JavascriptForJavaProgrammers

Christopher Oliver says _...In order to reduce the barrier to entry of the Cocoon flow layer for Java programmers that haven't done much with JavaScript I thought I'd give an overview of some of the similarities and differences between

JavaScript..._

Read the full message in the cocoon-dev archives.

In order to reduce the barrier to entry of the Cocoon flow layer for Java programmers that haven't done much with JavaScript I thought I'd give an overview of some of the similarities and differences between JavaScript (specifically Rhino) and Java.

I also encourage you to play around with the Cocoon Flow debugger, as it contains an interactive environment where you can type in JavaScript code and see the results.

JavaScript contains the following built-in objects and top-level functions:

Built-in Objects

- Array
- Boolean
- Date
- Error
- Function
- Math
- Number
- Object
- RegExp
- String

Top-level Functions

- decodeURI()
- decodeURIComponent()
- encodeURI()
- eval()
- escape()
- isFinite()
- isNAN()
- parseFloat()
- parseInt()
- unescape()

These objects and functions are comprehensively documented in many places on the web so I won't describe them further here. Please see Quick Reference or Netscape Core JavaScript Reference 1.5.

Type of Variables

JavaScript has two types of variables: global variables and local variables. Local variables are only visible in the function in which they are declared. However, they are also visible to other functions defined in the same scope. Inside a function definition local variables are identified by the **var** keyword:

```
function foo() {
    var x;
    x = 3; // x is a local variable
    y = 5; // y is a global variable
    function inner() {
        print(x); // refers to the var x declared above in foo()
    }
}
```

You should *always* use the **var** keyword to declare variables.

Statements

JavaScript statement syntax and control structures are pretty much the same as Java: if, for, switch, while, do, etc.

JavaScript also supports an "in" operator that tests whether an object contains a named property:

```
if ("propName" in obj) {
     // do something
}
```

The JavaScript "for" statement together with the "in" operator supports a kind of "foreach": it iterates over the properties of an object:

```
for (i in obj) {
     print(obj[i]);
}
```

As you can see you can also use a string argument to the [] operator. This is equivalent to using the "." (dot) operator with an identifier literal, i.e.

```
       foo.bar
```

is the same as

```
       foo["bar"]
```

A new instance of an object can be created with the "new" operator:

```
    var x = new String();
```

Alternatively you can use the literal syntax to create instances of built-in types:

```
Literal Syntax                              Equivalent using new
""                                          new String()
''                                          new String()
[]                                          new Array()
{}                                          new Object()
```

Using Java Objects

Rhino includes bidirectional support for Java objects: you can create Java objects and call their methods, and you can extend Java classes and implement Java interfaces in JavaScript.

Classes in packages under "java" are accessible directly in your scripts:

```
    var map = new java.util.HashMap();
```

Note that classes under "java.lang" are not automatically imported, however:

```
    var n = new java.lang.Integer(3);
```

All other java packages and classes are accessible under the property "Packages":

```
    var tree = new Packages.javax.swing.JTree();
```

You can get the effect of Java imports using the importPackage() and importClass() functions:

In Java:	**In Rhino:**	
import foo.*;	importPackage(Packages.foo);	
import foo.Bar;	importClass(Packages.foo.Bar);	

Note: Be sure to make importPackage(...) the first code line in your script, or you may get an "Ambiguous import" error on the second and following runs of your script when attempting to reference an imported class.

Rhino understands Java bean properties, so if your Java classes have getters and setters you can access them as properties in JavaScript:

```
  var d = new java.util.Date();
  d.year = 2003;    // same effect as d.setYear(2003);
```

Continuations

The Rhino interpreter that supports continuations adds the following two objects:

- Continuation
- ContinuationException

See http://marc.theaimsgroup.com/?l=xml-cocoon-dev&m=102781075226697&w=2 for a description of these.

And the Cocoon flow layer adds the following objects and functions to that:

```
// interface to various cocoon abstractions:
- cocoon
      - environment // property of type org.apache.cocoon.environment.Environment
      - parameters  // JavaScript array of <not sure?>
      - request // property of type org.apache.cocoon.environment.Request
      - response // property of type org.apache.cocoon.environment.Response
      - session // property of type org.apache.cocoon.environment.Session
      - context // property of type org.apache.cocoon.environment.Context
      - componentManager // property of type org.apache.avalon.framework.ComponentManager
      - load(fileName) // loads a script
      - createSession() // attaches this flow script instance to session
      - removeSession() // detaches this flow script instance from session

      // action/input/output module interfaces:
      - callAction(type, source, parameters)
      - inputModuleGetAttribute(type, attribute)
      - outputModuleSetAttribute(type, attribute, value)

// flow API
- sendPage(uri, bizData); // call presentation layer to present bizData
- sendPageAndWait(uri, bizData, timeToLive) // call presentation layer to present bizData and
                                            // wait for subsequent request

// action/input/output module interfaces:
- act(type, src, param)
- inputValue(type, name)
- outputSet(type, name, value)
- outputCommit(type)
- outputRollback(type)

// logging support:
- print(args) // prints args to standard out

// log object: provides access to cocoon logging system via its methods:
- log
   - error(message)
   - debug(message)
   - warn(message)
   - info(message)
```

See Also

- RhinoShell
- For a higher level description of the Cocoon objects see: http://www.webweavertech.com/ovidiu/weblog/archives/000042.html
- For more on using Rhino for scripting Java see: http://www.mozilla.org/rhino/scriptjava.html