Implementing Business Logic

In This Chapter

- Overview
- Scope
- Creating a Variable
  - Create a Scope Variable (two ways)
  - Create a Form Variable (two ways)
- Creating a Method
  - Create a Scope Method in One of Two Ways
  - Create a Form Method in One of Two Ways
- Implementing Basic Business Logic
  - Insert a Code Fragment
  - Insert Sample Code

Overview

While the Servoy platform is based entirely in Java, one does not need to write **any** Java during the course of development or deployment.

Instead, all business logic is implemented using JavaScript. JavaScript was selected because it is an internet standard, easy to learn and as such, the most widely used scripting language on the planet. JavaScript is far more productive than coding in pure Java and Servoy provides robust APIs with which to quickly and easily implement business logic.

Note

Developers who are familiar with JavaScript may cite issues with browser support and speed of execution.

However, it is worth noting that Servoy does not deploy any JavaScript. All code written in Servoy is deployed using Mozilla's Rhino project, which is an open-source, Java-based JavaScript implementation.

This means that:

1. All methods are executing in Java (orders of magnitude faster than interpreted JavaScript)
2. No business logic is ever exposed or executed in the browser, thereby eliminating browser support issues.
3. Experienced developers can optionally use 3rd-party java APIs, mixing Java code directly in their Servoy methods.

Scope

Scope defines the domain in which code is executed and subsequently determines the namespace by which scripted objects are referenced. JavaScript code (functions and variables) may be defined in the following scopes:

- solution-wide scopes:

  Global Scope: Found in the globals.js file and accessible via the namespace globals, i.e.

  ```
  scopes.globals.createNewCustomer(); // invokes the global method defined in the 'globals' scope
  ```

- form scopes

  Form Scope: Found in the formName.js file and accessible via the namespace forms.formName, i.e.

  ```
  forms.customers.controller.newRecord();  // invokes a form object from another scope
  ```

compared to:

```
function createNewCustomer(){  // a method defined within the 'customers' form scope

    controller.newRecord();    // invokes the same form object from within the form's scope. Notice the fully
qualified namespace is NOT required

}
```

Creating a Variable

Create a Scope Variable (two ways)

From context menu:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Scopes** > **myScope** > **variables** node.
2. Right-click the **variables** node and select **Create Variable** from the pop-up menu.
3. Choose a variable name, a data type and optionally choose an initial value. The variable declaration will be generated in the **myScope.js** file, which will be opened in the Script Editor.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Scopes** > **myScope** > **variables** node.
2. Select the **variables** node and click the **Create Variable** button from the lower toolbar in the Solution Explorer.
3. Choose a variable name, a data type and optionally choose an initial value. The variable declaration will be generated in the **myScope.js** file, which will be opened in the Script Editor.

Create a Form Variable (two ways)

From context menu:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Forms** > **myForm** > **variables** node.
2. Right-click the **variables** node and select **Create Variable** from the pop-up menu.
3. Choose a variable name, a data type and optionally choose an initial value. The variable declaration will be generated in the **myForm.js** file, which will be opened in the Script Editor.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Forms** > **myForm** > **variables** node.
2. Select the **variables** node and click the **Create Variable** button from the lower toolbar in the Solution Explorer.
3. Choose a variable name, a data type and optionally choose an initial value. The variable declaration will be generated in the **myForm.js** file, which will be opened in the Script Editor.

Creating a Method

Create a Scope Method in One of Two Ways

From context menu:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Scopes** > **myScope** node.
2. Right-click the **myScope** node and select **Create Method** from the pop-up menu.
3. Choose a method name. The method declaration will be generated in the **myScope.js** file, which will be opened in the Script Editor.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Scopes** > **myScope** node.
2. Select the **myScope** node and click the **Create Method** button from the lower toolbar in the Solution Explorer.
3. Choose a method name. The method declaration will be generated in the **myScope.js** file, which will be opened in the Script Editor.

Create a Form Method in One of Two Ways

From context menu:

1. From theSolution Explorertree, navigate to the **active solution** > **Forms** > **myForm** node.
2. Right-click themyFormnode and selectCreate Methodfrom the pop-up menu.
3. Choose a method name. The method declaration will be generated in themyForm.jsfile, which will be opened in the Script Editor.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to the **active solution** > **Forms** > **myForm** node.
2. Select the **myForm** node and click the **Create Method** button from the lower toolbar in the Solution Explorer.
3. Choose a method name. The method declaration will be generated in the **myForm.js** file, which will be opened in the Script Editor.

Implementing Basic Business Logic

To implement some business logic, create a method and fill in the body of the JavaScript function with executable code.

The following example implements the functionality to advance the selected record index on a form:

```
function nextRecord(){

    var index = controller.getSelectedIndex(); // store the current index

    controller.setSelectedIndex(index+1);      // increment the index by 1

}
```

The example uses the form's **controller** object, part of the JavaScript API provided by Servoy.

Developers need not memorize the API or look it up. The scripting APIs are self documenting, and code fragments can easily be inserted into the Script Editor.

Insert a Code Fragment

Any scripting API (including methods written by a developer) can be inserted directly into the Script Editor in two ways.

From context menu:

1. From the **Solution Explorer** tree, navigate to and select the node of a scriptable resource, (i.e. **active solution** > **Forms** > **myForm** > **controller**).
2. In the list of methods and properties provided in the lower part of the **Solution Explorer**, right-click the method or property wished to be invoked and select **Move Code**. The code will be copied into the Script Editor and will be referenced with the correct namespace for the current scope. It may be needed to fill in specific arguments.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to and select the node of a scriptable resource.
2. In the list of methods and properties provided in the lower part of the **Solution Explorer**, select the method or property wished to be invoked and click the **Move Code** button in the lower toolbar of the **Solution Explorer.** The code will be copied into the Script Editor and will be referenced with the correct namespace for the current scope. It may be needed to fill in specific arguments.

Insert Sample Code

Any scripting API's commented sample code can be inserted directly into the Script Editor in two ways.

From context menu:

1. From the **Solution Explorer** tree, navigate to and select the node of a scriptable resource, (i.e. **active solution** > **Forms** > **myForm** > **controller**).
2. In the list of methods and properties provided in the lower part of the **Solution Explorer**, right-click the method or property wished to be invoked and select **Move Sample**. A verbose, commented sample will be copied into the Script Editor.

From Solution Explorer toolbar:

1. From the **Solution Explorer** tree, navigate to and select the node of a scriptable resource.
2. In the list of methods and properties provided in the lower part of the **Solution Explorer**, select the method or property wished to be invoked and click the **Move Sample** button in the lower toolbar of the **Solution Explorer.** A verbose, commented sample will be copied into the Script Editor.

Tip

Any scripting API (including methods written by a developer) can be auto-completed using key strokes.

To use **auto-complete**, begin typing the code to be executed, then hit **Ctrl+space** on the keyboard. A type-ahead list of available scripting objects will appear allowing for choosing from them while typing. This is a highly productive and accurate way to write code when one has become more familiar with the APIs.

Tip

For developers not so familiar with JavaScript, there is an easier way to use the JavaScript API. In the **Solution Explorer**, navigate to the **JS Lib** node. This node contains a list of APIs for dealing with the JavaScript language and native data types. It even contains syntax completion for common statements in the **Statements** node.