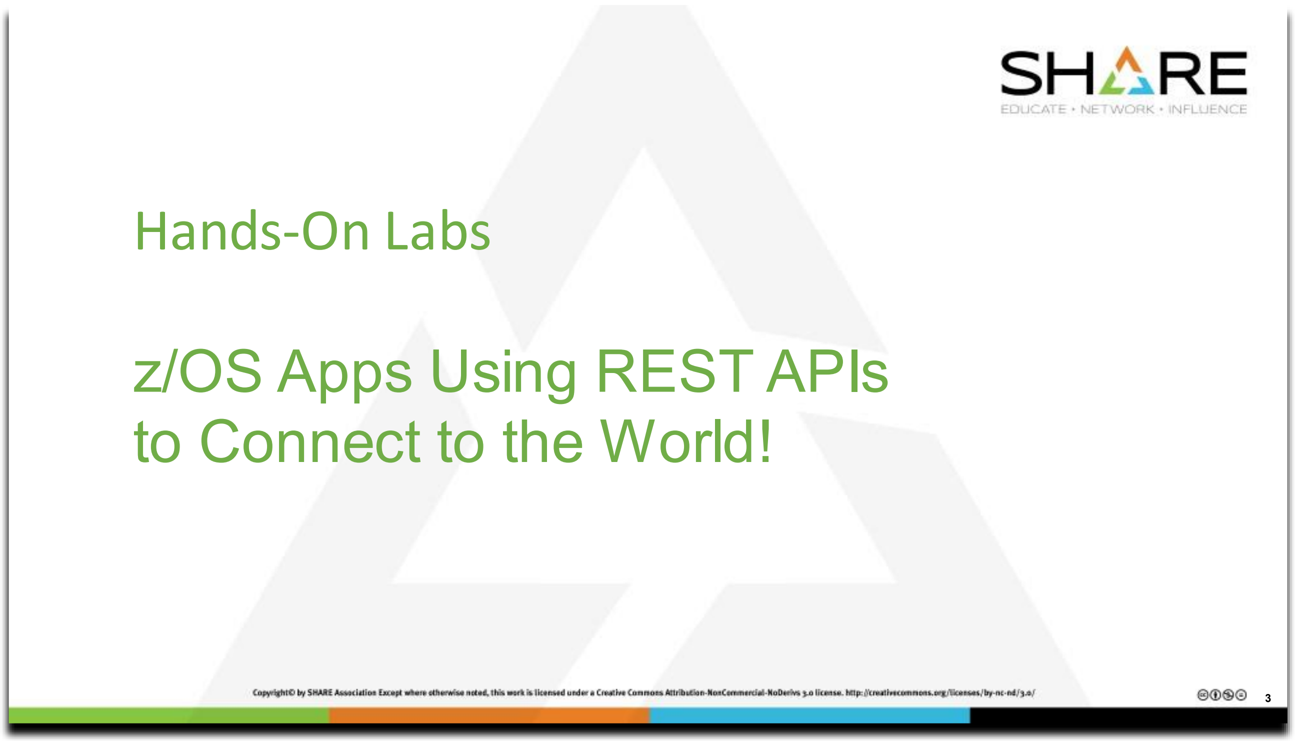


SHARE
EDUCATE • NETWORK • INFLUENCE

Hands-On Labs

z/OS Apps Using REST APIs to Connect to the World!

3

Lab Exercise: Writing a z/OS application leveraging REST APIs

Use the z/OS Client Web Enablement Toolkit to have a REXX application easily leverage published REST APIs with very simple API calls.

Lab: Writing a z/OS Application Leveraging REST APIs

What is the z/OS Client Web Enablement Toolkit?

