

1 Background

BrightSword Technologies has built many web applications for customers worldwide. We have always endeavored to stay on the cutting edge of the industry, and when BrightSword Designer was launched in 2003, it contained several innovations that embedded within.

One of the problems that we have faced is the issue of being able to modularize a web-page into so that we could easily compose a page from its constituents at will. We tried many approaches to achieve this – the code generated by BrightSword Designer in 2003 had some pretty clever JavaScript that allowed for Zero-Glue™ interaction between page components, and we got an IFRAME-based solution to cleverly rotate quotes by our senior staff on our web-page.

When we researched the problem in some detail, we discovered (as have many others), that *the bulk of the pain and suffering experienced whilst using traditional web application platforms such as ASP.NET and PHP, stemmed from the fact that the page was one huge HTML form, with a will and mind of its own, and any component participating in the page needed to be known by, and intimately know of, the containing form.*

The emergence of AJAX as a Web Application Development Methodology vindicated several thoughts that have been floating around the company since its beginning. The generic grid for our very first project in 2000 was unmistakably AJAXian in spirit, if not in manifestation!

We discovered (as have many others) that the solution to freely composed web pages is to have independent portions of the page speak to the web server independently, and that a “RESTful” approach to POST and GET resources was a key prerequisite to this approach.

Therefore, with the influence of Jim Baker, a friend and colleague, who showed that it is possible and feasible to write high-performance RESTful web servers, we took up the challenge and developed the BrightSword Reminisce™ Web Server.

We obviously cite Dr. Roy Fielding for his seminal PhD dissertation for REST.

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

Roy Fielding¹

Further credit goes to Dr. Shriram Krishnamurthy who, in his casual discussions, threw up so much food for thought that I'm still gorging!

The implementation of a RESTful web server is difficult in that certain fundamental mind-blocks relating to what the structure of a web-server and web-application should be have to be systematically addressed and removed.

The novel approach taken by BrightSword is that we have brought to bear our expertise in code-generation to develop an elegant solution. We believe that our solution most effectively addresses a niggling problem in building high performance web applications, especially on the .NET platform.

¹ <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

2 Synopsis

The BrightSword Reminisce™ Web Server is a RESTful web interface to an object store with preliminary support for interaction workflows.

3 Technical Discussion

3.1 Summary of the RESTful approach

The RESTful approach, very simply put, is that every object in a system is given a unique resource identifier (URI), and operated on using the primitives of the HTTP protocol specification.

This means that with a RESTful web server, an object is mapped to an URL, and operations on the object can be mapped to the HTTP verbs of GET, POST and DELETE.

Our implementation of a RESTful server uses a superset of the following convention to deal with objects and sets of objects.

- A GET request on URL http://server/app/<object_type>/ returns an XML document containing the set of all known objects of type <object_type>. A GET request does not change *anything* on the server.
- A GET request on URL http://server/app/<object_type>/<object_guid>/ returns an XML document containing the single object of type <object_type> identified by <object_guid>. A GET request does not change *anything* on the server.
- A POST request on URL http://server/app/<object_type>/<object_guid>/ creates or updates the object of type <object_type> identified by <object_type>. A successful POST request necessarily changes some state on the server. There is no return value, and a POST request terminates with a redirect to GET the same URL.
- A DELETE request on URL http://server/app/<object_type>/<object_guid>/ deletes the object of type <object_type> identified by <object_type>. Since the URL ceases to refer to a valid object, a DELETE request terminates with a redirect to GET the URL of the object_type.

3.2 The Reminisce™URL Scheme

Since the primary interface to a web server is through an URL, it is of great importance to codify the structure of a well-formed URL so that a RESTful server can uniformly process and serve objects in a predictable manner.

3.2.1 Sample URLs

The following table schematically describes the various portions of the URL and query string that are examined and processed to control the behaviour of the web application.

It is important to note that the query string arguments specified are all orthogonal. They can be used in any combination and they give the result that one intuitively expects. For example, any combination of any number of 'filter=' and 'sort=' can be used, along with specifications of 'widget=' and 'render='.

