## Using MiServer to Provide RESTful Web Services

A MiSite can be configured to provide RESTful web services. This document describes the steps necessary to do so. While we will cover some RESTful concepts, this is not tutorial on RESTful web services themselves – for a good overview of RESTful web services, please see <http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>.

### Web Service Technologies

A web service is a method of communication between two processes over a network. A web service does not have a GUI interface, though GUI interfaces to web services are common. For instance, a web page that prompts for a postal code may use a web service for validation, but it's the web page that provides the GUI, not the web service.

There are currently two predominant web service technologies – SOAP-based and REST. SOAP-based web services use XML-based protocols for message exchange. Dyalog provides support for SOAP-based web services with SAWS, the Stand Alone Web Services framework. REST (Representational State Transfer) web services have become popular for providing public APIs. REST is an architectural style, unlike SOAP which is a standardized protocol. REST makes use of HTTP, and does not create any new standards. It can structure data into XML, YAML, or any other machine readable format, but usually JSON – JavaScript Object Notation – is preferred. REST is very data-driven, compared to SOAP, which is strongly function-driven. A web service that uses REST is termed "RESTful".

### The 5 Minute Oversimplified Guide to REST

This section is intended to give you a very basic understanding of REST. REST implements a verb-noun style of interaction.

#### Verbs

The verbs in REST are the set of HTTP commands. If you're familiar with web site technology, you probably already know about the HTTP commands **GET** and **POST**. The HTTP protocol has several other commands that are often used for RESTful web services. The first four verbs are typically used to implement CRUD – the operations CREATE, READ, UPDATE, and DELETE.

|  |  |
| --- | --- |
| **Verb (HTTP Command)** | **Common RESTful Use** |
| **GET** | READ – retrieve a resource |
| **POST** | UPDATE – update an existing resource |
| **PUT** | CREATE – create a new resource |
| **DELETE** | DELETE – delete a resource |
| **HEAD** | check if a resource exists |
| **OPTIONS** | query what verbs are available for a resource |

#### Nouns

The nouns in REST are resources and are accessed via URIs (Uniform Resource Identifiers).

A resource is some collection of data. It could be a list of customers, information about a specific customer, a list of orders for a specific customer, the order details for a specific order for a specific customer,... you get the idea.

A URI is simply a string of characters to identify a resource. You're probably most familiar with URLs, which is a type of URI that identifies a web address - typically a file somewhere on the web. A URI doesn't necessarily have to point to an actual physical resource like a file, it merely identifies it. For instance, the URI <http://someWebService.com/customer/123> might identify the customer with id 123 though there may not be an actual file for customer 123.

### A Few Observations

* REST is very flexible and you have the power to implement (or not) whatever set of verbs upon   
  whatever resources make sense for your application. For instance:  
  **POST** <http://myservice/Persons/> might add a new record to the Persons table in your application, but  
  **POST** <http://myservice/> may not make any sense in the context of your application.
* REST is inherently stateless which means that each request is independent. This can pose some design challenges.
* REST itself does not address issues of authentication – these are typically handled by HTTP or some external authentication mechanism. MiServer and REST

### Configuration

In your MiSite configuration folder, create or update the following entries in Server.xml.

* Set RESTful to 1 – this is important so that MiServer knows how to interpret URIs.   
  For example, the URI <http://myservice/Persons/Brian>   
  RESTful≠1 – look for the file /Persons/Brian

RESTful=1 – look for the file /Persons and supply "Brian" as a parameter

* Set AllowedHttpCommands to a comma-delimited string of the verb you wish to support in your web service.

<Server>  
 <RESTful>1</RESTful>  
 <AllowedHttpCommands>get,put,delete,post</AllowedHttpCommands>  
</Server>

**It is strongly recommended that you update /Config/Server.xml file in your MiSite folder and not at the MiServer level.**

### RESTful Page Requirements

#### Use the RESTful base class for your pages

Instead of using **MiPage**, or some other base class, you will use **RESTful** as the base class for your RESTful pages.

:Class myService : **RESTful**    
:EndClass

#### Use Respond instead of Compose

MiServer expects a public method named **Respond** to return the results for your web service.   
**Respond** must be niladic and return a result.

∇ r←Respond  
 :Access public  
 ⍝ your code here  
 ∇

### Features of the RESTful Base Class

The RESTful base class shares many features of the MiPage class.