The z/OS Client Web Enablement Toolkit provides a set of application programming interfaces (APIs) to enable traditional, native z/OS programs to participate in modern web services applications. Applications running in traditional z/OS environments can play the client role of a RESTful web application and initiate a request to a web server residing on z/OS or any other platform that supports web applications. First released into z/OS V2R2 and rolled back to V2R1, the z/OS client web enablement toolkit provides the following components to enable these applications to more easily participate in the client/server realm:

1. a z/OS JSON parser to parse JSON text coming from any source, build new JSON, or add to existing JSON text, and
2. a z/OS HTTP/HTTPS protocol enabler that uses interfaces similar to other industry-standard APIs.

The toolkit allows callers from many z/OS environments and supports many programming languages including: C/C++, COBOL, PL/I and Assembler. However, today's lab will focus on the REXX API support for the toolkit.

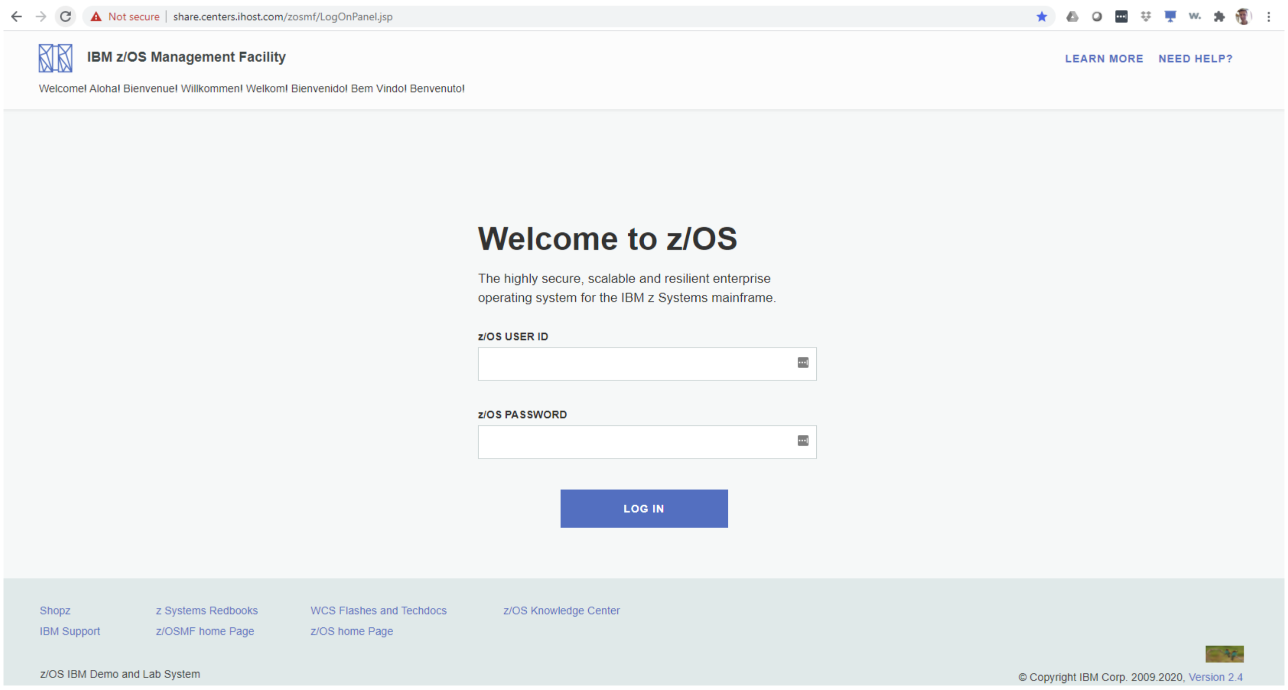
What this two-part lab exercise entails:

1. Use a published REST API to gather information to calculate the distance between two locations.
2. Using z/OSMF REST APIs, submit a healthcheck print job on a remote system, query the status of the job until complete using the z/OSMF get job status REST API, get output of job from data set using the z/OSMF data set REST API, and send an email to the sysprog.

