Web Application Development Using TCL/Apache Rivet and JavaScript

Anne-Leslie Dean, Senior Software Developer, FlightAware

Presented: 25th Annual TCL/Tk Conference, Oct 15-19, 2018

Introduction

Modern Web applications rival native applications in user experience. This is accomplished by manipulating the Web document dynamically in the client browser environment. There are a many choices of scripting (language) environments a developer may choose to use on the web server: PHP, Python, Ruby, TCL etc. But on the client side, the JavaScript environment is the standard for programmatic control of the Web application.

Apache Rivet provides a server environment to build Web applications using TCL. This case study will demonstrate the application of fundamental software design principles such as separation of concerns, abstraction, and encapsulation in a TCL/Apache Rivet environment with integrated JavaScript.

The basic structure of the Rivet script will distinguish *content management* from *presentation* and will visibly identify a clear interface between the TCL/Apache Rivet and the JavaScript environments.

Each step of the case study will conclude with an analysis of the software development principles being applied.

Case Study Step 1: Framing Out The Rivet Script

The distinguishing architectural feature of modern Web applications boils down to a simple model: a document template rendered against an abstracted data model. The code concerned with managing the abstracted data model is commonly referred to as *content management*. The code concerned with describing the document template is commonly referred to as *presentation*. The basic structure of an Apache Rivet script should always reinforce this Web application architectural distinction. This can be accomplished by applying the software principle of *separation of concerns* with respect to *content management* and *presentation*.

Figure 1 demonstrates generating a simple web page using Apache Rivet and TCL. The TCL code section is focused on content management, while the composition of HTML markup template for presentation is separated into a secondary section.

Figure 1:

```
figure1.rvt:
<?
    # Use Apache Rivet "load_response"
    # to expose the web page request arguments in the TCL environment.
    ::rivet::load_response

    # CONTENT MANAGEMENT
    # Select the "n"th example and load it into the
    # abstracted "content" array for rendering according
    # to the HTML template which follows.
    set examples(1) {
        title "Example 1"
        description "This is the first example"
    }
    set examples(2) {
        title "Example 2"
        description "This is the second example"
    }
    if { [info exists response(n)] } {
        array set content $examples($response(n))
    } else {
        array set content $examples(1)
    }
?>

<!--
PRESENTATION
The HTML document template
-->
<html>
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3>Description</h3>
    <p>
        <?=$content(description)?>
    </p>

</body>
</html>
```

Case Study Step 1: Analysis

Keeping the content management code visually separate from the presentation (HTML) markup allows the developer to compose the document template in markup that is almost pure HTML. The HTML markup section of code is focused on a single responsibility of sorts, that of presentation.

Leveraging Rivet directives rather than embedding TCL code to integrate content with the document markup keeps the code clean and as uncluttered as possible.

Thus,

```
<h1><?=$headerVariable?></h1>
```

is preferable to

```
<? puts "<h1>$headerVariable</h1>" ?>
```

Sometimes it may be expedient to embed TCL in addition to Rivet directives in the Rivet (HTML) markup. For example the markup in Figure1 could be altered to output an HTML table using TCL to loop through the `examples` array.

Figure 1b:

```
<table>
<tr><th>Title</th><th>Description</th></tr>
<!-- Embed some TCL to loop through the examples -->
<? foreach {num case} [array get examples] {
    array set content $case
?>
  <tr><td><?=$content(title)?></td><td><?=$content(description)?></td></tr>
<? }?>
</table>
```

However, the developer should always keep in mind the distinction between content management versus presentation. When introducing TCL into the Rivet (HTML) markup section the developer should question whether the code being written is the responsibility of content management or presentation logic. It the answer is, "Content management," then the developer should refactor the TCL code into the content management portion of the Rivet script.

Case Study Step 2: Adding JavaScript

In the TCL/Apache Rivet stack, TCL/Apache Rivet is used to render HTML in the Apache server environment. JavaScript is introduced solely to drive dynamic document behaviors within the browser environment. Figure 2 introduces JavaScript into our case study implementing the ability to dynamically change the *presentation* of the description.