* data passed with the HTTP request, either via query parameters in the URI, or in the message body are stored in \_PageData and are accessible with the GET method.
* \_Command returns the lowercased HTTP command
* \_URI returns a vector of vectors containing the segments of the URI  
  e.g. for the URI /customer/123/order, \_URI will contain

\_URI

┌────────┬───┬─────┐

│customer│123│Order│

└────────┴───┴─────┘  
Note: the first element is the name of the your RESTful page.

* SetStatus will set the HTTP status code (for more detail about HTTP status codes than you'll ever want to see look at: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> ).  
  Note: Status 200 (OK) is set by default, you only need to set the status when something goes wrong, like a resource not being found (status 404).
* SetHeader is used to set HTTP headers in the response.  
  MiServer takes care of setting most headers like content-length and content-type for you.

### Examples

#### Simple Read Only Loan Calculation Service

The page below implements a loan calculation web service.  
The parameters are provided in the query string of the URI.  
The results are stored in a namespace, which MiServer converts into a JSON object.   
  
:Class mortgagews : RESTful   
⍝ calculate payment   
 calcpmt←{0::'Error' ⋄ p r n←⍵÷1 1200 (÷12) ⋄ .01×⌈100×p×r÷1-(1+r)\*-n}  
⍝ calculate principle  
 calcprin←{0::'Error' ⋄ r n m←⍵÷1200 (÷12) 1 ⋄ .01×⌈100×m÷r÷1-(1+r)\*-n}  
  
 ∇ response←Respond;mask;r ⍝ render the initial page  
 :Access Public   
 r←⎕NS ''  
 r.msg←'Please provide either (prin rate term) or (rate term pmt)'  
 :if 1=+/mask←⍬∘≡¨r.(prin rate term pmt)←⍬ Get 'prin rate term pmt'  
 :if mask[1] ⍝ principle missing  
 r.(msg prin)←'' (calcprin r.(rate term pmt))  
 :elseif mask[4] ⍝ payment missing  
 r.(msg pmt)←'' (calcpmt r.(prin rate term))  
 :endif  
 :endif  
 response←1 #.JSON.fromAPL r  
 ∇  
:EndClass

Request: <http://localhost:8080/mortgagews?prin=100000&rate=.05&term=30>

Response: {msg:"",pmt:279.88,prin:100000,rate:0.05,term:30}

#### More Comprehensive Web Service

Still working on this... I'm not happy with what I have at present... expect to finish on Sunday

:Class customer : RESTful  
 ∇ r←Respond;custid;urlParams  
 :Access public  
 r←''  
 :Select \_Command ⍝ which HTTP command?  
  
 :Case 'post' ⍝ create  
 :If ~0∊⍴name←Get'name' ⍝ was a name supplied?  
 r←#.bl.AddCustomer name  
 :Else ⋄ SetStatus 400 'Invalid format' ⋄ :EndIf  
  
 :Case 'get' ⍝ retrieve  
 :If 0∊⍴\_URI ⍝ if no information passed in the URI  
 r←#.bl.GetCustomer Get'id' ⍝ retrieve all customers  
 :Else  
 :Select ⊃⍴\_URI  
 :Case 1  
 r←#.bl.GetCustomer 1⊃\_URI  
 :CaseList 2 3 ⍝ either 'order' or 'order' ordered  
 :If 'order'≡2⊃\_URI  
 r←⊃#.bl.GetCustomerOrder/1 0 1/3↑\_URI,⊂''  
 :Else ⋄ SetStatus 400 'Invalid format' ⋄ :EndIf  
 :EndSelect  
 :EndIf  
  
 :Case 'put' ⍝ update  
 :If ~0∊⍴name←Get'name' ⍝ was a name supplied?  
 :AndIf 1=⍴\_URI ⍝ make sure id was passed in URL  
 r←(1⊃\_URI)#.bl.UpdateCustomer name  
 :EndIf  
  
 :Case 'delete' ⍝ delete  
 :If 1=⍴\_URI ⍝ make sure id was passed in URL  
 :AndIf 0∊⍴GetNames'' ⍝ and nothing in the query string  
 r←#.bl.DeleteCustomer 1⊃\_URI  
 :EndIf  
  
 :Else ⍝ invalid command  
 SetStatus 400 ⍝ Bad Request  
 :EndSelect  
  
 :If 0∊⍴r  
 SetStatus 404 ⍝ Not Found  
 :EndIf   
 ∇  
:EndClass