SHARE San Jose 2017

Using JSON and COBOL for RESTful Services on the Web

Tom Ross ‘Captain COBOL’
SHARE San Jose
March 8, 2017

JSON and COBOL

- What is JSON?
- IBM products support JSON!
- Scenarios

What is JSON?

- JavaScript Object Notation
- JSON is the new XML
  - Lighter weight
  - Simpler
  - More compact
- JSON serves the same purpose as XML
  - Carries data for Services on the Web
    - Typically RESTful services
  - "Web Services" often means SOAP and XML
  - RESTful services are on the Web and often use JSON

JSON vs XML

COBOL:

```
01 G.
    05 h.
       10 a pic x(10)    Value ‘Eh?’.
       10 3_ pic 9         Value 5.
       10 C-c pic x(10) Value ‘See’.
```

JSON:

```
{“G": {“h": {“a": "Eh?", "3_": 5, “C-c": "See"}}}
```

XML:

```
<G><h><a>Eh</a><3_>5</3_><C_c>See</C_c></h></G>
```

What is JSON?

```
01 request.
   05 request-type          Pic X(10).
   05 client-age            Pic 9(3).
   05 smoker                Pic X.
   05 quote-response.
      10 quote-status       Pic X(7).
      10 quote-amount       Pic 9(9).
```

{“request": {“request-type":”QUOTE”} ,{“client-age":58},
{“smoker”:”N”},
{“quote-response“:{“quote-status”:”SUCCESS”},
{“quote-amount”:167}}}

Good News!
IBM products support JSON!

- z/OS Client Web Enablement Toolkit
- z/OS Connect V2
- Enterprise COBOL V6.1
- Enterprise PL/I V4.5
- The CICS JSON assistant
- IMS Mobile Solution
- WebSphere Application Server
- Java EE 6

Bad News? Too many IBM product choices for JSON support?

- z/OS Client Web Enablement Toolkit
- z/OS Connect V2
- Enterprise COBOL V6.1
- Enterprise PL/I V4.5
- The CICS JSON assistant
- IMS Mobile Solution
- WebSphere Application Server
- Java EE 6

IBM products support JSON!

- Enterprise COBOL V6.1
  - JSON GENERATE
- Enterprise PL/I V4.5
  - Parsing and generation of JSON texts
- The CICS JSON assistant
  - Independent of z/OS Connect
- IMS Mobile Solution
  - Requires z/OS Connect

IBM products support JSON!

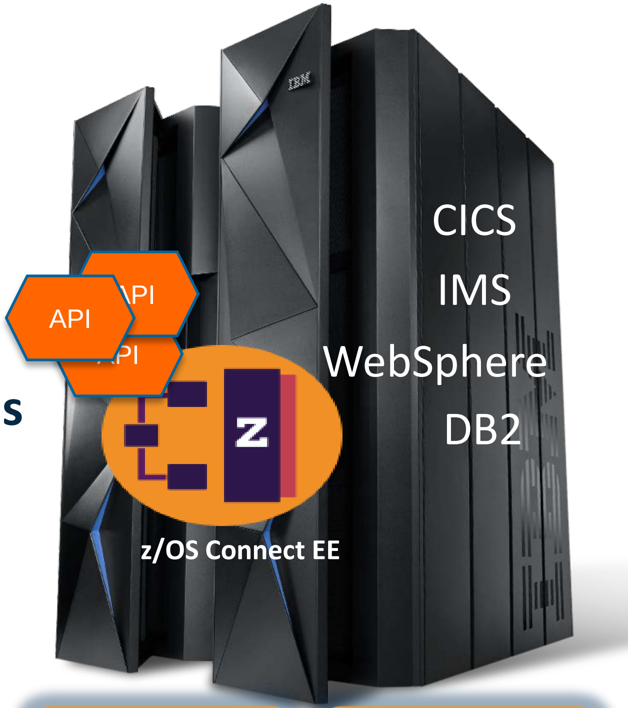
- This presentation will focus on how COBOL can work with z/OS Connect and z/OS Client Web Enablement Toolkit

IBM products support JSON!

- z/OS Client Web Enablement Toolkit
  - JSON Parse and Serialize (generate)
  - HTTP services
  - Classic callable interfaces
- z/OS Connect V2

z/OS Connect Enterprise Edition (EE) V2.0

- **Provides "System APIs" creation for z/OS subsystem applications**
- **Integrates with "IBM API Connect"** for enterprise-class API management
  - Create, Run, Secure
- **Delivers RESTful APIs as a discoverable, first-class resource with OpenAPI Spec (Swagger 2.0) descriptions**
  - Ready for consumption by today's enterprise application developers and integration with API management solutions
- **Comprehensive tooling that enables API developers to create RESTful APIs from z/OS-based assets**
- **Supports standard JSON message format** and conversion to z/OS subsystem backend format requirements

