



Extend Your Solutions with Advanced JavaScript

ServoyWorld 2008

Greg Pierce, Agile Tortoise

who am i?

- ▶ Greg Pierce, President, Agile Tortoise
- ▶ Software development and consulting
 - ▶ SAN member, working with Servoy since 2006.
 - ▶ Web, business software and ERP development for 15+ years.
 - ▶ Module development and mentoring.

where am i?

- ▶ Company: <http://www.agiletortoise.com>
- ▶ Blog: <http://greg.agiletortoise.com>
- ▶ Email: greg@agiletortoise.com
- ▶ Earth: North Texas, USA
- ▶ Now: Talk to me!

I am a geek.



why are we here?

- ▶ Interesting features of JavaScript.
- ▶ Details of JavaScript in Servoy.
- ▶ Practical uses of these features to make your code more elegant, readable and maintainable.
- ▶ *Too early to drink beer **
 - * *(subject to personal interpretation)*

Assumptions

- ▶ Basic familiarity with JavaScript.
- ▶ Know common programming terminology.
- ▶ Know basics of Servoy forms and methods.
- ▶ *Not scared of code!!!!*

JavaScript

- ▶ Internal scripting language of Servoy.
- ▶ Originally developed for browsers.
- ▶ Becoming a common embedded scripting language.
- ▶ Syntax derived from C, Java.

Dynamically typed

vs. Statically typed

- ▶ Variables and object properties can hold any data type.
- ▶ No type checking done prior to execution.
- ▶ “Duck typing”

See: <http://plpatterns.blogspot.com/2007/08/static-vs-dynamic-typing.html>

Interpreted

vs. Compiled

- ▶ No compiling
- ▶ Code is evaluated at runtime
- ▶ Enables “eval” of code, and live manipulation of the runtime environment
- ▶ “Monkey patching” or “Duck punching”

Functions

- ▶ A function is a block of callable code.
- ▶ All your code in Servoy resides in functions.
- ▶ “function” is a data type.
- ▶ “methods” are functions assigned to an object property.

function statements

```
function functionStatement()  
{  
    // function statement  
    // creates named function available locally...  
    function nestedFunction(value) {  
        return value+1;  
    }  
  
    // call using () operator...  
    nestedFunction(1);    // returns 2  
  
    // nestedFunction ceases to exist when parent returns  
}
```

function literals

```
function functionLiteral()
{
    // function literal
    // creates unnamed function, assigned to a var (or obj property)
    // scope only limited to life of the variable
    var literalFunction = function(value) {
        return value+1;
    }
    // call using () operator...
    literalFunction(1); // returns 2
    // optional name makes it callable from within itself
    var func2 = function funcName(value) {
        funcName(value+1); // recursive call
    }
    // literalFunction can live as long as the var it's
    // assigned to is in scope and can be passed as a parameter
}
```

arguments

- ▶ Any function can accept any number of arguments.
- ▶ Arguments are available via the array-like “arguments” object inside the function.
- ▶ Anything (including a function) can be passed as an argument.

arguments

```
function myFunction(first, second) {  
  first == arguments[0]; // true!  
  second == arguments[1]; // true!  
  var third = arguments[2] || "default value"; // optional?  
  arguments.callee; // reference to enclosing function  
  
  return arguments.length; // ? depends ?  
}
```

```
myFunction(); // returns 0  
myFunction(1,2); // returns 2  
myFunction(1,2,3); // returns 3
```

Scopes

- ▶ Scope == Context for variables and expressions.
- ▶ Scopes act like objects, accessed using 'this'
- ▶ Global scope created when interpreter launches (when solution opens in Servoy)
- ▶ “function” creates new inner scope
- ▶ No block-level scope (if, for, etc.)

