# How do I add hypermedia to a resource?

On its own, information represented in resource objects is simply *media*. However, in order to encourage client exploration and access to related resources we need to **add *hypermedia* to our resources**. RESTful API developers often abandon hypermedia (the "H" in *HATEOAS)* implementation because it can be considered more effort than it is worth. The SRP HTTP Framework makes this much easier through the use of automation and dedicated services.


- Using the AddLinkRelationship Service
- Using the AddEmbeddedResources Service
- Using the GetDatabaseItems Service
- Using the AddFormAction Service


## Using the *AddLinkRelationship* Service

In our *How do I create a resource?* article we introduced the *GetObject* service as a primary method for creating resource objects. To keep our documentation focused on creating a resource objects, we did not introduce hypermedia. Now we will use the same code sample and introduce a new service: ***AddLinkRelationship***. The *AddLinkRelationship* service creates the HAL ***_links*** reserved property:

```
API customers.ID.GET

    KeyID           = EndpointSegment

    ColumnNames     = 'FIRST_NAME' : @FM : 'LAST_NAME' : @FM : 'ADDRESS' : @FM : 'CITY' : @FM : 'STATE' : @FM :
'ZIP'
    PropertyNames   = 'firstName' : @FM : 'lastName' : @FM : 'address' : @FM : 'city' : @FM : 'state' : @FM :
'zipCode'
    // Create a JSON object in memory.
    objResource     = HTTP_Resource_Services('GetObject', 'CUSTOMERS', KeyID, ColumnNames, PropertyNames)

    If Error_Services('NoError') then
        // Add _links sub-properties for HAL implementation.
        HTTP_Resource_Services('AddLinkRelation', objResource, 'self', FullEndpointURL)
        HTTP_Resource_Services('AddLinkRelation', objResource, 'collection', ParentURL)
    end

    If Error_Services('NoError') then
        // Serialize the JSON object.
        jsonResource    = HTTP_Resource_Services('GetSerializedResource', objResource)
        // Set the response body with the serialized JSON object and set the Content-Type response header.
        HTTP_Services('SetResponseBody', jsonResource, False$, 'application/hal+json')
    end else
        // There is an error condition so call the SetResponseError service.
        HTTP_Services('SetResponseError', '', '', 500, Error_Services('GetMessage'), FullEndpointURL)
    end

end api
```

By using the *FullEndpointURL* and the *ParentURL* prepopulated variables, we automatically have the *self* and *collection* relations ready to add to the *_links* property:

```
{
   "address":"6649 N Blue Gum St",
   "city":"New Orleans",
   "firstName":"James",
   "lastName":"Butt",
   "state":"LA",
   "zipCode":"70116",
   "_links":{
      "self":{
         "href":"https://www.examples.org/api/customers/1"
      },
      "collection":{
         "href":"https://www.examples.org/api/customers"
      }
   }
}
```

There is also a ***AddLinkRelations*** service. It's a wrapper around the *AddLinkRelation* service that allows the developer to pass in an @FM delimited list of relations and URLs into a single call. Developers will need to decide if the *AddLinkRelations* service provides enough convenience in lieu of multiple *AddLinkRelation* calls.

## Using the *AddEmbeddedResources* Service

HAL also supports the ***_embedded*** reserved property, which is a way to include partial or whole resources within the primary resource object. These embedded resources are themselves stand-alone resources with their own endpoints. The usual reason they are embedded is so the client does not have to make multiple requests to get this same resource information. This is common with collection APIs (e.g., `GET /customers`). The following code sample uses the *GetObjects* service to return an array of *customers* resource objects. These are then embedded into the primary resource object using the *AddEmbeddedResources* service:

```
API customers.GET

    // Create the primary resource object.
    objResource     = HTTP_Resource_Services('GetObject')
    If Error_Services('NoError') then
        // Create the sub-resource objects.
        objSubResources = HTTP_Resource_Services('GetObjects', 'CUSTOMERS', '', 'FIRST_NAME' : @FM :
'LAST_NAME', 'firstName' : @FM : 'lastName', '', '', '', FullEndpointURL)

        If Error_Services('NoError') then
            // Pass the sub-resource object list to the AddEmbeddedResources service. Use the 'customers' label
to identify the type of sub-resource.
            HTTP_Resource_Services('AddEmbeddedResources', objResource, 'customers', objSubResources)
        end

        If Error_Services('NoError') then
            // Serialize the JSON object.
            jsonResource    = HTTP_Resource_Services('GetSerializedResource', objResource)
            // Set the response body with the serialized JSON object and set the Content-Type response header.
            HTTP_Services('SetResponseBody', jsonResource, False$, 'application/hal+json')
        end else
            // There is an error condition so call the SetResponseError service.
            HTTP_Services('SetResponseError', '', '', 500, Error_Services('GetMessage'), FullEndpointURL)
        end
    end

end api
```

