continue execution from the current statement and stop only when a breakpoint is reached. To do so:

- Click the Run button on the toolbar.

To “step into” function calls

When stopped in the debugger, if the line you’re stopped at is itself a function call, you may want to execute the individual statements in the function call. To execute the first statement inside the function call and then stop, do the following:

- Click the Into button on the toolbar at the location you want to begin.

To “step over” function calls

When stopped in the debugger, if the line you’re stopped at is itself a function call, you may want to execute the entire function and stop execution after the function returns. In this case, you do not step through the individual statements in that function call. To do so:

- Click the Over button on the toolbar at the location you want to begin.

To “step out” of a function call

When stopped in the debugger, if the line you’re stopped at is inside a function call, you may want to execute the rest of the statements in that function call and stop after the function returns. In this case, you do not step through the rest of the statements in that function call. To do so:

- Click the Out button on the toolbar at the location you want to begin.

To execute a single statement

When stopped in the debugger, if the line you’re stopped at is not a function call, you can execute that single statement. You can do either of the following:

- Click the Into button on the toolbar at the location you want to begin.
- Click the Over button on the toolbar at the location you want to begin.

To terminate a function call

When stopped in the debugger, you may want to terminate the current function and continue execution after it. To do so:

- Click the Abort button on the toolbar.

Displaying line numbers

To toggle between displaying and hiding line numbers:

1. From the Edit menu, choose Preferences.
2. From that submenu, choose Show Line Numbers or Hide Line Numbers.

Clearing the Console Window

If you work with the console window for an extended period, it can contain a lot of information, making it difficult to find lines you're interested in. You may want to clear that window to make it easier to see what is later added.

To clear the console window:

- From the Edit menu, choose Clear Console.

This does not change the state of the debugger, it simply erases the content of the console.

Manually adjusting source lines

There might be cases where the marks in the left column are not correctly aligned with the actual source. To manually adjust these marks:

1. Right-click (on Windows or Unix) or command-click (on Macintosh) in the left column at the point you want to start the adjustment.
2. Move the mouse (while continuing to hold down the mouse button) up or down.

   A + or - sign appears to indicate where the adjustments have been made, and in which direction.

You can reset the manual adjustments by Alt-clicking in the left column.

Error Reporter Dialog

The Error Reporter dialog displays information about syntax errors and runtime errors in your JavaScript code.

The Error Reporter dialog contains OK, Debug, and Pass On buttons.

- The OK button discards the error and continues execution.
- The Debug button activates the debugger.
- The Pass On button passes the error along to the normal error handling mechanisms in Navigator. This might bring up the error dialog or call the page's `onerror()` handler, if it has one.