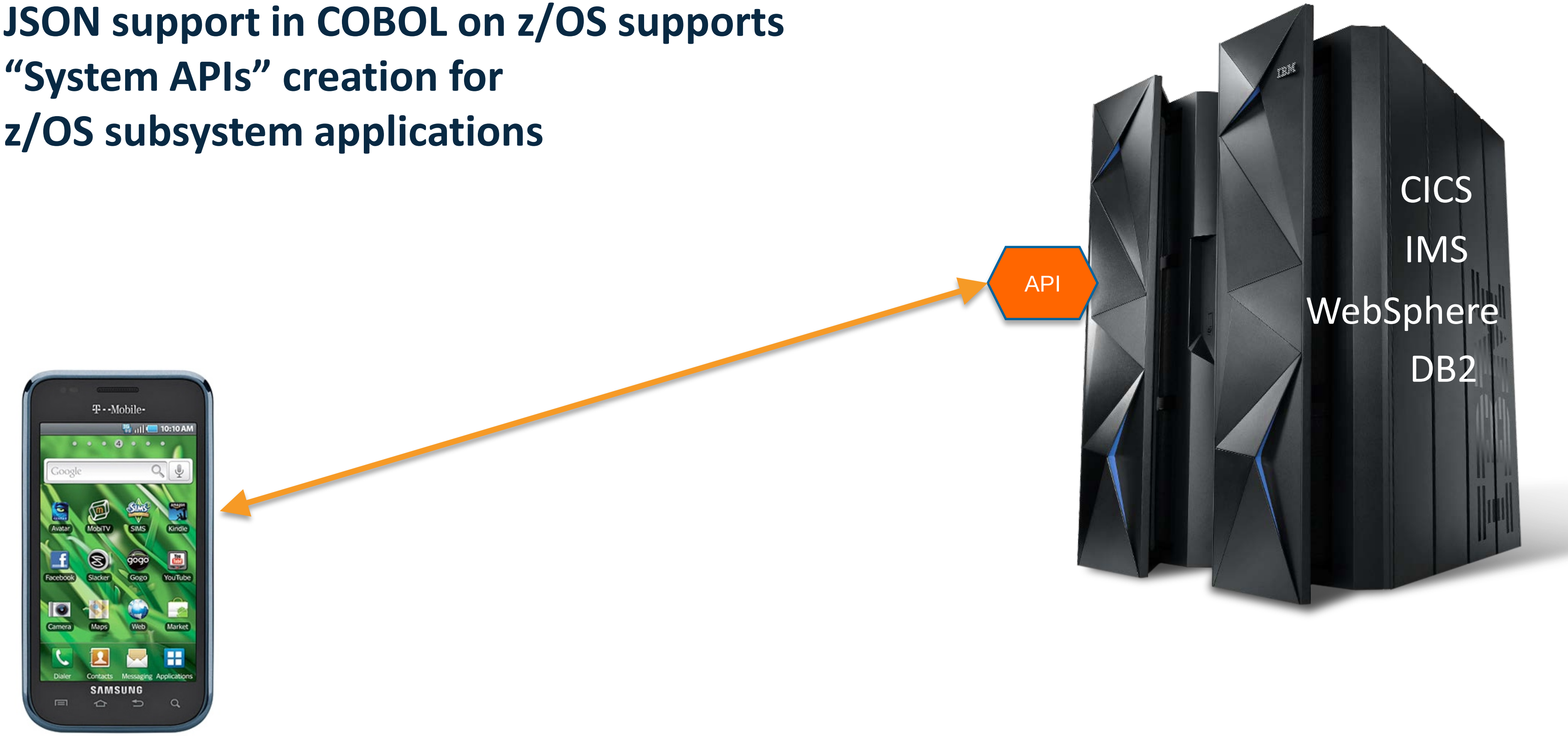


Announce
Dec 1, 2015

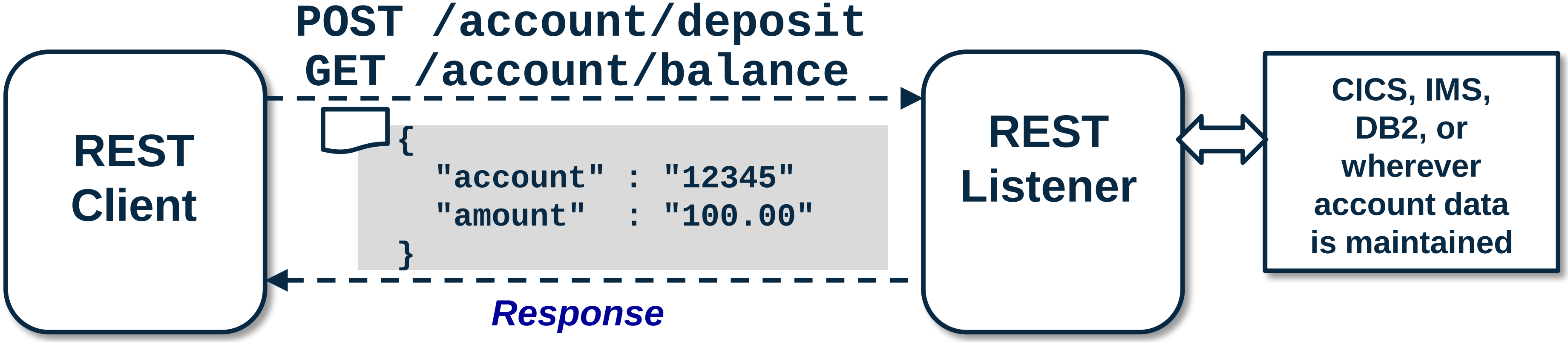
GA
Dec 11, 2015

RESTful services on z/OS

- **JSON support in COBOL on z/OS supports “System APIs” creation for z/OS subsystem applications**



Simplified Overview of REST/JSON



```
POST /account/deposit
GET /account/balance
```

REST Client

```
{
   "account" : "12345"
   "amount"  : "100.00"
}
```

Response

REST Listener

CICS, IMS, DB2, or wherever account data is maintained

REST - “Representational State Transfer” ... which uses HTTP and HTTP verbs to allow a client to interact with a server over the TCP/IP network.

JSON - “JavaScript Object Notation” ... a name/value pair representation of data that is relatively lightweight and generally simpler to handle and parse than XML.

REST is increasingly popular as an integration pattern because it is stateless, relatively lightweight, and is relatively easy to program

Where do the parts fit?

- I will talk mostly about 2 scenarios:

1. Making z/OS transactions available to mobile computing devices via z/OS Connect
2. Changing z/OS applications to enable them to access RESTful Services (either on or off z/OS) using the z/OS Client Web Enablement Toolkit

- There is at least one other scenario:
  - Changing existing z/OS services from handling only XML-encoded data to be able to handle JSON-encoded requests as well
    - ✓ Use techniques from number 2 above

Scenario 1

- z/OS transactions to be ‘converted’ into RESTful Services on the Web can be:
  - CICS transactions
  - IMS transactions
  - DB2 Stored Procedures
  - WAS applications
  - Batch applications that never stop ☺
- Clients for such RESTful requests can be:
  - Smart Phone applications
  - Web Browser interface
  - z/OS programs!