Other operations are definitely possible. The key thing to recognize is that the scheme is uniform and predictable and independent of the object type. This leads to a simple interface to interact with the server.

URL	GET	POST	DELETE
http://server/app/object	Returns all instances of type <object> If Baker extensions are disabled, return XML. If Baker extensions are enabled, return the HTML generated by the Grid widget.	Creates a new instance of type <object> if ID is specified in the payload. Forbidden otherwise.	Deletes all instances of type <object>. Can be disabled if so desired.
http://server/app/object/id/	Returns the single instance of type <object> with ID = <id>. This could be a newly created “volatile” object if there is no record with ID = <id> in the backend store. If Baker extensions are disabled, return XML. If Baker extensions are enabled, return the HTML generated by the ReadOnlyForm widget.	Creates a new instance of type <object> with ID=<id> if none exists in the backend store. Otherwise updates the existing records.	Marks the instance of type <object> with ID=<id> as inactive if records exist. Does nothing if no such records exist.
http://server/app/object?render=json 'render' is one of {json, xml, html}. Other values are ignored. Default is 'xml' if Baker extensions are disabled, 'html' if Baker extensions are enabled) The last specification of 'render' is the one used if multiple 'render' arguments are specified	Returns all instances of type <object> as JSON	The entire query string is ignored, so functionality is identical with that of the plain URL.	The entire query string is ignored, so functionality is identical with that of the plain URL.

URL	GET	POST	DELETE
<p>http://server/app/object?filter={column,value.op}&render=...</p> <p>Column is the name of the column to filter on. No quotes are required and the name is case-insensitive.</p> <p>Value is the value to filter for. No quotes are required. Type is inferred from the column name.</p> <p>Op is one of a set of binary operators supported by the backend store. Type-specific operators like 'like' are supported if the backend supports.</p> <p>Multiple specifications of 'filter' are allowed and are conjoined with the 'AND' operator.</p>	<p>Returns all matching instances of type <object> in the rendering format specified</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>
<p>http://server/app/object?sort={column,direction}&filter...</p> <p>Column is the name of the column to sort on. No quotes are required and the name is case-insensitive.</p> <p>Direction is the one of {asc, desc} and specifies the sorting direction.</p> <p>Multiple specifications of 'sort' are allowed and are treated in the order specified</p> <p>The operation is forbidden for those types that do not support sorting (like BLOBs)</p>	<p>Returns all matching instances of type <object> in the rendering format specified in the order desired.</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>
<p>http://server/app/object?top={10}&sort=...</p> <p>The number specified in the top argument limits the resultant object-set to those many instances.</p>	<p>Returns the first n matching instances of type <object> in the rendering format specified in the order desired.</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>	<p>The entire query string is ignored, so functionality is identical with that of the plain URL.</p>

URL	GET	POST	DELETE
http://server/app/object?widget=foo&top=... See the section on Baker Transforms below.	Returns the first <i>n</i> matching instances of type <object> in the rendering format specified in the order desired, applying the 'foo' widget transform to generate HTML	The entire query string is ignored, so functionality is identical with that of the plain URL.	The entire query string is ignored, so functionality is identical with that of the plain URL.
http://server/app/object/Form	Returns an empty FORM. When posted back to the URL, creates a new object.	Creates a new object as specified by the FORM.	Forbidden
http://server/app/object/id/Form	Returns a FORM pre-filled with the object specified. When posted back to the URL, modifies the specified object.	Modifies the object specified by the URL with values specified by the FORM.	Forbidden
http://server/app/object/Metadata	Returns a JSON object describing the properties of the object class. <i>This should be changed to return whatever render=... asks for.</i>	Forbidden	Forbidden
[PROPOSED] http://server/app/object/LastModifiedTime Many other such URLs and options are possible...	Returns the last time the object table was touched – for caching purposes	Forbidden	Forbidden

3.3 How REST is implemented in Reminisce™

The discussion on the Olive pattern (Part 2 of the LongReach™ White Paper) is recommended prerequisite reading for this document.

