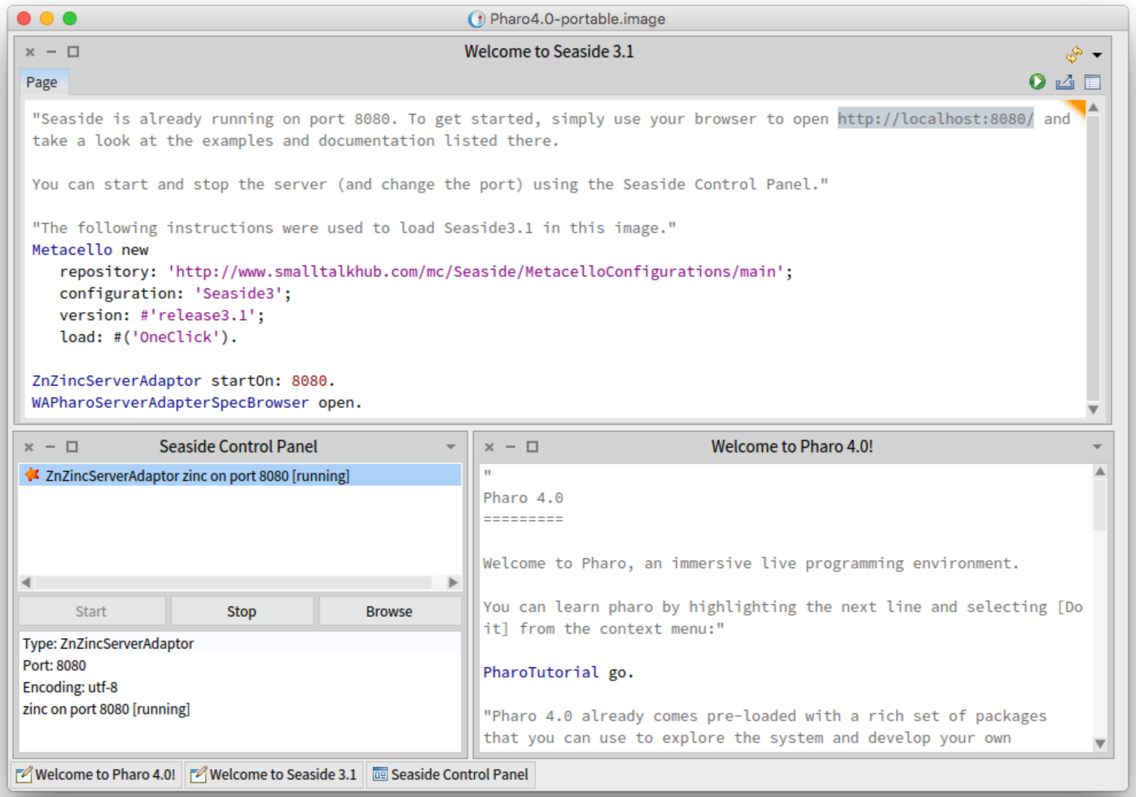
When learning a new programming language it is traditional to start with a 'Hello World!' application (see http://en.wikipedia.org/wiki/Hello_world_program). We will follow that practice and use this opportunity to introduce you to the programming environment and the Smalltalk programming language.

1. Start by launching the Seaside One-Click Experience executable as described in chapter 1. This will open a copy of 'Pharo4.0-portable.image' using Pharo. Three windows are already open (clockwise from the top): (1) a Workspace titled 'Welcome to Seaside 3.1', (2) a Workspace titled 'Welcome to Pharo 4.0!', and (3) a Seaside Control Panel. We will briefly examine each of these in turn.



```
"Seaside is already running on port 8080. To get started, simply use your browser to open http://localhost:8080/ and
take a look at the examples and documentation listed there.

You can start and stop the server (and change the port) using the Seaside Control Panel."

"The following instructions were used to load Seaside3.1 in this image."
Metacello new
    repository: 'http://www.smalltalkhub.com/mc/Seaside/MetacelloConfigurations/main';
    configuration: 'Seaside3';
    version: #'release3.1';
    load: #('OneClick').

ZnZincServerAdaptor startOn: 8080.
WAPharoServerAdapterSpecBrowser open.
```

```
ZnZincServerAdaptor zinc on port 8080 [running]

Type: ZnZincServerAdaptor
Port: 8080
Encoding: utf-8
zinc on port 8080 [running]
```

```
"
Pharo 4.0
=========

Welcome to Pharo, an immersive live programming environment.

You can learn pharo by highlighting the next line and selecting [Do
it] from the context menu:"

PharoTutorial go.

"Pharo 4.0 already comes pre-loaded with a rich set of packages
that you can use to explore the system and develop your own
```

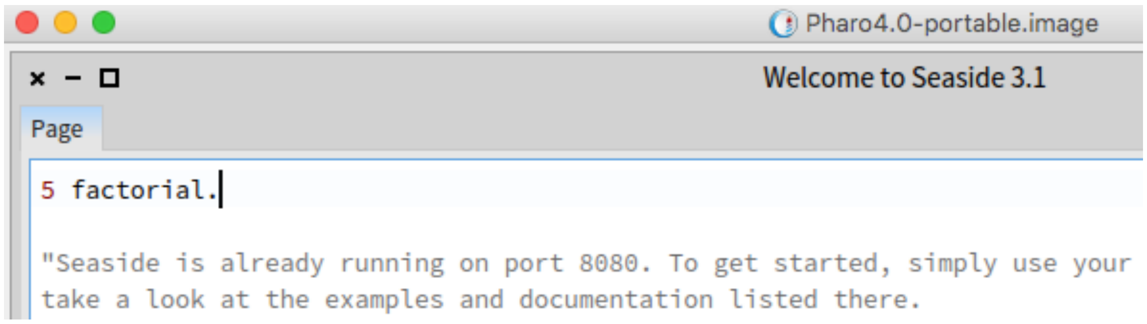
2. The two windows titled "Welcome to …" are examples of a **Workspace**, or a text area. These workspaces are pre-loaded with some information about Seaside and Pharo (respectively). The Pharo welcome workspace includes instructions for starting a tutorial. This is a good exercise.

3. You can also use a workspace to evaluate expressions and (optionally) display or inspect the results. Since this might be your first exposure to Smalltalk code, we'll give a brief introduction to evaluating Smalltalk code in a workspace. (If you already know Smalltalk or are just impatient, you can skim or skip this discussion!)

