3.x - URL Paths and Web Services

Let's go into a little more detail regarding the information introduced in the URL Navigation article. It is important to understand how the URL (and the individual segments within the URL) translate into specific web service routines within the SRP HTTP Framework. Below is the same URL graphic that we have already seen:

Let's use this sample URL and break it down so we can properly understand how this fits within the SRP HTTP Framework processing flow:

Segment	Description
`https://`	This is the communication protocol. It will either be HTTP or HTTPS. While this information is available to the SRP HTTP Framework, it is rare that your API code will use it.
`www.myapplication.com`	This is the domain (aka Home URL). While this information is available to the SRP HTTP Framework, it is rare that your API code will use it. However, this *normally* should match the value in the Home URL configuration so your responses can include well-formed links that the client can use to navigate further.
`/api`	This is the API URL. Technically, this isn't a complete URL. The complete URL would be a combination of the protocol, Home URL and the API URL (e.g., `https://www.myapplication.com/api`). This defines where the API entry point begins. Normally, the API URL is not the same as the Home URL so that users can enter into the website (e.g., `https://www.myapplication.com`) and be greeted with a home page rather than immediately invoke API logic. Normally, users never enter the API URL. This is handled behind the scenes via JavaScript or some other server-side scripting logic (PHP, ASP.Net, or even a proxy server routing requests from a customer facing server into separate API server.) Regardless of what the API URL is, when this is the address, the request will pass through the HTTP_MCP controller and into the entry point web service. The entry point will always be executed regardless of the full URL used for the request. **This is the first, and most important, step of walking the URL path.** The entry point is responsible for authentication and it is responsible for allowing the rest of the URL path to be processed.
`/customers`	This segment is associated with the *customer* (or *customers*) resource type. Remember, RESTful URLs are to be treated as *nouns* (resources). Any segment that is associated with a resource type will normally be its own web service and have its own BASIC+ stored procedure designed to handle this resource. The SRP HTTP Framework is designed to automatically assume that any web service will be named like this: `HTTP_<URLSegment>_Services`. For this example, the web service routine would be named `HTTP_Customers_Services`. **Therefore, any web service you create should follow this naming pattern.** If you already know the name of a resource type you intend to use (e.g., `/invoices`, `/vendors`, `/parts`, etc.), then you may want to create a basic web service shell first (e.g., `HTTP_Invoices_Services`, `HTTP_Vendors_Services`, `HTTP_Parts_Services`, etc.). Using one of the sample service routines that are included (HTTP_Users_Services or HTTP_Contacts_Services) can help you get a web service working fairly quickly, especially if your web service is tied to a single database table.
`/5678`	Intuitively we see that this segment represents a specific customer of interest. Because *5678* represents a specific resource, rather than a resource type, there will not be a specific web service to handle this segment. **Therefore, it is the responsibility of the preceding web service (`HTTP_Customers_Services` in this example) to look ahead at the *RemainingURL* argument to see if there is a customer ID being passed in.** If there is one, then the web service will attempt to retrieve this resource and prepare a response containing details for this customer. If there isn't one, then the web service will normally assume that all of the customers (typically referred to as a *collection*) are being requested. This does not necessarily mean that every detail of every customer needs to be returned. The normal practice is to return a list of customer IDs, names, and URLs needed to easily access each specific customer.
`/phone`	This segment is associated with the *phone* (number) for the customer. The previous segments give us the broader context: Customer #5678. Now, *phone* is certainly a type of resource. That is to say, there are different types of phones (work, fax, mobile), so it is certainly reasonable for this segment to be managed by its own web service (e.g., `HTTP_Phone_Services`). However, this doesn't have to be implemented this way. Since *phone* is clearly related to the customer, this could also be handled by the `HTTP_Customers_Services` routine. In this case, *phone* is *less* of a resource, and *more* of a field being request. That is `/customers/5678` would return all of the customer details, but `/customers/5678/phone` would limit the information to the phone numbers. From an API point of view, it doesn't really matter how this is managed. The response will be the same and the client's URL would remain the same.
`/work`	Following our discussion about the *phone* segment, the *work* segment represents a specific type of phone number being requested. Therefore, it will never have its own web service. Instead, this will either be used by the *customers* web service or the *phone* web service, depending on which service was used to handle the *phone* segment in the first place. This is why the *RemainingURL* argument is very important. It gives the current web service logic a way to analyze the remaining segments and then decide how the rest of the URL should be processed. **/work represents the end point of the URL.** This is important as the end point determines what the final response should look like.

So, the above URL could follow one of two logical paths within the SRP HTTP Framework:

1. HTTP Request > OECGI > HTTP_MCP > HTTP_Entry_Point_Services > HTTP_Customers_Services > HTTP_Phone_Services > HTTP Response

   - or -

2. HTTP Request > OECGI > HTTP_MCP > HTTP_Entry_Point_Services > HTTP_Customers_Services > HTTP Response

In the first case, `HTTP_Customers_Services` is only responsible for validating the customer ID, and then it passes the API request through to `HTTP_Phone_Services`. In the second case, `HTTP_Customers_Services` validates the customer ID *and* it handles the phone number, which is the expected response. If there are no other segments that will be supported past this end point, then it may prove to be easier to use the second case. This keeps all customer related data managed under the same web service routine.