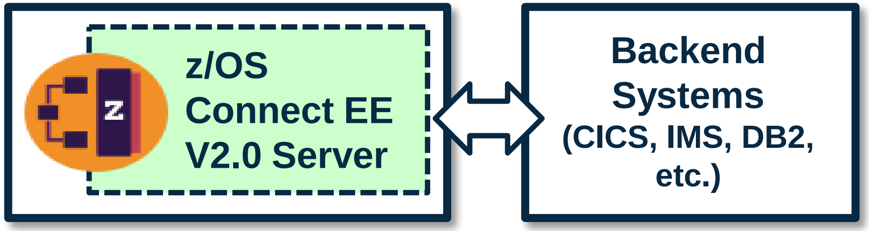
Scenario 1

- z/OS Connect can be used to
  - Build APIs that are used by remote devices or web sites
    - “System APIs”
  - Provide the ‘listener’ that is ready to accept service requests and
  - Pass the requests to the z/OS application

Scenario 1

Runtime Server

- Hosts APIs you define to run
- Connects with backend system
- You may have multiple instances

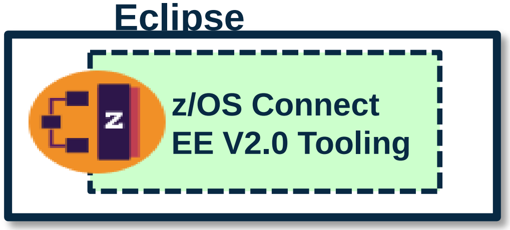

z/OS Connect EE V2.0 Server

Backend Systems (CICS, IMS, DB2, etc.)

Based on Liberty z/OS

- IBM z/OS V1R13, V2.1+
- IBM 64-bit SDK for z/OS, Java Technology Edition V7.1.0 or V8.0.0

Eclipse

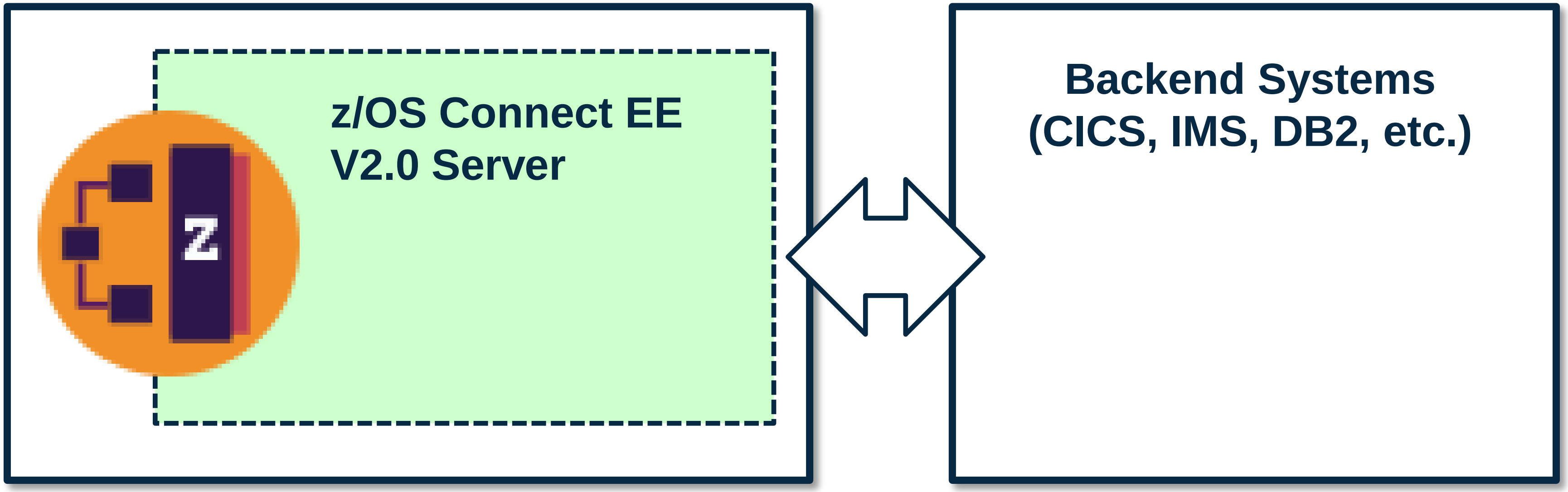

z/OS Connect EE V2.0 Tooling

- IBM CICS Explorer V5.3
- IBM IMS Explorer for Development V3.2
- IBM Explorer for z/OS Aqua V3.0

Tooling Platform

- Integrates with an Eclipse environment
- Define APIs
- Define data mapping
- Deploy APIs to runtime server
- Export API archive for other tools to deploy

Scenario 1



- z/OS Connect can be:
  - A ‘data conversion service’, that converts HTTP headers and JSON payloads into binary data interfaces to existing unchanged applications/transactions (Backend Systems)
  - A ‘passthrough service’, that passes HTTP headers and JSON texts through to the Backend Systems

Scenario 1.1

- With a ‘data conversion service’ HTTP headers and JSON payloads are converted into binary data interfaces to existing unchanged applications/transactions:
  - Ideally, no need to make changes to existing z/OS applications
  - There are some restrictions on data structures for interface that might not support your applications

Scenario 1.2

- With a ‘passthrough service’, the existing z/OS application code needs to be changed to process JSON text
  - HTTP headers can be processed by z/OS Connect or optionally passed through
  - For processing the request, this could be done in COBOL programs using z/OS Client Web Enablement Toolkit
    - In the future COBOL will have JSON PARSE
  - For processing the response, you could use JSON GENERATE in COBOL V6.1