Click in a Workspace window at the top left and press <Enter> a couple times to get some space. Then go back to the top and type the following:

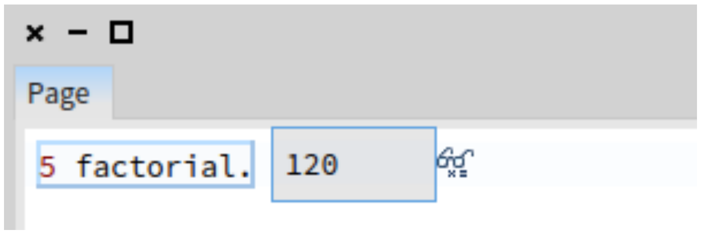
```
5 factorial.
```

The Workspace should look as follows:



```
5 factorial.

"Seaside is already running on port 8080. To get started, simply use your
take a look at the examples and documentation listed there.
```

Next, press <Ctrl>+<P> (a short cut for "print-it") to evaluate the expression, and show the returned object. The result should be '120' in a box with a small pair of eyeglasses next to it. Here a *unary* message, 'factorial,' is being sent to the integer 5, which answers the integer 120.

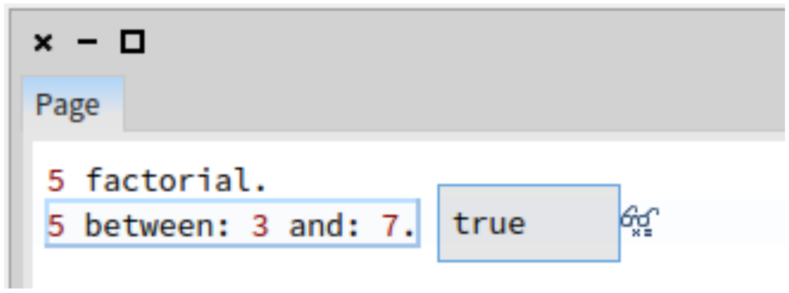


```
5 factorial.  120
```

4. In the Workspace add a new line, type the following, and 'print-it' (again, using <Ctrl>+<P>):

```
5 between: 3 and: 7.
```

The Workspace should look as follows:



```
5 factorial.
5 between: 3 and: 7.  true
```

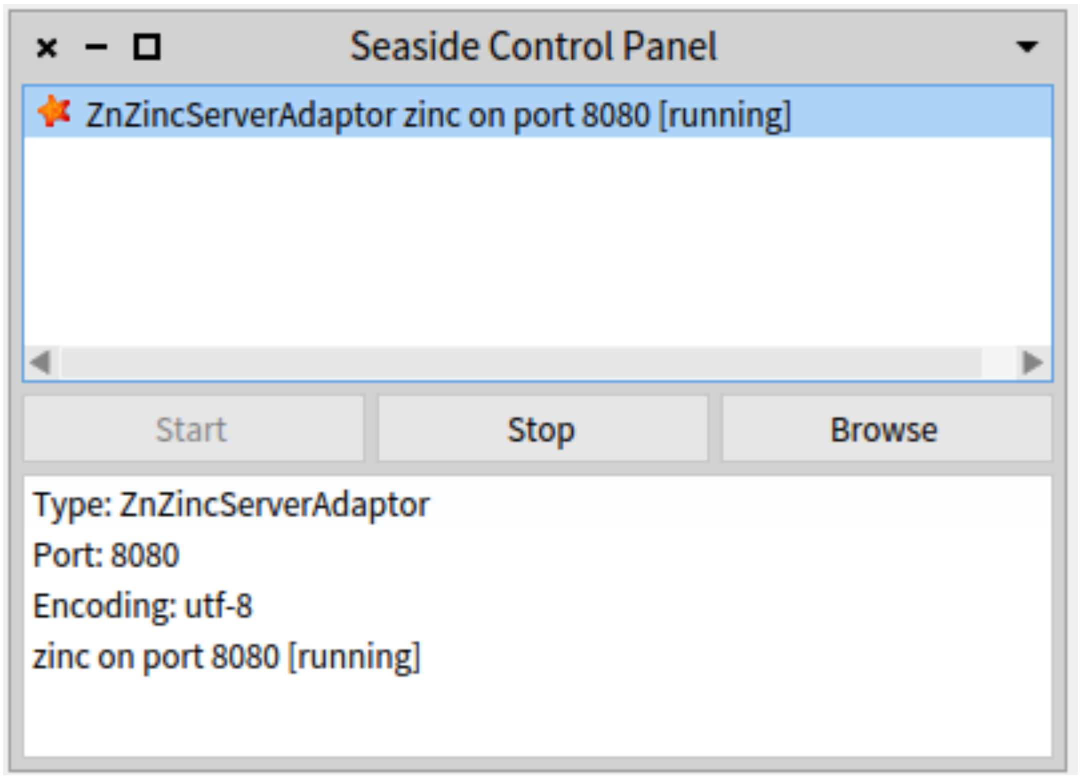
Here a single *keyword* message, 'between:and:', is being sent to the integer 5 with two integer arguments, 3 and 7, and it answers a Boolean. Smalltalk uses this keyword syntax rather than the more common position-based list of arguments, such as inRange(5,3,7), because the position-based list is ambiguous as to the meaning of each argument.

5. In addition to the *unary* message and the *keyword* message, Smalltalk defines a *binary* message that is generally used to handle what other languages do with operators. Like other messages, the format is a receiver followed by a message (with optional arguments), but this one uses punctuation characters for the message name and always has exactly one argument. In the Workspace add a new line, type the following, and ‘print-it’ (using <Ctrl>+<P>).

```
2 + 3.
```

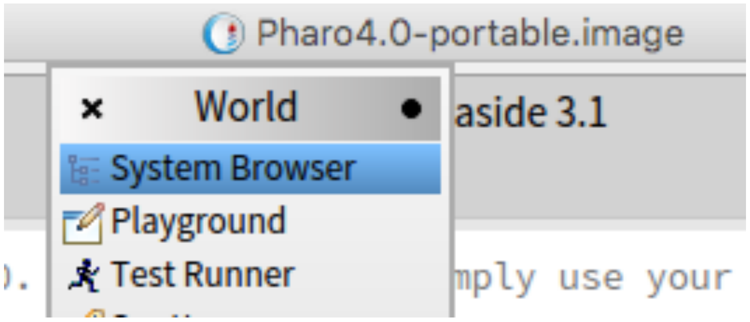
This should show 5 as the result of evaluating the expression.

