# 3.x - Pass Throughs and End Points

Web service routines will either serve as ***pass throughs*** to other web services or they will serve as ***end points***. As we observed in the 3.x - URL Paths and Web Services article, the URL itself usually makes this self-evident. But how does the web service code differ between the two operations?

## Pass Through Logic

One of the security barriers imposed by the SRP HTTP Framework is the requirement that preceding web services must allow the request to be passed through to the subsequent web services. This prevents any specific web service from just arbitrarily being tacked onto the end of a URL in hopes that it will force the SRP HTTP Framework to call it.

This design is very similar to the way MFS routines are stacked. That is, the first MFS listed in the stack must make sure the next MFS is called or it won't be implemented. The SRP HTTP Framework provides a specific service, *RunHTTPService*, to handle this pass through. The first instance where the *RunHTTPService* is used in in HTTP_MCP. Here is the relevant code:

```
EntryPointService   = HTTP_Services('GetEntryPointService')
RemainingURL        = HTTP_Services('GetHTTPPathInfo')
HTTP_Services('RunHTTPService', EntryPointService, RemainingURL)
```

The above code relies upon the *GetEntryPointService* and the *GetHTTPPathInfo* service to launch the entry point web service. The default entry point service (HTTP_Entry_Point_Services) provides a nice template for how a web service will process a typical pass through to a subsequent web service. Look at this snippet from the router block:

```
Case NextSegment _EQC 'users'
    // NextSegment contains the name of another service. The RemainingURL must be updated to show that the
    // NextSegment is not longer a part of the RemainingURL. Then the RunHTTPService service should be called.
    RemainingURL = Field(RemainingURL, '/', 2, 99)
    HTTP_Services('RunHTTPService', NextSegment, RemainingURL)

Case NextSegment _EQC 'contacts'
    // NextSegment contains the name of another service. The RemainingURL must be updated to show that the
    // NextSegment is not longer a part of the RemainingURL. Then the RunHTTPService service should be called.
    RemainingURL = Field(RemainingURL, '/', 2, 99)
    HTTP_Services('RunHTTPService', NextSegment, RemainingURL)
```

The *NextSegment* variable is pre-assigned for us via the HTTP_Service_Setup insert. So, if the next segment in the URL actually is the next web service, then this does not need to be changed. The *RemainingURL* variable comes from the incoming parameter of the current routine, so it will need to be updated before passing it onto the next web service (again, similar to the way MFS routines remove themselves from the stack before moving on).

If the *NextSegment* variable contains a resource ID, then it will need to be updated with the name of the next web service segment. As the developer, you can can populate this in whatever way makes sense, but most likely you will pull this out of the *RemainingURL* variable. Consequently, the *RemainingURL* will be updated accordingly.

## End Point Logic

The end point of any URL defines the nature of the response being requested. This is typically a collection (e.g., `/customers`) or a specific resource (e.g., `/customers/5678`). The nature of the action is defined by the HTTP method used to make the request.

While the end point will almost always be one of the two types of URLs demonstrated the above paragraph, there can also be *query params* which might be used to qualify (or filter) a resource collection (e.g., `/customers?name="acme"`). This can create a third option that might be handled in the web service. Here is what the router block might look like:

```
Case Len(NextSegment)
    // NextSegment contains the Item ID for the current service/resource. This means the URL ends with
    // /customers/<ItemId>. The client is requesting a specific customer item.
    SelfURL = HTTP_Services('GetSelfURL')

    Begin Case
        Case HTTPMethod _EQC 'GET'      ;   GoSub GetItem
        Case HTTPMethod _EQC 'OPTIONS'  ;   GoSub OptionsItem
        Case HTTPMethod _EQC 'POST'     ;   GoSub PostItem
        Case HTTPMethod _EQC 'DELETE'   ;   GoSub DeleteItem
        Case Otherwise$                 ;   ValidMethod = False$
    End Case

Case HasGetString
    // This means the URL ends with /customers?name=<string>. The client is searching for matching contact
items.
    SelfURL = HTTP_Services('GetSelfURL')

    Begin Case
        Case HTTPMethod _EQC 'GET'      ;   GoSub GetSearch
        Case HTTPMethod _EQC 'OPTIONS'  ;   GoSub OptionsSearch
        Case Otherwise$                 ;   ValidMethod = False$
    End Case

Case RemainingURL _EQC ''
    // This means the URL ends with /customers, which means this is the end point. The client is requesting a
    // collection of all customers.
    SelfURL = HTTP_Services('GetSelfURL')

    Begin Case
        Case HTTPMethod _EQC 'GET'      ;   GoSub Get
        Case HTTPMethod _EQC 'OPTIONS'  ;   GoSub Options
        Case HTTPMethod _EQC 'POST'     ;   GoSub Post
        Case Otherwise$                 ;   ValidMethod = False$
    End Case
```

Each condition in the above code immediately validates the HTTP method. This is another control mechanism. Each end point configuration will allow only specific HTTP methods to be processed. Otherwise, the web service will set a 405 (*Method Not Allowed*) status.

RESTful APIs make full use of the HTTP method to determine how to process the request. For simple web services where the end point is tied to a single database table, the HTTP_Users_Services and the HTTP_Contacts_Services routines provide a good starting point and should be easy to adapt for your own purposes.