Scenario 1.2

- Using z/OS Client Web Enablement services
  - RESTful Service to provide a life insurance quotation
  - Parse Incoming request JSON text
    - Note that the z/OS Client Web Enablement Toolkit requires the JSON text to be in EBCDIC, USA codepage only!
  - Use existing code to get the quotation
  - Return response JSON text

Scenario 1.2

```
PROCEDURE DIVISION USING DFHCOMMAREA.
  begin.
    PERFORM initialize-parser.

    If HWTJ-OK Then
      SET parser-initialized-true TO TRUE
      PERFORM get-JSON-text
      PERFORM parse-JSON-text
      PERFORM process-JSON-text
    End-If
```

Scenario 1.2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Method: initialize-parser                                      *
*                                                                *
* Initializes the parser handle variable.                        *
*                                                                *
* HWTJINIT: Provides a handle to a parse instance which is then  *
*           used in subsequent service calls. The HWTJINIT       *
*           service must be invoked before invoking any other    *
*           parsing service.                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 initialize-parser.
*    Set the parser-initialized flag to false.
     SET parser-initialized-false TO TRUE.

     CALL "HWTJINIT" USING
       HWTJ-RETURN-CODE
       workarea-max       *> parser work area size in bytes (input)
       HWTJ-PARSERHANDLE
       HWTJ-DIAG-AREA.

     IF (HWTJ-OK)
       DISPLAY "SUCCESS: Parser initialized."
     END-IF
```

Scenario 1.2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  Get incoming JSON text and set pointers
*  For passthrough in CICS, z/OS Connect puts the JSON
*  text in a container
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 Get-JSON-text.
*  Get the JSON text from the service request
      EXEC CICS GET CONTAINER(DFH-BODY)
        INTO request-JSON

*   Calculate the length of the JSON string.
      PERFORM VARYING pos FROM LENGTH OF request-JSON
          BY -1 UNTIL request-JSON (POS:1) NOT = SPACE
      END-PERFORM
      COMPUTE request-JSON-len = POS

* Set a pointer to the JSON string.
      SET request-JSON-ptr TO
          ADDRESS OF request-JSON
```

Scenario 1.2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
* Parse the sample JSON data.  Services Used:                   *
*                                                                *
* HWTJPARS: Builds an internal representation of the specified   *
*           JSON string. This allows efficient search, traversal,*
*           and modification of the JSON data.                   *
*                                                                *
* USAGE: HWTJPARS does not make a local copy of the JSON source  *
*        string. Therefore, the caller must ensure that the      *
*        provided source string remains unmodified for the       *
*        duration of the parser instance. If the source string  *
*        is modified, subsequent service calls may result in     *
*        unexpected behavior.                                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 parse-JSON-text.

     CALL "HWTJPARS" USING
          HWTJ-RETURN-CODE
          HWTJ-PARSERHANDLE
          JSON-text-ptr   *> address of JSON text string (input)
          JSON-text-len   *> length of JSON text string (input)
          HWTJ-DIAG-AREA.
```

Scenario 1.2

```
 get-request-array.
* Set the search string with the name of the request array.
     MOVE ‘request’ TO search-string.
     SET search-string-ptr TO ADDRESS OF search-string.
     Compute search-string-len = 7
* In order to start the search at the root, we specify zero for
* the search handle.
     MOVE X'00000000' TO HWTJ-HANDLE.

* Limit the search scope to within the specified object.
     SET HWTJ-SEARCHTYPE-OBJECT TO TRUE.

* Search for request array starting at root of JSON text.
     CALL "HWTJSRCH" USING     HWTJ-RETURN-CODE   HWTJ-PARSERHANDLE
        HWTJ-SEARCHTYPE        *> limit the search scope (input)
        search-string-ptr      *> search string address (input)
        search-string-len      *> search string length (input)
        HWTJ-HANDLE            *> object to be searched (input)
        starting-search-handle *> search start point (input)
        request-array-handle  *> search result handle (output)
        HWTJ-DIAG-AREA.
```

Scenario 1.2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Loop through the request array, retrieving each name:value    *
* pair, saving each value in WORKING-STORAGE.                   *
*                                                               *
*   HWTJGAEN - Retrieves a handle to an array entry.            *
*   HWTJGNUE - Gets number of entries in a JSON object/array.   *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 process-JSON-text Section.
 process-request-array.

*    Get the number of entries in the request array.
     CALL "HWTJGNUE" USING  HWTJ-RETURN-CODE   HWTJ-PARSERHANDLE
       request-array-handle      *> Handle to an array (input)
       num-of-requests           *> Number of array entries (output)
       HWTJ-DIAG-AREA.

*    Initialize the loop index value and the request table index.
     MOVE 0 TO index-value.
     SET request-array-index TO 1.
```

Scenario 1.2

```
*    Use the value returned by HWTJGNUE to loop thru the array.
     PERFORM VARYING index-value FROM 0 BY 1
       UNTIL index-value >= num-of-requests

*      Retrieve the i-th entry from the requests array.
       CALL "HWTJGAEN" USING  HWTJ-RETURN-CODE  HWTJ-PARSERHANDLE
         request-array-handle      *> request array handle (input)
         index-value               *> index into the request array (input)
         request-entry-handle      *> array entry handle (output)
         HWTJ-DIAG-AREA

       IF (HWTJ-OK)
* Extract the request information from the JSON data and process.
         PERFORM process-request-info
       ELSE
         DISPLAY "ERROR: Unable to retrieve array entry."
         CALL "DISPDIAG" USING HWTJ-RETURN-CODE HWTJ-DIAG-AREA
       END-IF

       SET request-array-index UP BY 1
     END-PERFORM
```

Scenario 1.2

```
 process-request-info.
* Move data from JSON into COBOL group for this request
* Use the "find-" routines to retrieve each value from the request
* entry, and move the value into the request entry table.
     MOVE ‘request-type’ TO search-string
     Compute search-string-len = 12
     CALL ‘findstring’ USING
                        request-entry-handle  *> input
                        search-string          *> input
                        search-string-len      *> input
                        output-buffer          *> output
                        actual-value-length    *> output

     MOVE output-buffer(1:actual-value-length)
       TO request-type

     If request-type = ‘get-quote’
        Perform process-quote
```