6. **The Seaside Control Panel** is used to manage the web server built into the one-click image. In it you can create, configure, and delete a server. You can select a server and inspect it.

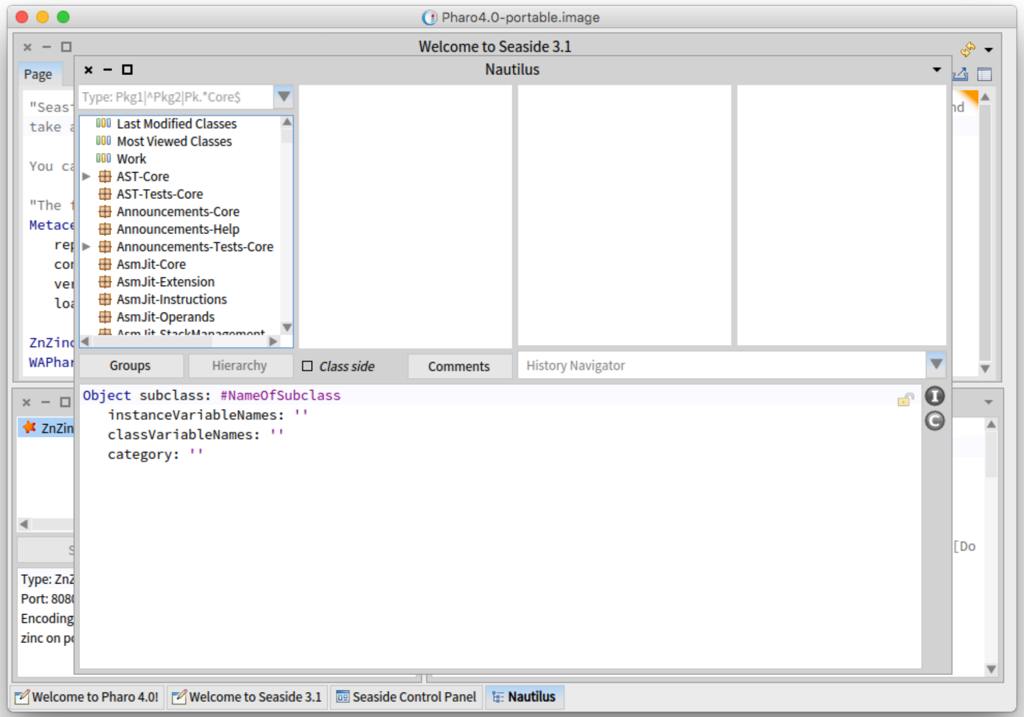


Type: ZnZincServerAdaptor
Port: 8080
Encoding: utf-8
zinc on port 8080 [running]

7. A **System Browser** is the tool typically used to browse code (classes and methods). To open one click on the background of the main window and select ‘System Browser’.

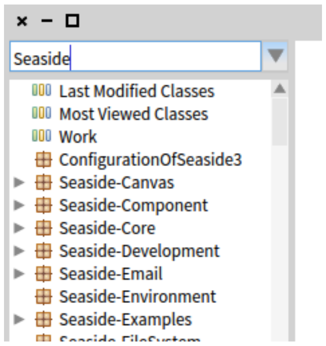


This should open a Nautilus window:

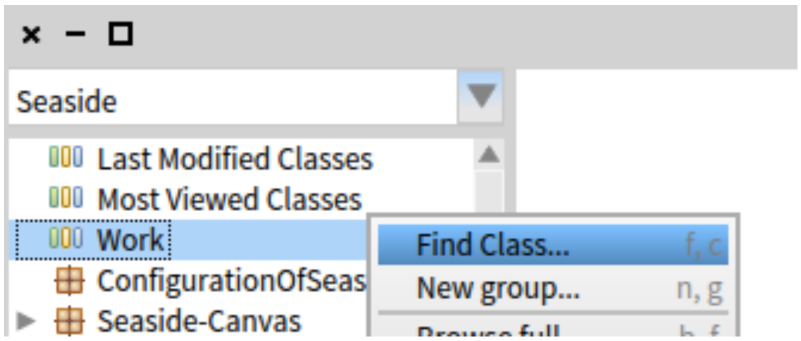


```
Object subclass: #NameOfSubclass
	instanceVariableNames: ''
	classVariableNames: ''
	category: ''
```

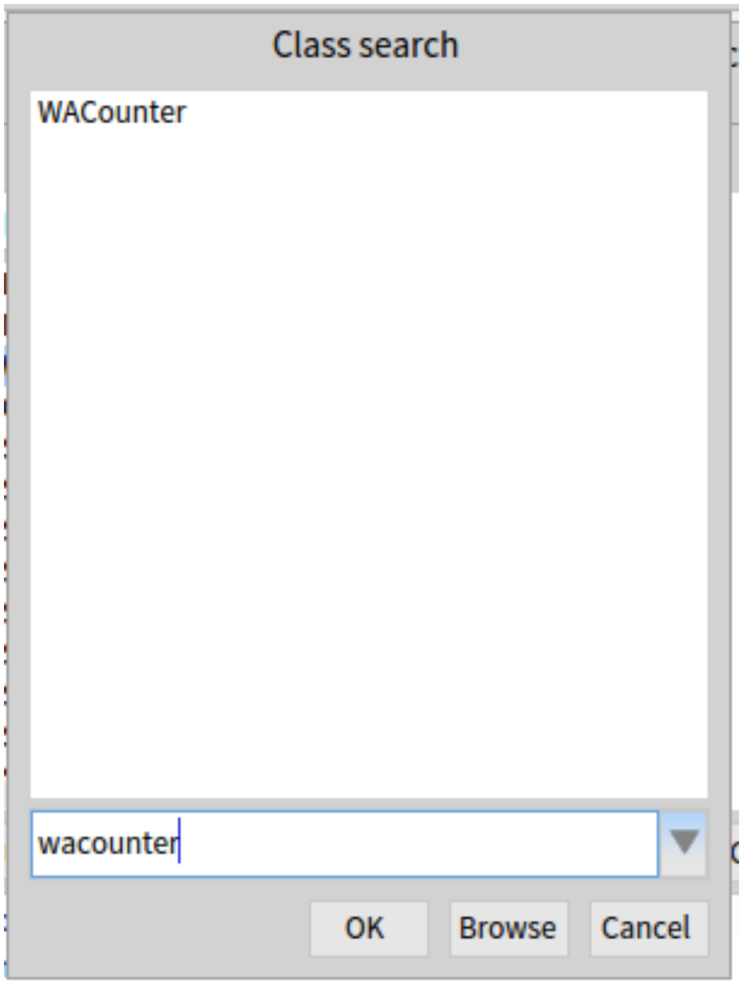
The first column contains a search drop-down text field above a list of packages containing classes. In the text field, type ‘Seaside’ (without the quotes) and notice how it filters the package list:



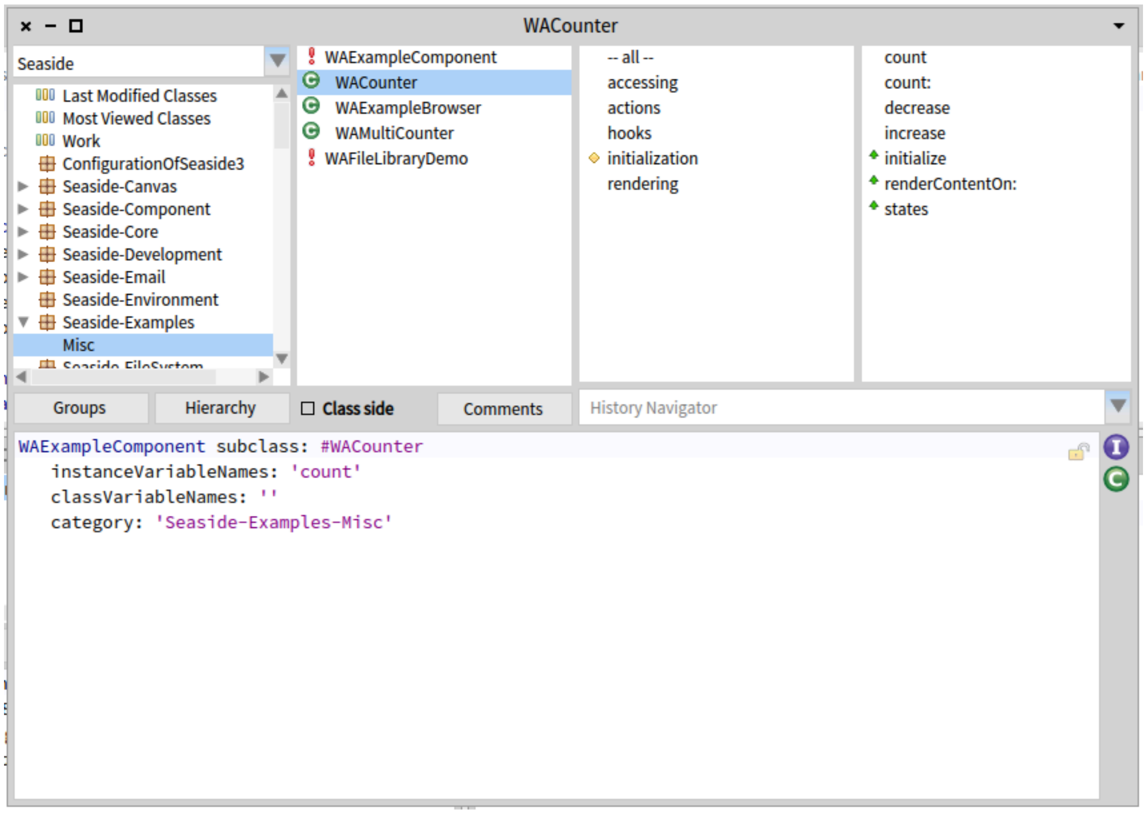
You can click the small triangle to the left of a package to expand its contents. You can also use the context menu (right or center mouse button?) to bring up a context menu. Select ‘Find Class…’ from this menu:



In the provided dialog enter 'wacounter' (all lowercase is fine) to filter the list:



If you click the 'OK' button the System Browser should show the WACounter class:



```
WAExampleComponent subclass: #WACounter
    instanceVariableNames: 'count'
    classVariableNames: ''
    category: 'Seaside-Examples-Misc'
```

The second column lists classes that are in the selected package ('Misc' in 'Seaside-Examples') in a combined alphabetical/hierarchal order. That is, classes in the list that do not have their superclass in the list will have no indent (such as 'WAFileLibraryDemo'). Subclasses are listed indented under their superclass if the superclass is in the same class category ('WACounter' is a subclass of 'WAExampleComponent'). Selecting a class name will show the class definition in the text area in the bottom half of the window. (The red exclamation mark next to a class name indicates that it does not have a class comment. Click the Comments button to see this.)

The third column is a list of method categories for the selected class (with an extra '-- all --' at the top of the list to show all methods). The fourth column is a list of methods in the class and method category selected. Selecting a method name in the fourth column will show that method's definition in the text area in the bottom half of the window.

Below the second column (the class list) is a checkbox labeled '*class side*' and a button labeled 'Comments'. Clicking on the 'Comments' button will show a class comment (if available) for the selected class. The '*class side*' checkbox is used to specify whether we are looking at methods that can be called on instances of the class or on the class itself. When the browser has selected the class 'WACounter' we see that it has five method categories (including 'rendering') and one method in the 'rendering' category. We generally refer to these methods as being 'on the *instance side*.'

If you click on the 'class side' checkbox you will see three method categories and four methods. These methods are 'on the *class side*' and define code that will run when a message is sent to the class 'WACounter.' Class-side methods are what other languages might describe as 'static functions' and generally support instance creation and singleton management (where the nature of the class is that there should be only one instance).

In Smalltalk, a typical beginner's programming error involves putting methods on the wrong 'side' of a class. In some cases there might even be methods with the same name (e.g., 'initialize') on the instance side and on the class side of a class. In the instructions that follow, pay special attention when a method is added to the class side so that when you are done you remember return to the instance side. Also, if things don't work, go back and see if the methods have been added properly to the correct side of a class.

Again, as I've worked with students going through this tutorial the most common error has been creating methods on the wrong 'side' of a class. If you reread the preceding paragraph you will save yourself problems later.