Scopes

```
// interpreter launched...
```

```
var gGlobal = 1;
```

```
function f()
```

```
{
```

```
    var fLocal = 2;
```

```
    fGlobal = 1; // no 'var' means it's global
```

```
    var ttl = fLocal + gGlobal; // ttl = 3
```

```
    function g()
```

```
    {
```

```
        var gLocal = 3;
```

```
        var gTtl = gGlobal + fLocal + gLocal; // ttl = 6
```

```
        ttl = gTtl;
```

```
    }
```

```
    ttl = gLocal; // BOMB!!
```

```
}
```

```
gGlobal = gLocal; // BOMB!!
```

```
gGlobal = fLocal; // BOMB!!
```

Not “object-oriented”

- ▶ Objects, but no classes

```
typeof 'abc' == 'string';  
typeof 2.0 == 'number';  
typeof new Object() == 'object';  
typeof new Array() == 'object';  
typeof new Date() == 'object';  
typeof new Function() == 'function';
```

Not “object-oriented”

- ▶ Objects are treated as associative arrays

```
var obj = new Object();  
  
obj.prop1 = 'abc';  
obj['prop1'] = 'abc';  
  
var propName = 'prop1';  
obj[propName] = 'abc';
```

Constructors

- ▶ Constructor function can mimic class
- ▶ Constructor == function intended to initialize an Object called with 'new'

```
function Person(firstName, lastName)
{
  this.firstName = firstName;
  this.lastName = lastName;
}

var p = new Person("Joe", "Smith")
p.firstName; // "Joe"
```

Prototypes

- ▶ Constructor prototypes associate methods with objects.

```
Person.prototype.getFullName = function() {  
    return this.firstName + ' ' + this.lastName;  
}  
  
var p = new Person("Joe", "Smith")  
p.getFullName(); // "Joe Smith"
```

JavaScript in Servoy

- ▶ Rhino: Java based JavaScript interpreter
 - ▶ From Mozilla Foundation
 - ▶ Updated to 1.6R7 (1.5R3 in v3.5) in v4

What's new in v4

- ▶ functions are allowed in the editor
- ▶ E4X: Native XML
- ▶ 'apply' method

E4X: Native XML

- ▶ XML object and literal data type

```
var xml = new XML([xmlString]);
```

```
xml = <item id='1'>  
    <name>Juice</name>  
</item>;
```

E4X: object accessors

- ▶ Once you have an XML object, you can use object/array style accessors...

```
var products = <products>
  <item id='1'><name>Juice</name></item>
  <item id='2'><name>Wine</name></item>
</products>;
```

```
var firstName = products.item[0].name; // Juice
var secondId = products.item[1].@id // 2
```

'apply' method

- ▶ apply method used to forward arguments to another method...

```
function wrapper()  
{  
  // do something before  
  // don't call 'globals.my_function(arguments[0], etc.)'  
  // instead use 'apply'  
  globals.my_function.apply( this, arguments);  
  // do something after  
}
```

Servoy limitations

- ▶ ‘Sealed’ objects cannot be extended
 - ▶ foundset, application, security, etc.
- ▶ Common JavaScript examples and libraries rely on browser environment.

Examples

- ▶ Dynamic method calls
 - ▶ Testing if a form/method exists
 - ▶ Create a callback chain
- ▶ Wrap global methods
- ▶ *Use prototypes to extend Arrays*
- ▶ *Use 3rd party libraries: Date.js*
 - ▶ <http://www.datejs.com/>
- ▶ *Building form “types”*

form_exists

```
function form_exists()  
{  
  // make sure the form is loaded  
  eval( 'forms.'+arguments[0]);  
  // return true if it exists  
  return forms[arguments[0]] ? true : false;  
}
```

form_hasMethod

```
function form_hasMethod()
{
  var frmName = arguments[0];
  var methodName = arguments[1];

  // check if form exists
  if( !globals.form_exists(frmName) )
    return false;

  // return true if form has a property of type
  // 'function' with methodName
  return typeof( forms[frmName][methodName] ) == 'function';
}
```

