

Chapter 38

Water Debug: Integrated Debugging with Development

IN THIS CHAPTER

- ◆ Viewing objects in multiple ways
- ◆ Using the Stepper and Object Inspector
- ◆ Performing remote inspection

HAVING GOOD DEBUGGING TOOLS IS ESSENTIAL for rapid software development. The creator of the Water language, Christopher Fry, has spent much of his career developing powerful debugging tools. He has published a number of articles in the journal “Communications of the ACM” on advanced debugging systems.

The Water language and the Steam IDE provide a great debugging environment with tools for inspecting objects, stepping through Water expressions, remote debugging, and basic output to the console.

When trying to debug a program, it is often useful to select a part of a program and execute that one piece. If you modify a method, you can select just that method and press Execute. The old definition will be replaced by the new definition in memory. After changing the implementation of a method, select a call to that method and execute. This shows you the returned value from calling that method.

Debugging Water Programs

When debugging, it’s often useful to see an object in multiple formats and views. If you’re working on a data structure, for example, you might want to see all the fields of the object, a presentation of that data in HTML format, or perhaps just the name of the object. Every execution returns an object, and the IDE can easily show four different views of the object: ConciseXML, rendered HTML, HTML source, and the Object Inspector.

The two right-side panes show two different formats for the value. Upon successful execution, the upper-right pane shows the ConciseXML text form of the return

value, and the lower-right pane shows the rendered HTML. The `to_concise_xml` method is called on the returned value, and the output from `to_concise_xml` is shown in the upper-right pane. Clicking anywhere in the upper-right pane opens the Object Inspector.

The `to_html` method is called on the returned value as well, and the string output is sent to the HTML viewer. The HTML viewer renders the HTML markup in the pane. Clicking the HTML output shows the HTML-formatted output in text form.

Print to Console

Printing values to a console is a very common debugging technique. The console shows a continuous log of the formatted text values printed to the console. The console for the Steam IDE is the Java Web Start console. To print an object to the console, call the `echo` method.

```
<echo "testing"/>
```

In the console you see a new line with `Echo: testing`. `echo` can take any number of values to print.

```
<echo number 100 true <thing "foo"/> /> → <thing "foo"/>
```

The following line shows the output that appears in the console:

```
Echo: thing.number 100 true <thing "foo"/>
```

`echo` returns the last value in addition to printing to the console, so you can set the value of `<echo 100/>` to a variable.

Inspecting Objects

The most useful tool in the IDE for exploring Water objects is the Object Inspector. It provides a browsable view of Water objects. To show the inspector, either click in the upper-right pane, or select an `Inspect` option from the Tools menu. The inspector can also be called from within your Water code by using `ii`. This is very useful when you want to inspect a value within your code.

```
<set x=<thing y=10/>.ii />
```

The preceding `ii` code opens the inspector window so that you can view the `<thing y=10/>` object.

```
10.<plus 1/>.ii.<times 10/>.ii.<minus 2/> → 108
```

```
10.<plus 1/>.<times 10/>.<minus 2/> → 108
```

The inspector can be called within a Water path as well. The inspector has no effect on the returned value. The following example shows calling the inspector with a label as the first argument (Figure 38-1).

```
<vector 10 20/>.<ii "vec"/>.
  <for_each returns='all'> value.<ii key/> </for_each>
→ <vector 10 20/>
```



Figure 38-1: Steam Inspector

The inspector can also be called with a label as the first argument. This is particularly useful when you have multiple calls to inspect and you need to distinguish between each call. The preceding example shows calling inspector with a label of the current key. The inspector shows a line for each call to inspect. If there is a label, then the label will appear at the start of every line. The first line in the inspector will be labeled `vec`; the second and third lines will be labeled `0` and `1`.

The name `ii` was chosen to make it easy to find the inspect points in your code. Search your code for `ii` to find all occurrences of calling the inspector.

The inspector has a top, middle, and bottom section – see Figure 38-1. The top section has some buttons, check boxes, and a type-in field. The list of fields to show can be filtered using the text box and check boxes in the top section. The middle section shows a navigation history list where each line represents an object. The bottom section shows the fields of the currently selected object in the middle section. Clicking a field of an object in the bottom section shows a new line in the middle section as well as the fields of the new object in the bottom.

Other Features

Other features are described in the following sections.

Stepping

Stepping can be very useful when you want to understand a particular program flow in detail. Stepping in the IDE highlights the exact Water expression being executed, not just the line. Second, at each step it shows you the value of that expression.

```
<defmethod compute x y=datetime.current>  
  y.hour.<plus x/>.<concat "PM"/>  
</defmethod>
```

```
<compute x=10.<plus 2/> />
```

Stepping works for any Water expression, including HTML tags. Stepping through HTML is quite interesting because it shows you the precise flow of execution.

```
<FORM action="foo".<concat "bar"/> >  
<H1>Form: <do 1.<plus 2/> /></H1>  
<INPUT value="test" />  
<INPUT type="submit" value="Click"/>  
</FORM>
```

Viewing Past Values of an Expression

The Steam IDE can also show past values for expressions. This means that you can run through the execution of a program, and then go back and view the value that an expression had during the previous execution.

The additional information starts to be recorded when you start stepping. To record expression values, start stepping and hit Execute. Now you can place your cursor at the end of any expression and click Value to see the value.

Remote Inspection

A pure Water Server can be created on a specific port and serve the top level `thing` object. This enables you to execute any Water code on that remote server. This is a big security risk, so it should only be done within a trusted environment.

The following example creates a server on port 8090 that will accept any Water expression in the URL.

```
<server.pure_water root_object=thing port=8090/>
```

Now you can view any Water object by typing in a Water Path in the browser location field.

```
http://localhost/number._id  
→ "_id"
```

Summary

This chapter covered several useful tools in Water Debug. You can programmatically call the Object Inspector, see multiple views of an object, and step through code expression by expression. These tools greatly assist the developer in their debugging tasks. The edit-run-debug cycle is reduced to less than one second.

Debugging is a very natural part of Water development. Water offers powerful debugging features that make it easier and faster to find and fix problems in your programs.