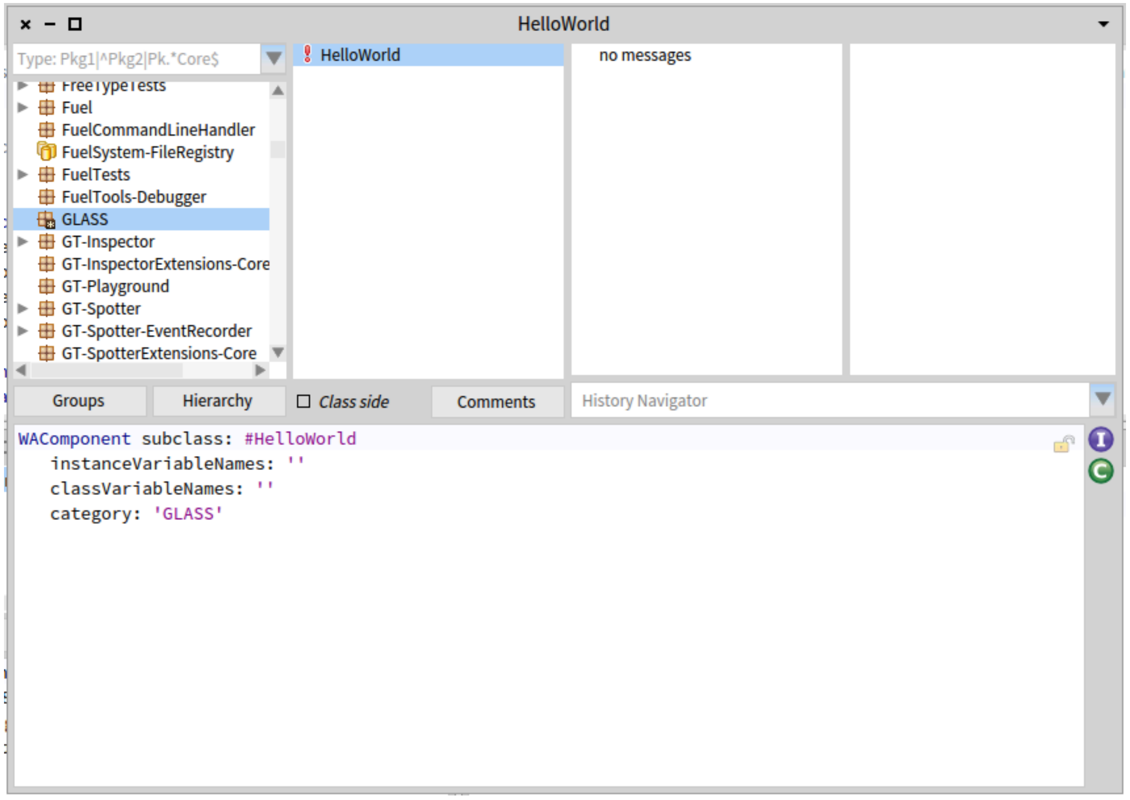
Now let's create a new class!

8. In Seaside, the basic user interface class is a subclass of WAComponent and any such component can be a root (the starting point for an application) and/or embedded in another component. **In the System Browser, click on any line in the first column to get a new class-creation template** (make sure that the 'class side' checkbox is unchecked)**.** In the text area replace the existing text (beginning with 'Object subclass: #NameOfSubclass') with the following and save it (<Ctrl>+<S>).

```
WAComponent subclass: #HelloWorld
   instanceVariableNames: ''
   classVariableNames: ''
   category: 'GLASS'
```

Unfortunately, copying from this document and pasting into Pharo will likely not preserve the formatting. If you want to copy/paste, you should use a simple text editor (such as Notepad on Windows, TextEdit on the Mac, or MousePad on Linux) as an intermediate paste/copy location.

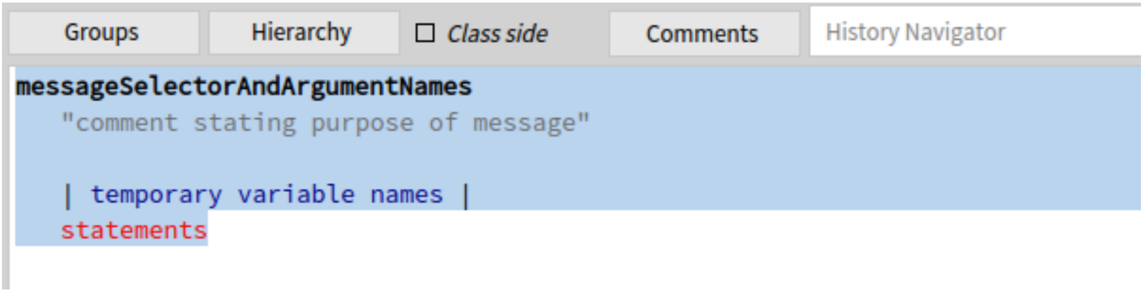
Depending on your environment, you might find that <Alt>+<S> (or <Apple>+<S> on the Mac) can be used instead of <Ctrl>+<S>. This should result in a new ‘GLASS’ entry in the class category list (the first column) and the single line ‘HelloWorld’ in the second column.



```
WAComponent subclass: #HelloWorld
    instanceVariableNames: ''
    classVariableNames: ''
    category: 'GLASS'
```

The first thing to note here is that we have created a subclass of WAComponent by sending a message to the class WAComponent, not by editing a text file and then sending it through a compiler and then starting an application. *In Smalltalk we do not ‘program’ by editing text files, but by interacting with existing objects in an existing, active, object-based environment using tools that are in that environment.* If we save this modified object space as an image, and then restart from that image, the class would still exist.

9. We are now ready to give our new class some behavior. To add an instance method to our HelloWorld class, ensure that the instance side of the class is selected (uncheck ‘*Class side*’), then click in the third column (the method categories list). This will change the text area from a description of the class to a method template.