1. Connect to the z/OSMF SHARE System:

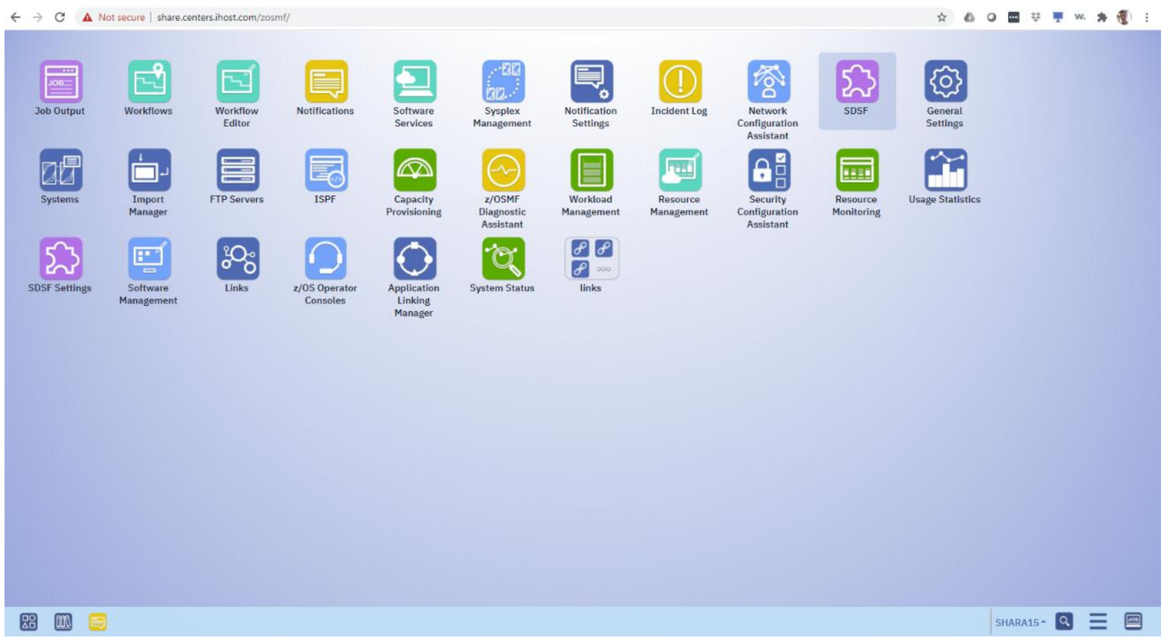
Point your web browser to **https://share.centers.ihost.com/zosmf/** . IBM recommends that you use Google Chrome for your browser.

Note: You may or may not receive a security error. If you do not get the z/OSMF logon screen as shown below but instead receive a security error, click on the Advanced button if shown and then Proceed. If the Advanced option is not displayed, click anywhere in the browser window and type "Thisisunsafe" to bypass the temporary certificate issue.


Not secure
share.centers.ihost.com/zosmf/LogOnPanel.jsp
IBM z/OS Management Facility
LEARN MORE
NEED HELP?
Welcome! Aloha! Bienvenue! Willkommen! Welkom! Bienvenido! Bem Vindo! Benvenuto!
Welcome to z/OS
The highly secure, scalable and resilient enterprise operating system for the IBM z Systems mainframe.
z/OS USER ID
z/OS PASSWORD
LOG IN
Shopz
IBM Support
z Systems Redbooks
z/OSMF home Page
WCS Flashes and Techdocs
z/OS home Page
z/OS Knowledge Center
z/OS IBM Demo and Lab System
© Copyright IBM Corp. 2009.2020, Version 2.4

