

Class Inheritance and Introspection

This document describes TIBCO General Interface™ extensions to the JavaScript class and inheritance model. With these extensions General Interface™ provides a richer programming environment that is better suited to building complex object-oriented applications.

Version 3.0. May 7, 2007

Scope: TIBCO General Interface 3.4



<http://www.tibco.com>

Global Headquarters

3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

© 2006-2007, TIBCO Software Inc. All rights reserved. TIBCO, the TIBCO logo, The Power of Now, and TIBCO Software are trademarks or registered trademarks of TIBCO Software Inc. in the United States and/or other countries. All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

Table of Contents

Classes and Inheritance in JavaScript 1.x	3
jsx3.lang.Object and Classes in TIBCO General Interface.....	3
Declaring a Class.....	4
Class, Interface, and Package.....	6
Introspection.....	7

Classes and Inheritance in JavaScript 1.x

JavaScript 1.x supports prototype inheritance. A class is defined by any function:

```
function Plant(latinName, englishName) {
    this.latinName = latinName;
    this.englishName = englishName;
}
```

The prototype field of the class function represents the prototypical instance of the class. A new instance of the class will be a copy of the prototype, including any fields and methods placed in the prototype.

```
Plant.prototype.relatedSpecies = new Array();

Plant.prototype.getLatinName = function() {
    return this.latinName;
};
```

Inheritance is supported by setting the prototype of a class function to a new instance of the superclass:

```
function Tree(latinName, englishName, flowering) {
    this.latinName = latinName;
    this.englishName = englishName;
    this.flowering = flowering;
};

Tree.prototype = new Plant();

Tree.prototype.isFlowering = function() {
    return this.flowering;
};
```

JavaScript supports an inheritance-aware `instanceof` operator. The following statements are true:

```
(var aPlant = new Plant()) instanceof Plant;
(var aTree = new Tree()) instanceof Plant;
(var aTree = new Tree()) instanceof Tree;
```

All classes implicitly extend the `Object` class so the following statement is also true:

```
(var aPlant = new Plant()) instanceof Object;
```

jsx3.lang.Object and Classes in TIBCO General Interface

General Interface defines its own base class, `jsx3.lang.Object`, which adds the following capabilities to the JavaScript 1.x model:

- Obtaining the class that defines an instance with the `getClass()` method.
- Calling an overridden method in a superclass with the `jsxsuper()` and `jsxsupermix()` methods.
- A better `toString()` method that overrides `Object.toString()` and includes the name of the class from which the object was instantiated.
- Introspection of class and interface membership using the `instanceOf()` method.

`jsx3.lang.Object` works in conjunction with the other classes in the `jsx3.lang` package – `Class`, `Exception`, `Method`, and `Package` – to provide these further advantages over plain old JavaScript 1.x:

- A rich exception model with intelligible stack traces
- Full introspection of classes and methods
- Mixin style interfaces
- Introspectable packages
- Simple declaration of classes and members using standard JavaScript 1.x syntax that frees the developer from the details of prototype inheritance

All classes in General Interface descend from `jsx3.lang.Object`. Developers are encouraged to extend the General Interface base class in application code in order to fully incorporate the features enumerated above.

Declaring a Class

Classes are defined with the static method `jsx3.lang.Class.defineClass()`:

```
static method defineClass(strName, objExtends, arrImplements, fctBody)
```

The first parameter, `strName`, is a string that is the fully-qualified name of the class to define, such as `"com.tibco.Widget"`. The `defineClass()` method ensures that the namespace object, `com.tibco`, exists before creating the `Widget` constructor.

The second parameter, `objExtends`, is the superclass of the class to define. If `null` is passed, `jsx3.lang.Object` is assumed. This parameter is usually provided as the constructor function of the superclass, i.e. the fully-qualified class name without quotes.

The third parameter, `arrImplements`, is an array of the interfaces that the class implements. This array can be empty or `null`.