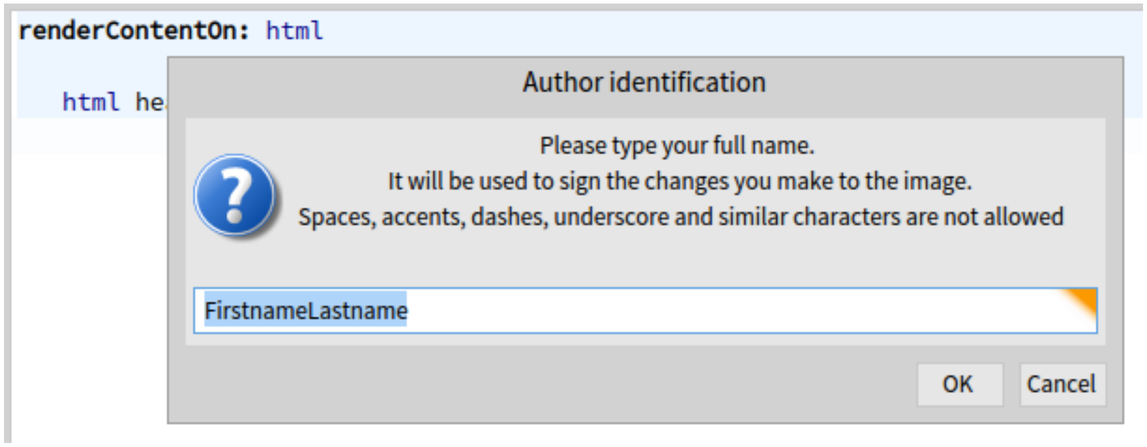
```
messageSelectorAndArgumentNames
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

In Smalltalk, methods start with the name of the method (the message selector) and the name of any arguments. The remainder of a method consists of Smalltalk expressions that are evaluated when the method is called. Just as all expressions return an object, all methods return an object (which, of course, the caller is free to ignore). By default, the method returns the receiver, but any expression following the 'circumflex accent', usually referred to as 'up arrow' or 'caret' (^), will be returned as the expression's result. (This up arrow is one of Smalltalk's two operators. See step #15 for the assignment operator.)

10. Perhaps the most important method in a Seaside component is 'renderContentOn:' which creates the HTML. In the spirit of the 'Hello world!' convention, replace the text in the text area with the following method.

```
renderContentOn: html

   html heading: 'Hello world!'.
```

11. When you save your first method (using <Ctrl>+<S>), Pharo asks for your name so it can keep a timestamp of who and when the method was last edited. In the following dialog box enter your name (with no punctuation characters) and press <Enter> or click the OK button:



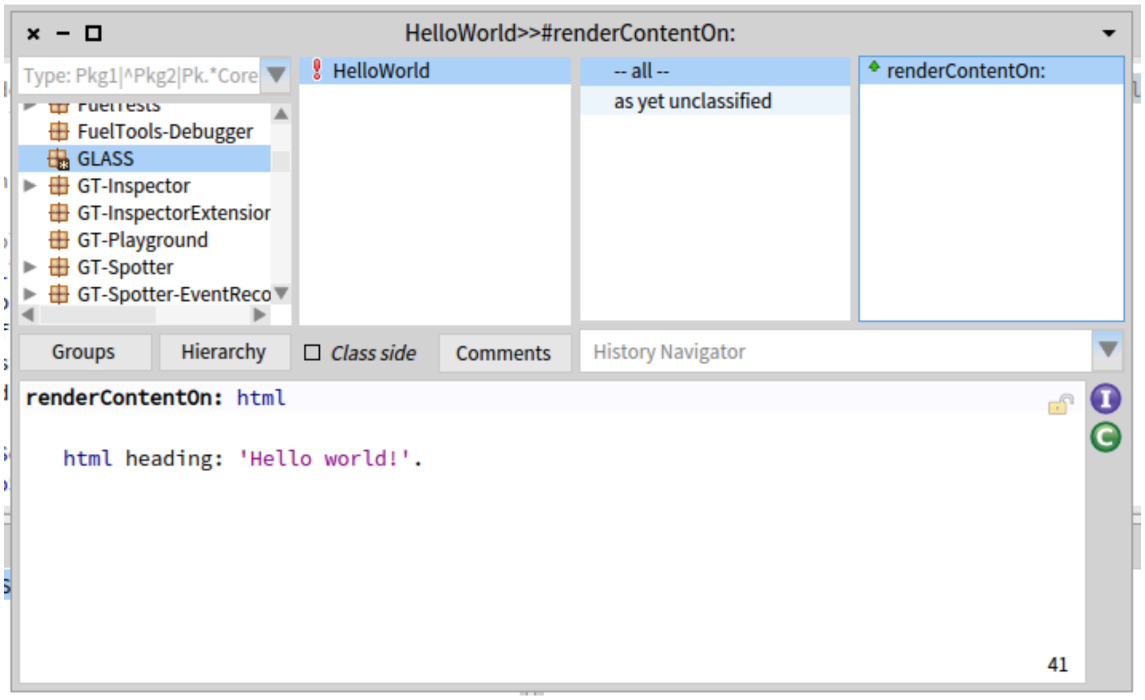
```
renderContentOn: html

   html he
```

Author identification

Please type your full name.
It will be used to sign the changes you make to the image.
Spaces, accents, dashes, underscore and similar characters are not allowed

FirstnameLastname

The result should be a System Browser that has a method category of 'as yet unclassified' and lists your method as 'renderContentOn:'. The green arrow pointing up next to the method name in the fourth column is an indicator that you have overridden a superclass method.

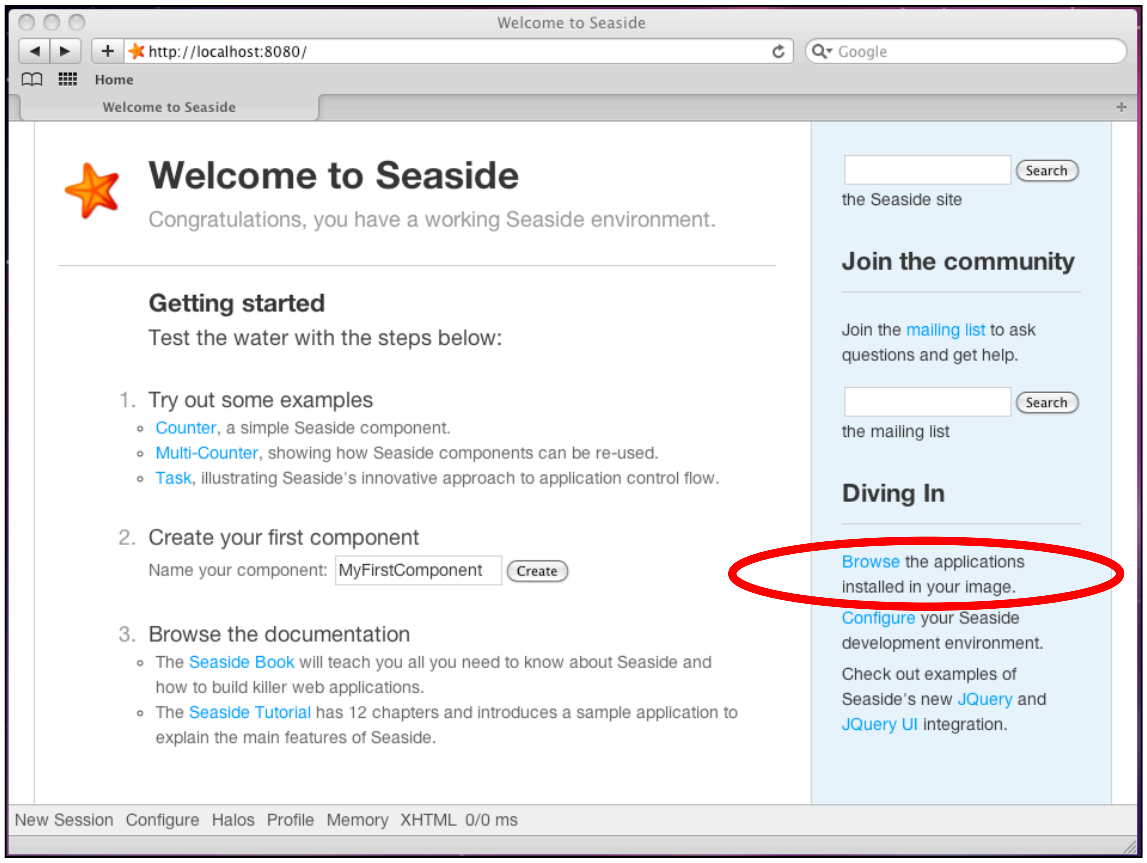