Scenario 1.2

```
 process-quote.
* Move data from JSON into COBOL group for this quote request
     MOVE ‘client-age’ TO search-string
     Compute search-string-len = 10
     CALL ‘findstring’ USING request-entry-handle   *> input
                             search-string          *> input
                             search-string-len      *> input
                             output-buffer          *> output
                             actual-value-length    *> output

     Compute client-age = FUNCTION NUMVAL output-buffer(1:actual-value-length)

     MOVE ‘smoker’ TO search-string
     Compute search-string-len = 6
     CALL ‘findstring’ USING request-entry-handle   *> input
                             search-string          *> input
                             search-string-len      *> input
                             output-buffer          *> output
                             actual-value-length    *> output

     MOVE output-buffer(1:actual-value-length)
       TO smoker
```

Scenario 1.2

```
* process-quote. (continued)

•  Now get the quotation using the input data and existing
•  CALL ‘GETQUOTE’ Using Get-Quote-Group   *> input
                        Quote-amount      *> output

* Now prepare the response to the service Request
* Create the JSON string
    If quote-success Then
       String '{‘ '“quote-response":’
                  '{‘ '“quote-status":success ' ','
                      '“quote-amount":’ Quote-amount '}'
              '}'
           Into JSON-string with Pointer JSON-string-len

* Insert the string into the existing parsed JSON
```

Scenario 1.2

```
* Insert the string into the existing parsed JSON
* Set a pointer to the JSON string.
       SET response-JSON-ptr TO ADDRESS OF JSON-string

* HWTJCREN service expects an entry name of 0 in this case.
* We are not specifying an entry name, set entry name length to 0.
       MOVE x'00000000' TO entry-name
       MOVE 0 TO entry-name-len

* We are inserting JSON text, set the entryValueType accordingly.
       SET HWTJ-JSONTEXTVALUETYPE TO TRUE

* Perform the insertion.
       CALL "HWTJCREN" USING  HWTJ-RETURN-CODE HWTJ-PARSERHANDLE
        request-array-handle  *> handle to the insertion point (input)
        entry-name            *> name of the object entry (input)
        entry-name-len        *> length of the name (input)
        HWTJ-ENTRYVALUETYPE   *> entry type (input)
        response-JSON-ptr     *> JSON string address (input)
        JSON-string-len       *> JSON string length (input)
        new-request-handle    *> entry handle (output)
        HWTJ-DIAG-AREA
```

Scenario 1.2

```
* Serialize the new JSON into a string
  serialize-JSON-data.

      SET serialize-buffer-ptr TO
        ADDRESS OF serialize-buffer.

 * Generate a JSON formatted text string using the current state
 * of JSON data.
      CALL "HWTJSERI" USING
           HWTJ-RETURN-CODE
           HWTJ-PARSERHANDLE
           serialize-buffer-ptr  *> buffer address (input)
           serialize-buffer-size *> buffer size (input)
           actual-output-length  *> size of JSON data (output)
           HWTJ-DIAG-AREA.

*  Put the JSON response text for this service request
      EXEC CICS PUT CONTAINER(DFH-BODY)
        FROM serialize-buffer FLENGTH(actual-output-length)
```

Scenario 1.2.1

- Using new COBOL statement JSON GENERATE
  - Enterprise COBOL V6.1
  - Use a COBOL group with input and output areas:

```
01 request.
   05 request-type       Pic X(10). *> Input
   05 client-age         Pic 9(3).  *> Input
   05 smoker             Pic X.     *> Input
   05 quote-response.               *> Output
      10 quote-status    Pic X(7).  *> Output
      10 quote-amount    Pic 9(9).  *> Output
```

Scenario 1.2.1

- Using new COBOL statement JSON GENERATE
  - No need to insert into existing JSON since we already have the data in a COBOL group

```
*  Serialize the data into JSON text
      JSON GENERATE      *> COBOL V6.1
                    From request Into serialize-buffer
                    Count In actual-output-length

*  Put JSON response text for this service request
      EXEC CICS PUT CONTAINER(DFH-BODY)
                FROM serialize-buffer
                FLENGTH(actual-output-length)
```

Scenario 1.2.x

- If COBOL added JSON PARSE…INTO (in design now)
  - You could do all of the code in slides 20-31 with one statement!
  - If would be about as simple as this:

```
    EXEC CICS GET CONTAINER(DFH-BODY)
          INTO request-JSON
    JSON PARSE request-JSON Into request
    PERFORM process-request
    JSON GENERATE From request Into buffer
       COUNT IN actual-output-length

*   Put the JSON response text for this service request
    EXEC CICS PUT CONTAINER(DFH-BODY)
       FROM buffer FLENGTH(actual-output-length)
```

Scenario 2

Scenario 2

- We have an existing z/OS application that wants/needs to invoke a RESTful Service
  - Using Apache and Java is complicated and requires a different skill set
  - Use z/OS Client Web Enablement Toolkit services to do the same thing, but with no requirement for 3rd party software or Java
    - In this Scenario we will use HTTP services provided by z/OS Client Web Enablement Toolkit to create and use a connection to a RESTful Service
    - In this example, a sample RESTful service built for IBM IMS that implements a phone book application will be used. Entries in the phone book can be looked up, added or deleted.