In summary, the “pickled object” pattern represents database tables as classes and rows as instances of those classes. The mechanism that queries the database for objects that satisfy a certain criterion is also codified in the olive Pattern.

The Reminisce™ Web Server is implemented to run on IIS, and require no support from any of the ASP.NET scripting engines, and is sufficiently modular as to port itself to any other HTTP-server such as Apache with *mod_rewrite*.

The details of URL parsing and delegation of calls to the various factory methods is outside the scope of this document².

3.3.1 GET

GET translates itself into the GetObjects() factory method call on the specified object. The filter and sort arguments are appropriately parsed and type-cleaned, and converted into palatable SQL by the appropriate StorageAdapters, and passed into the stored procedure for execution.

Also, in the case of MSSQL, the render argument is examined to see if the FastXMLLoader is appropriate. In any case, the appropriately carried row-reader function is passed into the DataReader iterator to render the database results to the desired format.

3.3.2 POST

POST translates itself into the Persist() call on the object that is loaded from the posted form by the FormLoader class. Since there is no ASP.NET request processing, we use whatever is provided to us by the System.Web.HttpRequest class.

3.3.3 DELETE

DELETE translates itself into the Delete() call on the object that is specified on the URL, loaded by the SqlServerObjectLoader class. Subsequent GET calls to the specified URL will return a 404.

3.4 What about HTML?

It's a reasonable expectation that a web server serve up web pages in HTML.

Since it is a design goal that the presentation layer of the Web Application be fully disconnected from the other layers, we are in a dilemma about how to generate HTML (a presentation layer artifact) from the RESTful Object Server (something that only delivers objects, and really shouldn't be bothered with details on what columns go into the form in which order, and certainly about what colors should be used).

Enter the Baker Transform.

3.4.1 The Baker Transform

This transform is named in honor of our friend and colleague Jim Baker, who first outlined the idea.

² This is not an understatement. There is significant complexity involved in implementing this server, and describing it would take a complete document which should be forthcoming shortly.

When the Baker Extensions are enabled in the Reminisce Server, the request for 'render=html' calls the same methods as does 'render=xml', and processes the resultant XML with an XSLT transform that converts the XML to HTML.

What is interesting about this XSLT is that *it is itself generated* at application-development time by a tool known as the Baker Transform.

Because it's not reasonable to ask a Graphics Designer (or even most developers) to write XSLT, we developed the concept of a widget – snippets of HTML which represent the boilerplate of the HTML, along with placeholders where data can be injected from the object XML – and the Baker transform, which is an XSLT that consumes the widget definition and generates the XSLT which is used by the web server to transform the object XML into HTML.

What this means is that any number of faces (widgets) can be designed and applied much like masks on the object or object set, transforming it into various forms of HTML which can be further adorned using unobtrusive JavaScript and CSS.

In addition, using JavaScript frameworks like Dojo prove to be extremely amenable to this approach, as it allows the UI developer an impressive arsenal of tools (simple template-based layout specification, CSS, JavaScript and Dojo widgets) to be able to build compelling AJAXian applications.

3.4.2 One Level Higher: widgen³ - the 'uber-Baker' Transform

We know that all objects require Forms and Grids to create, edit and show them off.

When we want provide a RESTful interface to dozens of objects in a project, it is definitely drudgery to have to write the dozens of Grid and Form widgets. Since any change in the object model would require a corresponding change in the widgets, the management of these widgets would be drudgery upon drudgery.

So we did what any self-respecting code-generation-engine developer would do – we wrote a code-generator to generate the widget definitions themselves.

widgen.exe takes the input class definition file and a few transforms to generate all the default Baker widgets for a given system⁴. A developer can then modify or create extra widgets to suit requirements

4 Description of Suite Components

³ Well, we finally ran out of cool names! I wanted to name this the Raman transform, in honor of my friend, mentor and erstwhile boss, but since I didn't get his permission, we settled for 'widgen' – which is a somewhat less inspiring contraction of 'widget generator'.

⁴ The astute reader will recognize this as an example of *xslt transforming xml to xml which is then transformed by xslt to xslt, which then transforms xml to html*. Debugging this comes alarmingly close to black magic - accompanied as it is by a lot of muttering, mumbling and command line invocations!