```
renderContentOn: html

    html heading: 'Hello world!'.
```

12. Now we need to inform Seaside that this new component can be used as a root component. In order to do this we need to tell the class what name to use when it registers itself as an application. Switch to the Workspace (the window titled 'Welcome to Seaside 3.1) and evaluate the following text by clicking anywhere in the line and typing <Ctrl>+<D> (for 'do-it'). If you have trouble, review step #3.

```
WAAdmin register: HelloWorld asApplicationAt: 'hello'.
```

13. If the above steps were successful, you should be able to open a web browser on [http://localhost:8080/](http://localhost:8080/) and click on the ‘Browse’ link.



Welcome to Seaside

Congratulations, you have a working Seaside environment.

Getting started

Test the water with the steps below:

1. Try out some examples
   - Counter, a simple Seaside component.
   - Multi-Counter, showing how Seaside components can be re-used.
   - Task, illustrating Seaside's innovative approach to application control flow.
2. Create your first component

   Name your component: MyFirstComponent Create
3. Browse the documentation
   - The Seaside Book will teach you all you need to know about Seaside and how to build killer web applications.
   - The Seaside Tutorial has 12 chapters and introduces a sample application to explain the main features of Seaside.

Search the Seaside site

Join the community

Join the mailing list to ask questions and get help.

Search the mailing list

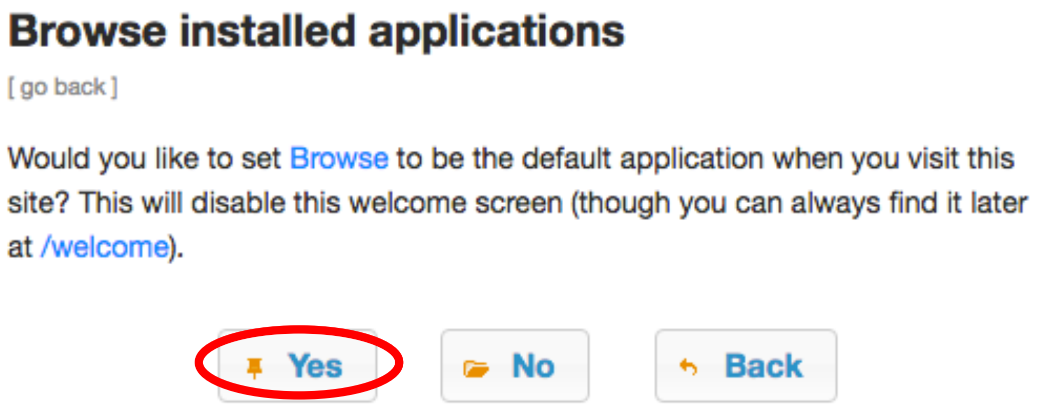
Diving In

Browse the applications installed in your image.

Configure your Seaside development environment.

Check out examples of Seaside's new JQuery and JQuery UI integration.

Before showing you a list of installed applications, Seaside prompts you to change your Seaside home page from the welcome page (above) to a page listing the installed applications. Click the ‘Yes’ button to change your home page.

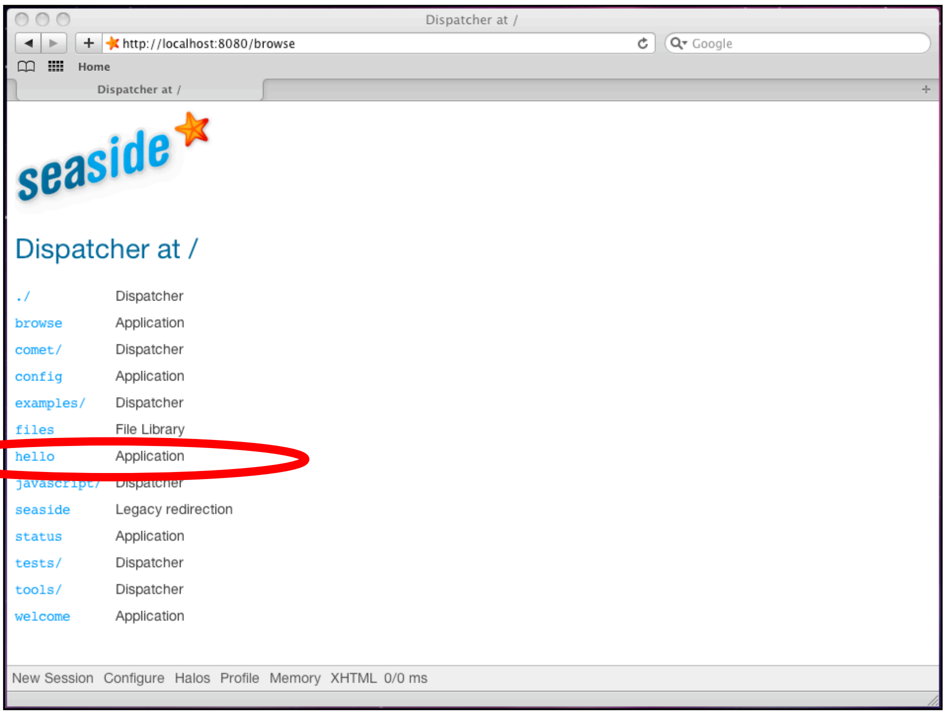


Browse installed applications

[ go back ]

Would you like to set Browse to be the default application when you visit this site? This will disable this welcome screen (though you can always find it later at /welcome).

The 'Browse Installed Applications' page gives you a list of the top-level links being served from by this web server, including our new 'hello' application.



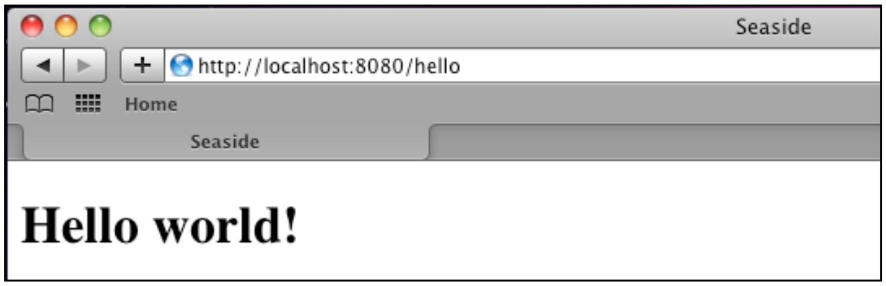
Dispatcher at /
http://localhost:8080/browse
Google
Home
seaside

Dispatcher at /

| | |
|---|---|
| ./ | Dispatcher |
| browse | Application |
| comet/ | Dispatcher |
| config | Application |
| examples/ | Dispatcher |
| files | File Library |
| hello | Application |
| javascript/ | Dispatcher |
| seaside | Legacy redirection |
| status | Application |
| tests/ | Dispatcher |
| tools/ | Dispatcher |
| welcome | Application |

New Session Configure Halos Profile Memory XHTML 0/0 ms

14. Click on the 'hello' link and note that the expected message is displayed. While not very sophisticated, we have demonstrated that we can create a *Web Application* by adding one method to a subclass of WAComponent. (Can you guess what the "WA" in WAComponent stands for?)



Hello world!

15. In order to make the application a bit more dynamic, modify the ‘renderContentOn:’ method as follows (the changed lines have been highlighted). Note that the line numbers are in comments (the double quote character delineates a comment), so do not need to be entered.