Figure 2:

```
figure2.rvt:
…Content management code unchanged…
<html>
<head>
```

```
    <style>
        #plus-minus-toggle {
            color: red;
            cursor: pointer;
        }
        #plus-minus-toggle:before {
            content: "-";
        }
        #plus-minus-toggle.collapsed:before {
            content: "+";
        }
        #description.collapsed {
            display: none;
        }
    </style>
</head>
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3 id="plus-minus-toggle" class="collapsed" >Description</h3>
    <p id="description" class="collapsed" >
        <?=$content(description)?>
    </p>

    <script type="application/javascript">
        // CONFIGURE DYNAMIC DOCUMENT BEHAVIOR
        // A simple JavaScript interaction is wired up which
        // toggles the visibility of the "description"
        // in response to user clicks.
        var toggle = document.getElementById("plus-minus-toggle");
        toggle.onclick = function(event) {
            var desc = document.getElementById("description");
            if ( this.className === "collapsed" ) {
                this.className = "";
            } else {
                this.className = "collapsed";
            }
            desc.className = this.className;
        };
    </script>
</body>
</html>
```

Case Study Step 2: Analysis

When adding JavaScript into the document the developer should keep in mind that JavaScript is only executed in the browser client environment. Keeping the JavaScript contained in the context of the HTML document markup and free of too much embedded TCL helps the developer keep a conceptual separation of concerns between

the parts of script controlling backend behavior versus script that is controlling frontend behavior in the Web application.

Case Study Step 3: Changing Content Using JavaScript

JavaScript can be used to dynamically modify both *presentation* and *content* of a Web application in the browser client environment.

In both figures 1 and 2, the web page content is statically rendered when the page is served. In figure 2, JavaScript is simply used to alter the presentation of already rendered content. Figure 3 demonstrates how JavaScript may also be used to dynamically alter the document content.

Figure 3 adds a jQuery* UI Tooltip widget to the document description toggle. The Tooltip widget enhances the document presentation by showing a tool tip when the user hovers the mouse over the description toggle. When the user clicks on the toggle, the JavaScript onclick handler alters the document content of the tooltip.

*jQuery is a representative third-party JavaScript library that can be incorporated into the TCL/Apache Rivet/JavaScript stack. jQuery is chosen for having wide audience literacy. In practice, many Web application developers are replacing jQuery in preference for newer libraries and frameworks.

Figure 3:

```
figure3.rvt:
<?
…
    # Add tooltip content
    set content(tipCollapse) "Click to hide description"
    set content(tipExpand) "Click to expand description"
…
?>
<html>
<head>
…
    <link rel="stylesheet" href="jquery-ui-1.12.1.custom/jquery-ui.min.css">
…
</head>
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3 id="plus-minus-toggle" class="collapsed" title="">Description</h3>
    <p id="description" class="collapsed" >
        <?=$content(description)?>
    </p>
```

```
    <script src="jquery-3.3.1.min.js"></script>
    <script src="jquery-ui-1.12.1.custom/jquery-ui.min.js"></script>
    <script type="application/javascript">
        // CONFIGURE DYNAMIC DOCUMENT BEHAVIOR
        // A simple JavaScript interaction is wired up which
        // toggles the visibility of the "description"
        // in response to user clicks.
        var toggle = document.getElementById("plus-minus-toggle");
        $("#plus-minus-toggle").tooltip({ content: '<?=$content(tipExpand)?>'
} );
        toggle.onclick = function(event) {
            var desc = document.getElementById("description");
            if ( this.className === "collapsed" ) {
                this.className = "";
                $("#plus-minus-
toggle").tooltip("option","content","<?=$content(tipCollapse)?>");
                $("#plus-minus-toggle").tooltip("close");
            } else {
                this.className = "collapsed";
                $("#plus-minus-
toggle").tooltip("option","content","<?=$content(tipExpand)?>");
                $("#plus-minus-toggle").tooltip("close");
            }
            desc.className = this.className;
        };
    </script>
</body>
</html>
```

Case Study Step 3: Analysis

Dynamically altering document content requires increased manipulation of the HTML document object model (DOM) in the browser client environment. Using a third party JavaScript library or framework, like jQuery, is hugely useful in hiding the complexity of interaction with the DOM through encapsulation of native JavaScript DOM functionality as well as managing compatibility of JavaScript DOM interaction across different browsers.