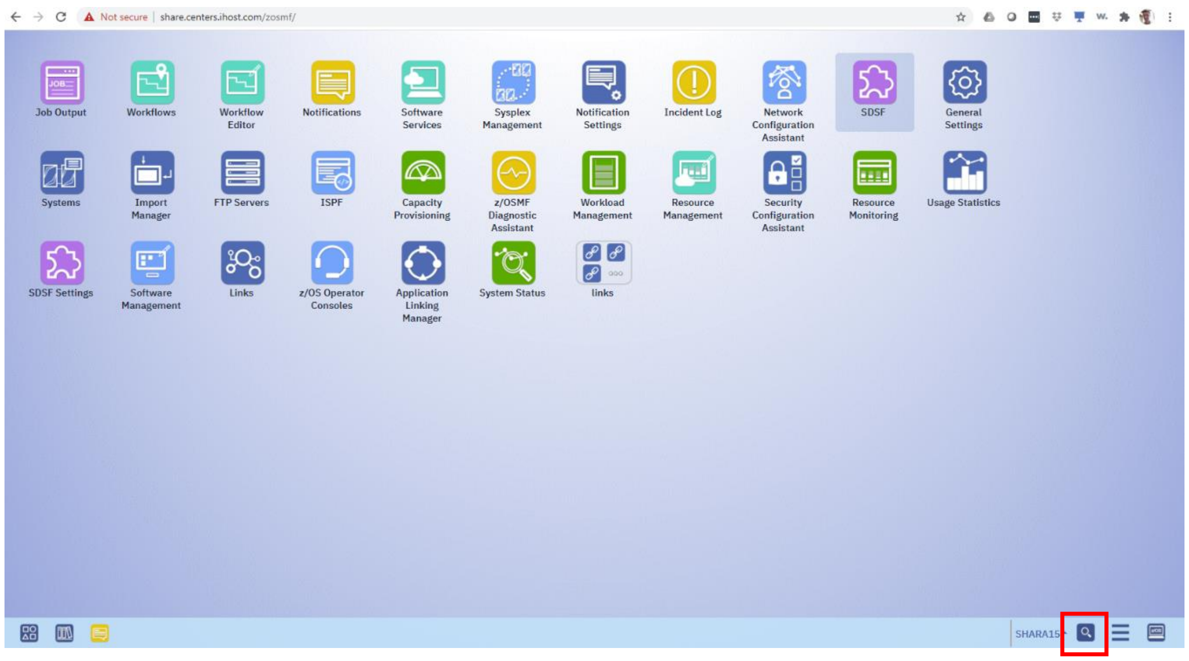
Enter your z/OS User ID and password assigned to you. For this handout, you will be shown examples using userid SHARA15. Click the Log In button after entering your credentials.

You will be presented with the z/OSMF Desktop for your userid that looks something like this:

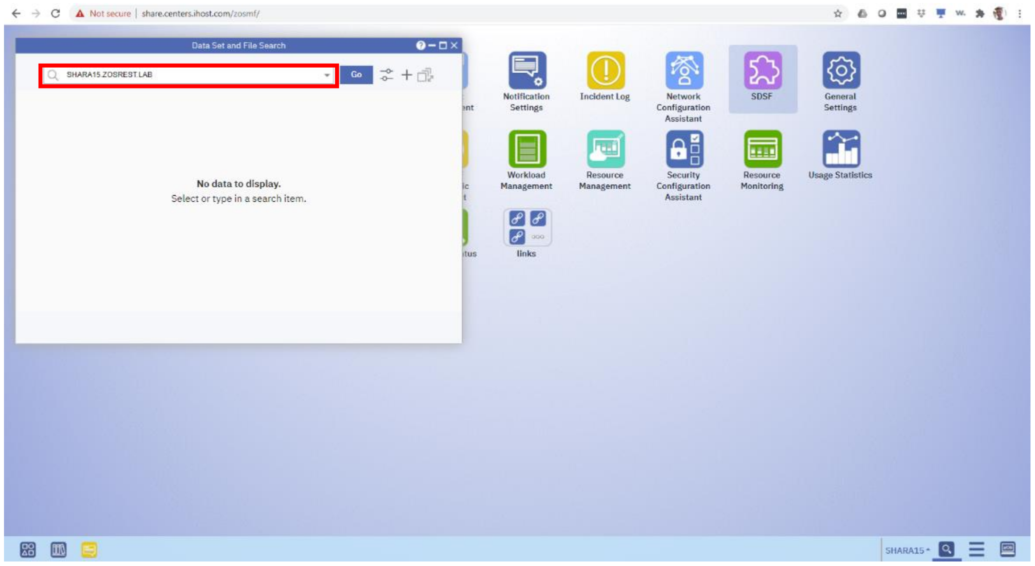


For this lab, we are going to be using the editing capabilities of z/OSMF, and thus avoiding to have to use a terminal emulator to edit our toolkit applications.