```
renderContentOn: html

"3"      | now |
"4"      now := DateAndTime now.
"5"      html heading: 'Hello world!'.
"6"      html heading
"7"         level3;
"8"         with: now;
"9"         yourself.
```

This code introduces some new syntax. Line 3 defines a method temporary variable, ‘now,’ that is declared by placing it between vertical bars (or pipe characters). In Smalltalk (like Perl, Python, PHP, and others), variables are dynamically typed (as opposed to the static typing of languages like C and Java where types must be declared at compile time). This means that all we need to do is specify the name and this will reserve space for the object reference.

Line 4 is an expression that introduces the second of Smalltalk’s two operators (the return operator was mentioned in step #9). The assignment operator, colon-equals (:=), is used to take the object returned by the expression on the right and place a reference to it in the variable defined on the left.

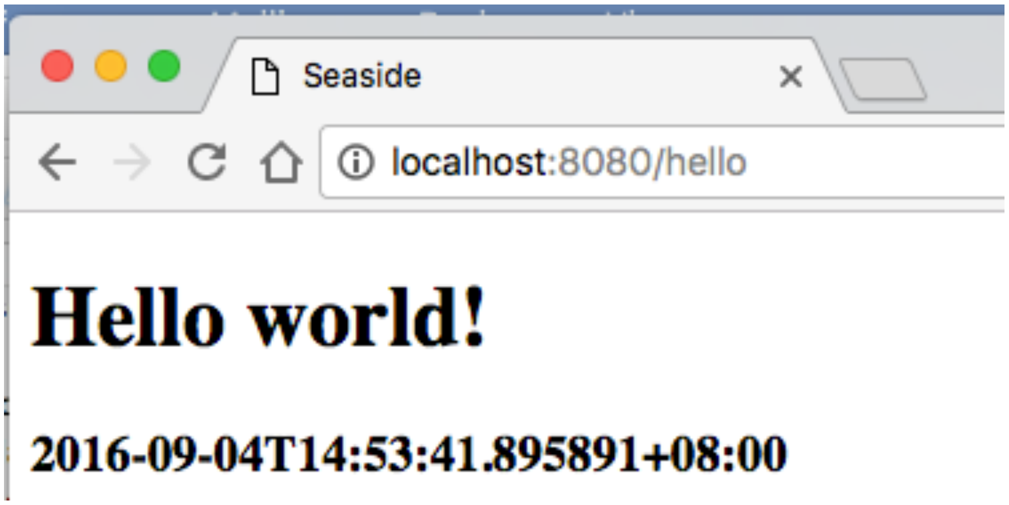
Line 5 remains from the original method and shows the greeting.

Lines 6-9 are a single expression. Since extra whitespace is ignored in Smalltalk you can add new lines and tabs for cosmetic styling. Next, recall that expressions are made up of receivers and messages and that there are three types of messages: unary, binary, and keyword (evaluated in that order). Before it was edited, the above method had only one message, the keyword message ‘heading:’ that was sent to ‘html’ with a String as an argument. The modified method has the same first message, but after it is a more complex expression. The expression starts with a unary message (‘heading’) that returns an instance of WAHeadingTag. This heading object is sent three messages, ‘level3’ (to set the level), ‘with:’ (to set the text contents), and ‘yourself’.

This semicolon indicates a ‘message cascade,’ meaning that that what follows is not a full expression (which would require an object reference to designate the receiver), but another message to the receiver of the most recent message. The receiver of the last message was the object returned from the ‘heading’ message send (an instance of WAHeadingTag), so the next message will be sent to the same object.

Finally, we have the 'yourself' message that is sent to the receiver of the most recent message. The 'yourself' message calls a method that simply returns the receiver. In our case, since we ignore the returned object, this message send is nothing more than (a slightly inefficient) cosmetic filler so that the previous line can end with a semicolon rather than a period (which would probably be the more common approach by most Smalltalkers). This programming style allows you to come back later and add another line or rearrange the lines without having to change the line ending character.

16. Now return to your web browser and refresh a few times and watch the time change.



Hello world!

2016-09-04T14:53:41.895891+08:00

17. At this point you can return to Pharo and save the image and quit (as described in Chapter 1).