Scenario 2 (flow)

```
*> Initialize and set up a connection handle
  Perform HTTP-Init-Connection

*> Set the required options before connecting to the server
  Perform HTTP-Setup-Connection

*> Connect to the HTTP server
  Perform HTTP-Connect

*> Initialize and set up a request
  Perform HTTP-Init-Request

*> Set the necessary options before connecting
*> to the server.
  Perform HTTP-Setup-Request

*> Send the request
  Perform HTTP-Issue-Request
```

Scenario 2 (flow)

```
*> Terminate the request
  Perform HTTP-Terminate-Request

*> Disconnect the connection
  Perform HTTP-Disconnect

*> Terminate the connection
  Perform HTTP-Terminate-Connection
```

Scenario 2

```
****************************************************************
*                                                              *
* Function: HTTP-Init-Connection                               *
*   Initializes a connection handle using the HWTHINIT service *
*                                                              *
****************************************************************
 HTTP-Init-Connection.

     Set HWTH-HANDLETYPE-CONNECTION to true.

     Call "HWTHINIT" using
       HWTH-RETURN-CODE
       HWTH-HANDLETYPE
       Conn-Handle
       HWTH-DIAG-AREA

     If (HWTH-OK)
       Display "** Initialize succeeded (HWTHINIT)"
     Else
       Display "HWTHINIT FAILED: "
       Call "DSPHDIAG" using
                       HWTH-RETURN-CODE
                       HWTH-DIAG-AREA
     End-If
     .
```

Scenario 2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
* Function: HTTP-Setup-Connection                                *
*           Sets the necessary connection options                *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 HTTP-Setup-Connection.
     *>*********************************************************
     *> | First, set the verbose option on. This option is     |
     *> | handy when developing an application. Lots of        |
     *> | informational error messages are written to          |
     *> | standard output to help in debugging efforts.        |
     *> | This option should likely be turned off with         |
     *> | HWTH_VERBOSE_OFF or just not set at all (default is  |
     *> | off) when the application goes into production.      |
     *>*********************************************************
     Set HWTH-OPT-VERBOSE to true.
     Set HWTH-VERBOSE-ON to true.
     Set option-val-addr to address of HWTH-VERBOSE.
     Compute option-val-len = function length (HWTH-VERBOSE).

     Call "HWTHSET" using HWTH-RETURN-CODE
                          Conn-Handle
                          HWTH-Set-OPTION
                          option-val-addr
                          option-val-len
                          HWTH-DIAG-AREA.
```

Scenario 2

```
* Function: HTTP-Setup-Connection                                *
     *>  ______________________________________________________
     *> |                                                      |
     *> | Set URI for connection handle.                       |
     *> |   Connect to the IBM RESTful service host            |
     *> |______________________________________________________|
     Set HWTH-OPT-URI to true
     Move "https://zserveros.centers.ihost.com" to option-val-char
     Set option-val-addr to address of option-val-char
     Move 36 to option-val-len

     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    Conn-Handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Connection                                *
      *>  ______________________________________________________
      *> |                                                      |
      *> | Set HWTH_OPT_COOKIETYPE                              |
      *> |   Enable the cookie engine for this connection.  Any |
      *> |   "eligible" stored cookies will be resent to the    |
      *> |   host on subsequent interactions automatically.     |
      *> |   interactions automatically.                        |
      *> |______________________________________________________|
      Set HWTH-OPT-COOKIETYPE to true
      Set HWTH-COOKIETYPE-SESSION to true
      Set option-val-addr to address of HWTH-COOKIETYPE
      Compute option-val-len =
          function length (HWTH-COOKIETYPE)
```

Scenario 2

```
*> Connect to the HTTP server
  HTTP-Connect.
      Call "HWTHCONN" using
                HWTH-RETURN-CODE
                Conn-Handle
                HWTH-DIAG-AREA

      If (HWTH-OK)
        Display "** Connect succeeded (HWTHCONN)"
      Else
        Display "Connect failed (HWTHCONN)."
        Call "DSPHDIAG" using
                           HWTH-RETURN-CODE
                           HWTH-DIAG-AREA
      End-If
```

Scenario 2

```
****************************************************************
*                                                              *
* Function: HTTP-Init-Request                                  *
*   Initializes a request handle using the HWTHINIT service    *
*                                                              *
****************************************************************
 HTTP-Init-Request.

     Set HWTH-HANDLETYPE-HTTPREQUEST to true.

     Call "HWTHINIT" using
         HWTH-RETURN-CODE
         HWTH-HANDLETYPE
         Rqst-Handle
         HWTH-DIAG-AREA

     If (HWTH-OK)
       Display "** Initialize succeeded (HWTHINIT)"
     Else
       Display "HWTHINIT FAILED: "
       Call "DSPHDIAG" using HWTH-RETURN-CODE  HWTH-DIAG-AREA
     End-If
     .
```

Complete your session evaluations online at SHARE.org/Evaluation

Scenario 2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
* Function: HTTP-Setup-Request                                   *
*           Sets the necessary request options                   *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 HTTP-Setup-Request.
     *>  ______________________________________________________
     *> |                                                      |
     *> | Set HTTP Request method.                             |
     *> |   A PUT request method is used to add an entry to    |
     *> |   the phone book database.                           |
     *> |______________________________________________________|
     Set HWTH-OPT-REQUESTMETHOD to true.
     Set HWTH-HTTP-REQUEST-PUT to true.
     Set option-val-addr to address of HWTH-REQUESTMETHOD.
     Compute option-val-len = function length (HWTH-REQUESTMETHOD).

     Display "** Set HWTH-REQUESTMETHOD for request"
     Call "HWTHSET" using HWTH-RETURN-CODE
                          rqst-handle
                          HWTH-Set-OPTION
                          option-val-addr
                          option-val-len
                          HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *>  ______________________________________________________
     *> |                                                      |
     *> | Set the request URI                                  |
     *> |   Set the URI that identifies a resource by name     |
     *> |   that is the target of our request.                 |
     *> |______________________________________________________|
     Set HWTH-OPT-URI to true
     Move 1 to option-val-len
     STRING “:33622/zwc/contacts" DELIMITED BY SIZE
            parm-string(1:parm-len) DELIMITED BY SIZE INTO
            option-val-char WITH POINTER option-val-len
     Set option-val-addr to address of option-val-char
     SUBTRACT 1 FROM option-val-len

     Display "** Set HWTH-OPT-URI for request"
     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    rqst-handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *> Create a list of HTTP headers
     Perform Build-Slist
```

Scenario 2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
* Function: Build-Slist                                          *
*           Sets the necessary request options                   *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
 Build-Slist
     *>  ______________________________________________________
     *> |                                                      |
     *> | Add the Accept request header                        |
     *> |   Create a brand new SLST and specify the first      |
     *> |   header to be an "ACCEPT" header that requests that |
     *> |   the server return the data requested by the GET    |
     *> |   request to be in JSON format.                      |
     *> |______________________________________________________|
     Move 1 to option-val-len.
     String "Accept: application/JSON" delimited by size
            into option-val-char with pointer option-val-len.
     Subtract 1 from option-val-len.