This API would return the following resource object:

```
{
   "_embedded":{
      "customers":[
         {
            "firstName":"Harrison",
            "lastName":"Haufler",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/266"
               }
            }
         },
         {
            "firstName":"Haydee",
            "lastName":"Denooyer",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/271"
               }
            }
         },
         {
            "firstName":"Heike",
            "lastName":"Berganza",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/254"
```

```
                }
            }
        },
        {
            "firstName":"Helga",
            "lastName":"Fredicks",
            "_links":{
                "self":{
                    "href":"https://www.examples.org/api/customers/202"
                }
            }
        },
        {
            "firstName":"Herman",
            "lastName":"Demesa",
            "_links":{
                "self":{
                    "href":"https://www.examples.org/api/customers/125"
                }
            }
        },
        {
            "firstName":"Herminia",
            "lastName":"Nicolozakes",
            "_links":{
                "self":{
                    "href":"https://www.examples.org/api/customers/287"
                }
            }
        },
        {
            "firstName":"Hillary",
            "lastName":"Skulski",
            "_links":{
                "self":{
                    "href":"https://www.examples.org/api/customers/186"
                }
            }
        },
        {
            "firstName":"Howard",
            "lastName":"Paulas",
            "_links":{
                "self":{
                    "href":"https://www.examples.org/api/customers/132"
                }
            }
        }
    ]
  }
}
```


## Using the *GetDatabaseItems* Service

We previously introduced the *GetDatabaseItem* service. There is a similar service called ***GetDatabaseItems*** and it is also a member of the HTTP_Resource_Services module. Like the *GetDatabaseItem* service, the *GetDatabaseItems* service is deprecated but we do not intend to remove it. For certain use cases it is quite handy for producing collection resources with embedded content:

```
API customers.GET

    jsonResource    = HTTP_Resource_Services('GetDatabaseItems', '', 'CUSTOMERS', FullEndpointURL, 'FIRST_NAME'
: @FM : 'LAST_NAME', 'firstName' : @FM : 'lastName')

    If Error_Services('NoError') then
        // Set the response body with the serialized JSON object and set the Content-Type response header.
        HTTP_Services('SetResponseBody', jsonResource, False$, 'application/hal+json')
    end else
        // There is an error condition so call the SetResponseError service.
        HTTP_Services('SetResponseError', '', '', 500, Error_Services('GetMessage'), FullEndpointURL)
    end

end api
```

This approach doesn't provide a way to customize the name of the embedded resource (e.g., *customers*, *vendors*, etc.). Instead, it just defaults to *item*. Here is an example of the resource object:


```
{
   "_embedded":{
      "item":[
         {
            "firstName":"Harrison",
            "lastName":"Haufler",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/266"
               }
            }
         },
         {
            "firstName":"Haydee",
            "lastName":"Denooyer",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/271"
               }
            }
         },
         {
            "firstName":"Heike",
            "lastName":"Berganza",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/254"
               }
            }
         },
         {
            "firstName":"Helga",
            "lastName":"Fredicks",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/202"
               }
            }
         },
         {
            "firstName":"Herman",
            "lastName":"Demesa",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/125"
               }
            }
         },
         {
            "firstName":"Herminia",
```

```
            "lastName":"Nicolozakes",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/287"
               }
            }
         },
         {
            "firstName":"Hillary",
            "lastName":"Skulski",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/186"
               }
            }
         },
         {
            "firstName":"Howard",
            "lastName":"Paulas",
            "_links":{
               "self":{
                  "href":"https://www.examples.org/api/customers/132"
               }
            }
         }
      ]
   },
   "_links":{
      "self":{
         "href":"https://www.examples.org/api/customers"
      }
   }
}
```

## Using the *AddFormAction* Service

The services described above add hypermedia in the form of URI relations using the reserved properties from the HAL specification (specifically *_links* and *_embedded*). While URI relations provide valuable hypermedia to the client, many developers have expressed a desire for additional metadata to better instruct clients how to call the URI. For example, a URI by itself doesn't inform the client which HTTP methods are supported. Granted, the *OPTIONS* meth od is intended to provide this information, but other requirements might necessitate out-of-band knowledge. This causes our APIs to lose their self-documenting character.

Mike Kelly, the author of the HAL specification, comments that omitting additional metadata in the HAL specification was "intentional" in order to keep it "focused on linking". However, he also suggests that "HAL is therefore a good candidate for use as a base media type on which to build more complex capabilities". This has spurred developers of RESTful APIs to shore up this gap through alternative *hypermedia* types (such as api+json, collection+json, hy per+json, and siren+json). Unfortunately, none of them have the maturity and broad acceptance that HAL does. Also, some of them over complicate the task of adding simple hypermedia to a resource. HAL, on the other hand, is relatively lightweight and easy to implement and consume.

Therefore, the SRP HTTP Framework uses the HAL specification as its base media type and the *AddLinkRelationship*, *AddEmbeddedResources*, and *Get DatabaseItems* services produce HAL compliant hypermedia. However, in those cases when a developer really wants to provide extra metadata to the client, we have provided the ***AddFormAction*** service. The *AddFormAction* service adds a custom hypermedia structure inspired by the work Ben Greenberg and his team did for the Comcast Xfinity APIs (you can watch his presentation on this here). To better explain the *AddFormAction* service, we have a dedicated article called *How do I add hypermedia controls to a resource?*.