The fourth parameter, `fctBody`, is a function that defines the contents of the class. The `defineClass()` method will execute this function exactly once after it has handled the details of the JavaScript 1.x prototype inheritance implementation. The `defineClass()` method passes two parameters to `fctBody`. The first parameter is the constructor function of the newly defined class. The second parameter is the prototype object of the class. This is simply a syntactical shortcut that allows for the following concise idiom:

```
jsx3.lang.Class.defineClass(  
    "com.tibco.Widget",  
    jsx3.lang.Object, // Object is implicit so this parameter could be null in this case  
    [jsx3.util.EventDispatcher],  
  
    function(Widget, Widget_prototype) {  
        // define class members in Widget  
        // define instance members in Widget_prototype  
    }  
);
```

There are four types of members that can be included in the class definition: static fields, static methods, instance fields, and instance methods. Each of these types is introspectable with methods of the `jsx3.lang.Class` class. Static members are declared by defining fields of the class constructor. Continuing the preceding code example:

```
// define two static fields
Widget.SERIAL = 1;
Widget.ALL_WIDGETS = new Array();

// define a static method
Widget.getWidgetBySerial = function(serial) {
    return Widget.ALL_WIDGETS[serial];
};
```

Static members are globally available using the fully-qualified name of the member, such as `com.tibco.Widget.getWidgetBySerial()`. The same member can be referenced with `Widget.getWidgetBySerial()` within the class declaration function because `com.tibco.Widget` is aliased as `Widget` inside the function. Unlike in Java, static members are not accessible through instances of a class (that is, `awidget.SERIAL` is undefined).

Instance members are declared by defining fields in the prototype field of the class constructor:

```
// define two instance fields
Widget.prototype.serial = null;
Widget.prototype.name = "?";

// define an instance method
Widget.prototype.getSerial = function() {
    return this.serial;
};
```

All classes must define a special instance method called `init()`. `defineClass()` throws an exception if no `init()` method is declared in the class declaration function. `init()` is effectively the constructor function. However, `defineClass()` actually creates the class constructor so the developer has no way of customizing it. Instead, the generated class constructor simply calls the `init()` method passing all the parameters that were passed to the constructor. Here is the `init()` method for the `Widget` class:

```
Widget.prototype.init = function(name) {
    this.name = name;
    this.serial = Widget.SERIAL++;
};
```

The following code will instantiate a `Widget`:

```
var widget = new com.tibco.Widget("myFirstWidget");
```

Class, Interface, and Package

The `jsx3.lang` package defines three class-like constructs – Class, Interface, and Package. These constructs are not exactly equivalent to their Java/C++ namesakes.

Classes:

- Define static fields, static methods, instance fields, instance methods, and abstract instance methods
- Are instantiated
- Can be the superclass of another class
- Descend from `jsx3.lang.Object`
- Are defined with `jsx3.lang.Class.defineClass()`

Interfaces:

- Define static fields, static methods, instance methods and abstract instance methods,
- Are not instantiated
- Can be the superclass of another interface
- Can be implemented by a class
- Are defined with `jsx3.lang.Class.defineInterface()`

Packages:

- Define static fields and static methods
- Contain classes and interfaces defined with `jsx3.lang.Class`
- Are defined with `jsx3.lang.Package.definePackage()`

Unlike in Java, General Interface interfaces may contain concrete instance methods. When a class implements an interface, the instance methods of the interface are mixed into the class. A mixed in method will override a method of the same name inherited from the superclass but will not override a method defined in the implementing class or mixed in from an interface coming earlier in the `arrImplements` parameter to `Class.defineClass()`.

Packages must be explicitly defined with `jsx3.lang.Package.definePackage()` in order to be introspectable. Simply defining the class `com.tibco.Widget` will not define the package `com.tibco`. Defining a package is not required for the class to work. It simply allows the package to be introspected.

Both classes, interfaces, and packages can "contain" classes/interfaces. Consider the following code assuming that the class `com.tibco.Widget.Encoding` is defined:

```
// returns [com.tibco.Widget, com.tibco.Widget.Encoding]
jsx3.lang.Package.forName("com.tibco").getClasses();

// returns [com.tibco.Widget.Encoding]
jsx3.lang.Class.forName("com.tibco.Widget").getClasses();
```

Introspection

Once a class, interface, or package has been defined as described above, it is introspectable. There are a few ways of getting a handle to an instance of `jsx3.lang.Class` or `jsx3.lang.Package`:

```
var aClass = anObject.getClass(); // returns an instance of jsx3.lang.Class
var aPackage = aClass.getPackage(); // returns an instance of jsx3.lang.Package
```

```
com.tibco.Widget.jsxclass; // instance of jsx3.lang.Class
com.tibco.jsxpackage; // instance of jsx3.lang.Package
```

```
// the following returns null if any part of the namespace is undefined:
jsx3.lang.Class.forName("com.tibco.Widget");
jsx3.lang.Package.forName("com.tibco");
```

Consult the API Documentation for all the methods of these classes that can be used to introspect their contents.