event_newRecordCmd

```
function event_newRecordCmd()
{
    var frmName = application.getMethodTriggerFormName();
    var frm = forms[frmName];

    if(!security.canInsert(frm.controller.getServerName(), frm.controller.getTableName()))
    {
        plugins.dialogs.showErrorDialog( 'Error', "No way, man!", "OK");
        return false;
    }
    if( globals.form_hasMethod( frmName, 'canInsert' ) && !frm.canInsert() )
    {
        plugins.dialogs.showErrorDialog( 'Error', "No way, man!", "OK");
        return false;
    }

    if( globals.form_hasMethod( frmName, 'event_newRecordCmd' ) )
        result = frm.event_newRecordCmd();
    else
        result = frm.controller.newRecord(false,true);
    if( !result )
        return false;

    if( globals.form_hasMethod( frmName, 'event_initRecord' ) )
        frm.event_initRecord();

    return result;
}
```

wrap_global_method

```
function wrap_global_method()
{
    var orig = arguments[0];
    var before = arguments[1] || null;
    var after = arguments[2] || null;
    var newName = orig + '_orig';

    if(typeof globals[newName] == 'function')
        globals[orig] = globals[newName];
    globals[newName] = globals[orig];

    var repl = function() {
        switch(typeof before)
        {
            case 'string' : eval(before); break;
            case 'function' : before(); break;
        }
        var result = globals[newName].apply(this, arguments);
        switch(typeof after)
        {
            case 'string' : eval(after); break;
            case 'function' : after(); break;
        }
        return result;
    }
    globals[orig] = repl;
}
```

Warnings!

- ▶ Don't overdo it!
- ▶ Don't stomp on built-in objects!



Conclusion



Don't be limited by the SOM!

ServoyCasts

- ▶ Instructional screencasts
- ▶ <http://servoycasts.agiletortoise.com/>

Modules

- ▶ Started on several free modules
- ▶ All on Google code
- ▶ MIT license (free as in beer)
- ▶ Would love to have other contribute

mod_console

- ▶ Runtime JavaScript interpreter for testing/debugging.
- ▶ Intro ServoyCast available
- ▶ <http://code.google.com/p/servoymodconsole/>

mod_js_core

- ▶ Base for common JavaScript language extensions.
- ▶ Adapting useful features of “prototype.js” library for Servoy.
- ▶ Unit testing! (via scriptaculous)
- ▶ <http://code.google.com/p/servoymodjscore/>

mod_sql

- ▶ SQL builder for Servoy, lets you generate SQL with JavaScript syntax.
- ▶ <http://code.google.com/p/servoymodsql/>

```
var q = $sql.find('orders').where(['company_id = ?', c_id]);  
foundset.loadRecords(q, q.args());
```

mod_datejs



- ▶ Adaptation of the date.js library for Servoy.
- ▶ <http://code.google.com/p/servoymoddatejs/>

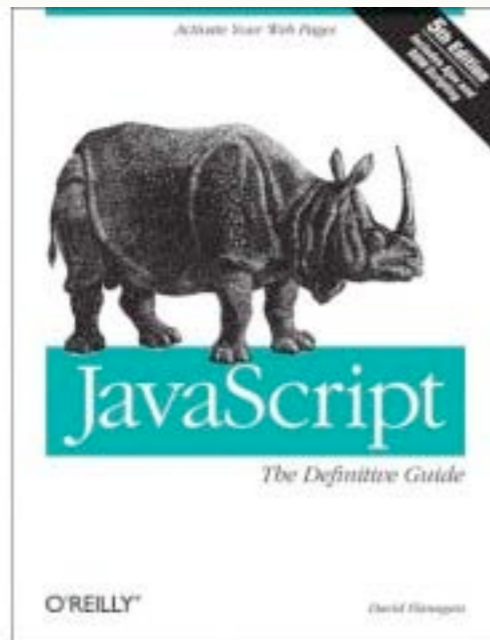
```
Date.parse("next friday");  
Date.parse("today");  
Date.today().next().thursday();  
  
(3).days().ago();  
(6).months().fromNow();
```

Q&A



References

- ▶ Wikipedia - <http://en.wikipedia.org/wiki/JavaScript>
- ▶ Rhino - <http://www.mozilla.org/rhino/>



- ▶ *JavaScript: The Definitive Guide*
- ▶ by David Flanagan