     Set option-val-addr to address of option-val-char.
     Set HWTH-SLST-NEW to true.

     Call "HWTHSLST" using  HWTH-RETURN-CODE rqst-handle
                     HWTH-SLST-function
                     Slist-Handle
                     option-val-addr
                     option-val-len
                     HWTH-DIAG-AREA.
```

Scenario 2

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
* Function: Build-Slist  (continued)                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * **
*> -----------------------------------------------------------------*
*> | Add the Accept-Language request header                         |
*> |   Append to the just-created SLST and specify an additional    |
*> |   additional option "Accept-Language" to tell server the       |
*> |   regional settings preferred by this application.             |
*> -----------------------------------------------------------------*
  Move 1 to option-val-len
  String "Accept-Language: en-US" delimited by size
         into option-val-char with pointer option-val-len
  Subtract 1 from option-val-len

  Set option-val-addr to address of option-val-char
  Set HWTH-SLST-APPEND to true

  Display "** Adding SLIST APPEND"
  Call "HWTHSLST" using  HWTH-RETURN-CODE
                  rqst-handle
                  HWTH-SLST-function
                  Slist-Handle
                  option-val-addr
                  option-val-len
                  HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *> Specify the HTTP request headers
     Set HWTH-OPT-HTTPHEADERS to true
     Set option-val-addr to address of Slist-Handle
     Compute option-val-len = function length(Slist-Handle)

     Display "** Set HWTH-OPT-HTTPHEADERS for request"
     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    rqst-handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *> Add a request body to this request

     *> Put the request data into a JSON text
     JSON GENERATE JSON-text From request   *> COBOL V6.1 only

     Set HWTH_OPT_REQUESTBODY to true
     Set option-val-addr to address of JSON-text
     Compute option-val-len = length of option-val-addr

     Display "** Set HWTH-OPT-REQUESTBODY for request"
     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    rqst-handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

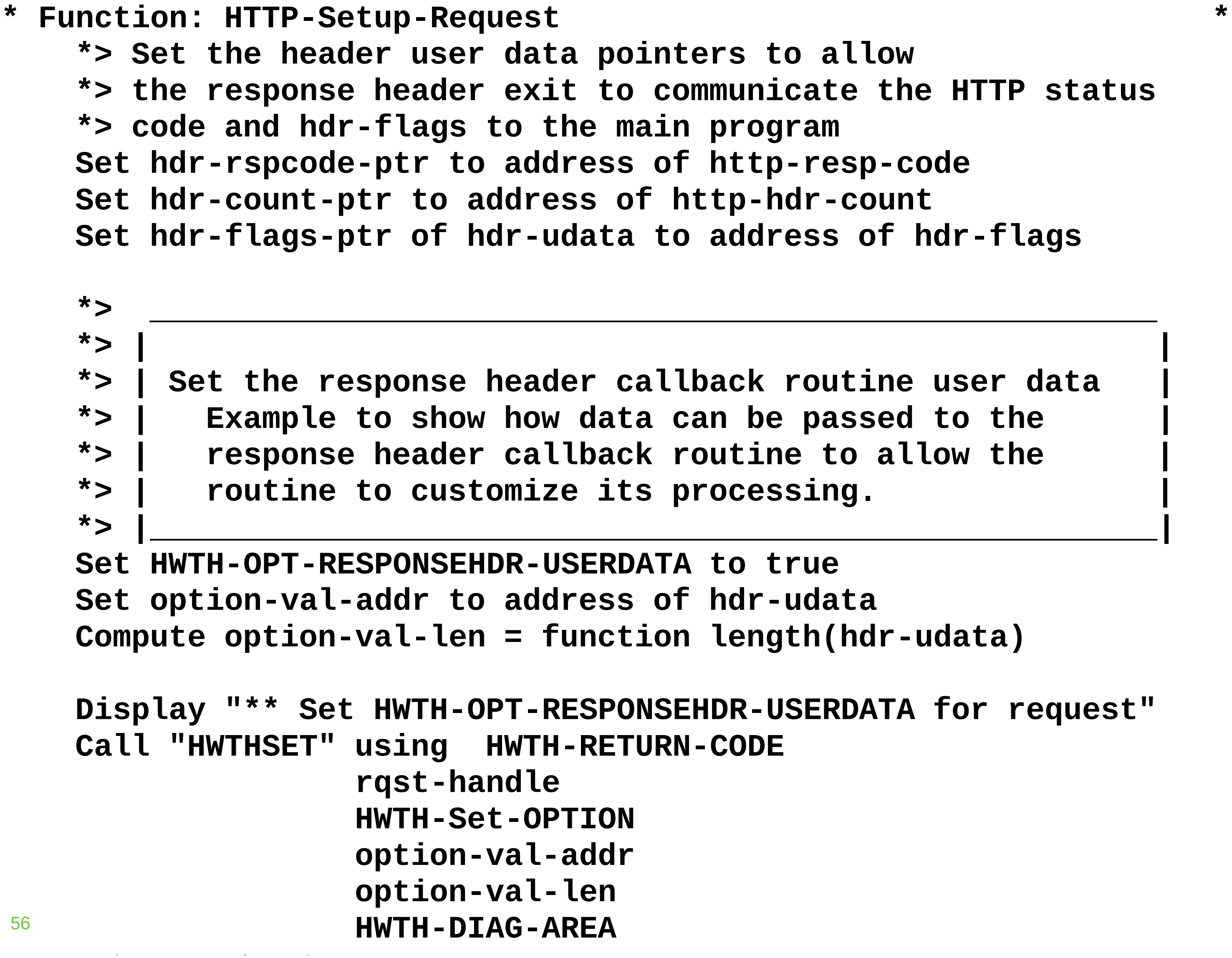
Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *> Direct the toolkit to convert the response body
     *> from ASCII to EBCDIC
     Set HWTH-OPT-TRANSLATE-RESPBODY to true
     Set HWTH-XLATE-RESPBODY-A2E to true
     Set option-val-addr to address of HWTH-XLATE-RESPBODY
     Compute option-val-len =
         function length (HWTH-XLATE-RESPBODY)

     Display "** Set HWTH-OPT-TRANSLATE-RESPBODY for request"
     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    rqst-handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                   *
     *>  ______________________________________________________
     *> |                                                      |
     *> | Set the response header callback routine             |
     *> |   Set the address of the routine that is to receive  |
     *> |   control once for every response header that we    |
     *> |   receive                                            |
     *> |______________________________________________________|
     Set HWTH-OPT-RESPONSEHDR-EXIT to true
     Set header-cb-ptr to ENTRY "HWTHHDRX"
     Set option-val-addr to address of header-cb-ptr
     Compute option-val-len =
         function length (header-cb-ptr)

     Display "** Set HWTH-OPT-RESPONSEHDR-EXIT for request"
     Call "HWTHSET" using
                    HWTH-RETURN-CODE
                    rqst-handle
                    HWTH-Set-OPTION
                    option-val-addr
                    option-val-len
                    HWTH-DIAG-AREA
```