The JavaScript code is generated as an extension of the HTML document by embedding literal JavaScript between the HTML `<script></script>` tags. This mechanism requires no refinement of the interface between the Rivet HTML template and the TCL content management logic. The only difference from the previous step is that Rivet directives are used to embed the value of TCL variables into JavaScript in addition to HTML markup.

Case Study Step 4: Encapsulating JavaScript into a Separate File

In the examples shown in figures 1-3, the transfer of data from the TCL portion of the script to the JavaScript portion of the script is accomplished by using the Rivet directives to inject the value of TCL variables directly into the HTML markup and dynamically generated JavaScript. In order to simplify the complexity of the Rivet script (and for performance considerations), a Web developer will commonly encapsulate Web application JavaScript into a stand-alone JavaScript file.

This means that the use of Rivet directives to transfer values from TCL into JavaScript need to be refined. This may be accomplished by setting the value of a variable in the JavaScript global namespace that will then be accessible in subsequently executed JavaScript files. Figure 4 shows the setting of the `tipContent` variable in the JavaScript global namespace using TCL/Rivet directives, that will then be accessed by the JavaScript in `application_fig4.js`.

Figure 4:

```
figure4.rvt:
…
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3 id="plus-minus-toggle" class="collapsed" title="">Description</h3>
    <p id="description" class="collapsed" >
        <?=$content(description)?>
    </p>

    <script src="jquery-3.3.1.min.js"></script>
    <script src="jquery-ui-1.12.1.custom/jquery-ui.min.js"></script>
    <script type="application/javascript">
        var tipContent = {
            'tipExpand' : '<?=$content(tipExpand)?>',
            'tipCollapse' : '<?=$content(tipCollapse)?>'
        };
    </script>
    <script src="application_fig4.js"></script>
</body>
…

application_fig4.js:
// CONFIGURE DYNAMIC DOCUMENT BEHAVIOR
// A simple JavaScript interaction is wired up which
// toggles the visibility of the "description"
// in response to user clicks.
var toggle = document.getElementById("plus-minus-toggle");
$("#plus-minus-toggle").tooltip({ content: tipContent.tipExpand } );
toggle.onclick = function(event) {
    var desc = document.getElementById("description");
    if ( this.className === "collapsed" ) {
        this.className = "";
```

```
        $("#plus-minus-
toggle").tooltip("option","content",tipContent.tipCollapse);
        $("#plus-minus-toggle").tooltip("close");
    } else {
        this.className = "collapsed";
        $("#plus-minus-
toggle").tooltip("option","content",tipContent.tipExand);
        $("#plus-minus-toggle").tooltip("close");
    }
    desc.className = this.className;
}
```

Case Study Step 4: Analysis

The approach in Figure 4 simplifies the structure of the Rivet script by encapsulating the complexity of the JavaScript initialization in a separate file. This is a recommended practice to both declutter the HTML markup as well as allowing optimization of JavaScript code through various minimization techniques.

However, the use of a global variable in this way should give any developer pause, and the scenario in Figure 4 is no exception. The mechanism of transferring data from the TCL content management part of the code is not clearly visible to the developer when reading the code. Order of execution dependencies between multiple JavaScript files are masked. Debugging and maintenance tasks are complicated.

Case Study Step 5: Use Modular JavaScript

A more refined solution is to leverage a modular JavaScript approach. Figure 5 avoids using the JavaScript global namespace by wrapping the application initialization code in an anonymous function. The code also clearly documents the flow of information across the boundary between TCL/Rivet and the JavaScript environments by passing `tipContent` as an argument to the `initializeApplication` function rather than relying on passing `tipContent` as a global variable as in Figure 4.