4.1 widgen.exe

When the LongReach™ suite is installed, the MSI installs widgen.exe and the widget generation transforms.

'widgen.exe' takes the class definition file consumed by olive and generates the default Grid and Form widget templates for each of the classes.

```
C:\Work\svn\projects\PJO\MTW\Olive>widgen.exe --help
*****
=== widgen.exe - Generates Widget Template Definitions for all widget template patterns and all classes in all xml files specified.
=== Usage: widgen.exe --help --verbose --xml=<value> --xmlfile=<value> --match=<value> --widget=<value> --output=<value>
=== --help          : [Optional] Prints this message and quits
=== --verbose       : [Optional] Emits more information
=== --xml=<value>   : [Optional] Specifies the directory in which the class definition xml files are stored. If not specified, uses the 'xml'
directory.
===                 --xml has a default value of [xml]
===                 The effective value of --xml is [xml]
=== --xmlfile=<value> : [Optional] Specifies a single xml file to generate widgets for. If not specified, generates widgets for all xml files in
the directory specified by --xml
===                 --xmlfile has a default value of []
===                 The effective value of --xmlfile is []
=== --match=<value>  : [Optional] If --xmlfile is specified, specifies a single xml node in the xml file to generate the widget for.
===                 --match has a default value of []
===                 The effective value of --match is []
=== --widget=<value> : [Optional] Specifies the single widget template to generate for. If not specified, generates widgets for all templates i
n the directory specified by --templates.
===                 --widget has a default value of []
===                 The effective value of --widget is []
=== --output=<value> : [Optional] Specifies the directory in which the output files are stored. If not specified, uses the 'output' directory.
===                 --output has a default value of [output]
===                 The effective value of --output is [output]
=== Please ensure that there is no space except between arguments. Quote values that embed a space in them. Use '--arg=value' and not '--arg = value'
=== Exiting...
*****
```

Figure 1: Widgen --help - generates widget templates for a project

4.2 baker.exe

When the LongReach™ suite is installed, the MSI installs baker.exe and the Baker transform.

'baker.exe' converts a widget template into an XSLT transform which can be used by Reminisce™ to convert object XML into HTML.

```
C:\work\svn\projects\PJO\MTW\Olive>baker --help
*****
===
=== baker - Generates Baker Transforms for all widget templates and all classes in all xml files specified.
===
=== Usage: baker --help --verbose --widgets=<value> --widget=<value> --output=<value>
===
=== --help          : [Optional] Prints this message and quits
===
=== --verbose       : [Optional] Emits more information
===
=== --widgets=<value> : [Optional] Specifies the directory in which the widget definition xml files are stored,
===                   --widgets has a default value of [widgets]
===                   The effective value of --widgets is [widgets]
===
=== --widget=<value>  : [Optional] Specifies the single widget template to generate for. If not specified, generates transforms for all widgets
===                   in the directory specified by --widgets.
===                   --widget has a default value of []
===                   The effective value of --widget is []
===
=== --output=<value>  : [Optional] Specifies the directory in which the output files are stored.
===                   --output has a default value of [output]
===                   The effective value of --output is [output]
===
=== Please ensure that there is no space except between arguments. Quote values that embed a space in them. Use '--arg=value' and not '--arg = value'
===
=== Exiting...
===
*****
```

Figure 2: baker --help - The Baker transform

5 Summary

We have implemented a completely RESTful Web Server for data-driven applications.

We have successfully separated out the presentation layer for this web server from the 'API' of the object server, so that the server can serve raw objects in XML and JSON formats. The presentation layer is highly customizable with the introduction of the Baker Widget concept, delegating presentation layer glitz to the browser where it belongs – with CSS and unobtrusive JavaScript.

We have incorporated some functionality from Dojo into our default Widgets so we can display compelling default Forms and Grids.

Details of how the Reminisce™ Web Server supports Azure, a “remote-object” pattern, can be found in Part 3 of the LongReach™ white paper.

Workflow support present in the Reminisce™ Server will be discussed in a forthcoming document.