Scenario 2



```
* Function: HTTP-Setup-Request                                  *
    *> Set the header user data pointers to allow
    *> the response header exit to communicate the HTTP status
    *> code and hdr-flags to the main program
    Set hdr-rspcode-ptr to address of http-resp-code
    Set hdr-count-ptr to address of http-hdr-count
    Set hdr-flags-ptr of hdr-udata to address of hdr-flags

    *>  ______________________________________________________
    *> |                                                      |
    *> | Set the response header callback routine user data   |
    *> |   Example to show how data can be passed to the      |
    *> |   response header callback routine to allow the      |
    *> |   routine to customize its processing.               |
    *> |______________________________________________________|
    Set HWTH-OPT-RESPONSEHDR-USERDATA to true
    Set option-val-addr to address of hdr-udata
    Compute option-val-len = function length(hdr-udata)

    Display "** Set HWTH-OPT-RESPONSEHDR-USERDATA for request"
    Call "HWTHSET" using  HWTH-RETURN-CODE
                   rqst-handle
                   HWTH-Set-OPTION
                   option-val-addr
                   option-val-len
                   HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                  *
      *>  ______________________________________________________
      *> |                                                      |
      *> | Set the response body callback routine               |
      *> |   Set the address of the routine that is to receive  |
      *> |   control if there is a response body returned by   |
      *> |   the server                                         |
      *> |______________________________________________________|
      Set HWTH-OPT-RESPONSEBODY-EXIT to true
      Set rspbdy-cb-ptr to ENTRY "HWTHBDYX"
      Set option-val-addr to address of rspbdy-cb-ptr
      Compute option-val-len =
          function length (rspbdy-cb-ptr)

      Display "** Set HWTH-OPT-RESPONSEBODY-EXIT for request"
      Call "HWTHSET" using
                     HWTH-RETURN-CODE
                     rqst-handle
                     HWTH-Set-OPTION
                     option-val-addr
                     option-val-len
                     HWTH-DIAG-AREA
```

Scenario 2

```
* Function: HTTP-Setup-Request                                    *
       *>  ______________________________________________________
       *> |                                                      |
       *> | Set the response body callback routine user data     |
       *> |   Example to show how data can be passed to the      |
       *> |   response body callback routine to allow the routine|
       *> |   to customize its processing.                       |
       *> |______________________________________________________|
       Set hdr-flags-ptr of body-udata to address of hdr-flags
       Set resp-body-data-ptr to address of resp-body-data

       Set HWTH-OPT-RESPONSEBODY-USERDATA to true
       Set option-val-addr to address of body-udata
       Compute option-val-len = function length(body-udata)

       Display "** Set HWTH-OPT-RESPONSEBODY-USERDATA for request"
       Call "HWTHSET" using
                      HWTH-RETURN-CODE
                      rqst-handle
                      HWTH-Set-OPTION
                      option-val-addr
                      option-val-len
                      HWTH-DIAG-AREA
```

Scenario 2

```
*****************************************************************
*                                                               *
* Function: HTTP-Issue-Request                                  *
*   Issues the HWTHRQST service and performs error checking     *
*                                                               *
*****************************************************************
 HTTP-Issue-Request.

     Call "HWTHRQST" using
       HWTH-RETURN-CODE
       Conn-Handle
       Rqst-Handle
       HWTH-DIAG-AREA

     If (HWTH-OK)
       Display "** Request succeeded (HWTHRQST)"
     Else
       Display "Request failed (HWTHRQST)."
       Call "DSPHDIAG" using
                       HWTH-RETURN-CODE
                       HWTH-DIAG-AREA
     End-If
```

Summary

- If you need to work with RESTful services today
  - Use z/OS Connect right away
    - Full support for CICS transactions
    - Partial support for IMS transactions (no PASS THROUGH)
  - Maybe wait on using z/OS Client Web Enablement Toolkit
    - Today only JSON in EBCDIC 1047 encoding can be parsed
    - A future release of COBOL will have JSON PARSE
      - Will be MUCH easier, and more efficient!
      - Will handle standard UTF-8 JSON
  - For creating JSON texts:
    - If you cannot get to COBOL V6 for a while
      - Use STRING (or similar) to create JSON texts manually
    - If you can get to COBOL V6
      - Use JSON Generate