Figure 5:

```
Figure5.rvt
…
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3 id="plus-minus-toggle" class="collapsed" title="">Description</h3>
    <p id="description" class="collapsed" >
        <?=$content(description)?>
    </p>
```

```
    <script src="jquery-3.3.1.min.js"></script>
    <script src="jquery-ui-1.12.1.custom/jquery-ui.min.js"></script>
    <script src="application_fig5.js"></script>
    <script type="application/javascript">
        (function() {
            var tipContent = {
                'tipExpand' : '<?=$content(tipExpand)?>',
                'tipCollapse' : '<?=$content(tipCollapse)?>'
            };
            initializeApplication(tipContent);
        })();
    </script>
</body>
…
application_fig5.js
// CONFIGURE DYNAMIC DOCUMENT BEHAVIOR
// A simple JavaScript interaction is wired up which
// toggles the visibility of the "description"
// in response to user clicks.
var initializeApplication = function( tipContent ) {
    var toggle = document.getElementById("plus-minus-toggle");
    $("#plus-minus-toggle").tooltip({ content: tipContent.tipExpand } );
    toggle.onclick = function(event) {
        var desc = document.getElementById("description");
        if ( this.className === "collapsed" ) {
            this.className = "";
            $("#plus-minus-
toggle").tooltip("option","content",tipContent.tipCollapse);
            $("#plus-minus-toggle").tooltip("close");
        } else {
            this.className = "collapsed";
            $("#plus-minus-
toggle").tooltip("option","content",tipContent.tipExand);
            $("#plus-minus-toggle").tooltip("close");
        }
        desc.className = this.className;
    };
}
```

Case Study Step 5: Analysis

Using a modular approach to encapsulation of JavaScript leads to several benefits. The interface of passing data between the TCL/Apache Rivet and JavaScript environment(s) is clearly visible when reading the code. This improves the ability of a developer to debug and maintain the code. Through articulating a clear interface, the modular approach lends itself to both testability and reusability as well.

The function, `intializeApplication`, could be refined to include parameter validation and further enhance the robustness of ensuring the correct behavior of the Web application.

Case Study Step 6: Standardize The TCL To JavaScript Interface Using JSON

A further refinement to the passing of TCL values into the JavaScript environment would be to leverage JSON to bundle up values to be passed into the JavaScript environment. Figure 6 illustrates the use of a TCL proc to encapsulate the JSON representation of `tipContent`.

Figure 6:

```
figure6.rvt:
…
    # Proc to generage tooltip JSON
    # replaces:
    # set content(tipCollapse) "Click to hide description"
    # set content(tipExpand) "Click to expand description"
    proc tipContentJSON { } {
        package require yajltcl
        set content [::yajl create #auto]
        $content map_open
        $content map_key tipCollapse string "Click to hide description"
        $content map_key tipExpand string "Click to expand description"
        $content map_close
        set json [$content get]
        $content free
        return $json
    }
…
<body>
    <!-- Embed the abstracted content using the Rivet <?= ?> directive -->
    <h2><?=$content(title)?></h2>
    <h3 id="plus-minus-toggle" class="collapsed" title="">Description</h3>
    <p id="description" class="collapsed" >
        <?=$content(description)?>
    </p>

    <script src="jquery-3.3.1.min.js"></script>
    <script src="jquery-ui-1.12.1.custom/jquery-ui.min.js"></script>
    <script src="application_fig5.js"></script>
    <script type="application/javascript">
        (function() {
            var tipContent = <?=[tipContentJSON]?>;
            initializeApplication(tipContent);
        })();
    </script>
</body>
…
```

Case Study Step 6: Analysis

Encapsulation of abstract content components using TCL procs such as `tipContentJSON` makes the components of the content management interface testable and reusable. Using JSON as a data exchange format between TCL and JavaScript is supported by various add-on packages in TCL (yajl-tcl in the example) and is seamlessly supported in JavaScript.

Establishing JSON as a standard data exchange format creates the potential for the content management layer to be refined into a content management service. For Web applications a content management service can be used to update the web page content JavaScript/Ajax requests. The same content management service endpoint might be further reused to supply content to other media delivery channels such as mobile device applications.

Leveraging the same content management service endpoint across multiple media channels can reap several business level benefits. Consolidation of business logic into a single environment ensures consistency of content regardless of the presentation medium. As media delivery channels come and go, the presentation layer of software can be refactored or replaced, while the content management layer can be refined and reused.

Conclusion

The case study offers a blueprint for a Web developer to implement well-structured code using TCL/Apache Rivet and JavaScript. The example code illustrates application of software design principles that produce code that structured for maintainability, testability and reuse.