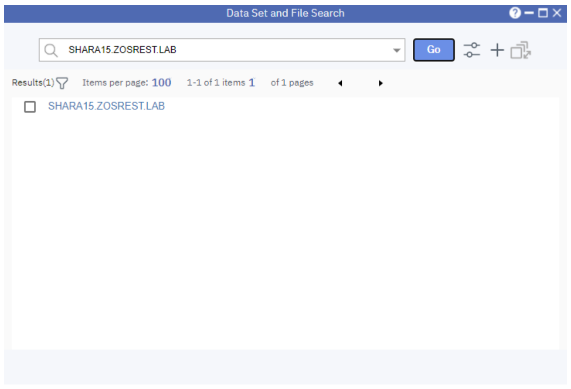
Click on the Data and File search icon in the bottom right corner of the desktop as shown below:



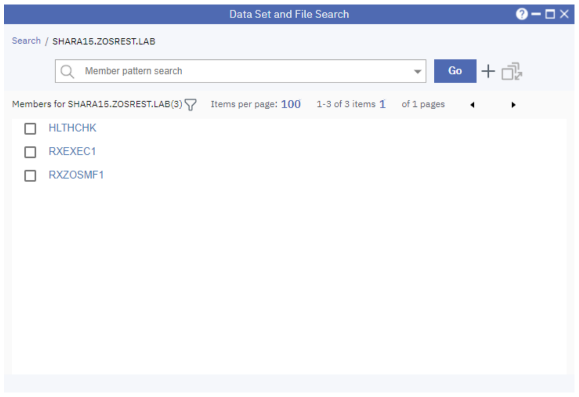
You will be presented with the following window. Enter ***labuserid*.ZOSREST.LAB (where *labuserid* is the user id you were assigned)** in the box provided.



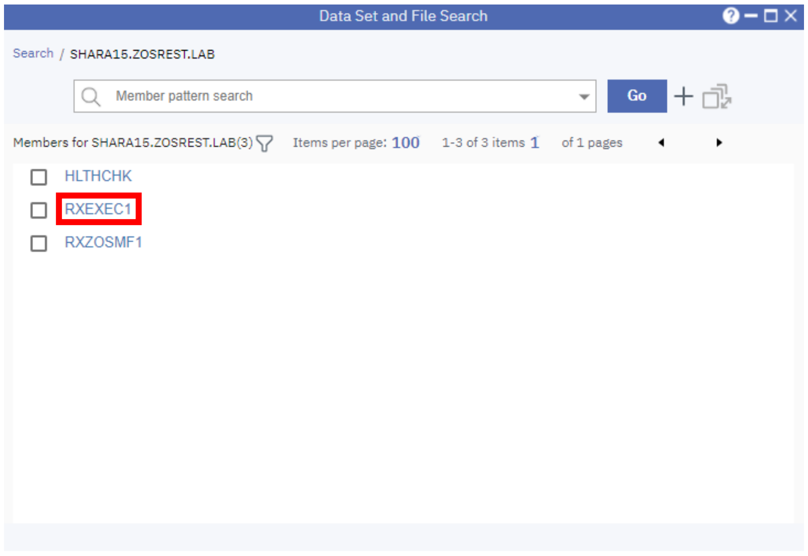
Click Go.



You are now presented with data sets that match your file search specification (in this case, exactly 1). Click on the data set name to show all the members inside this data set.

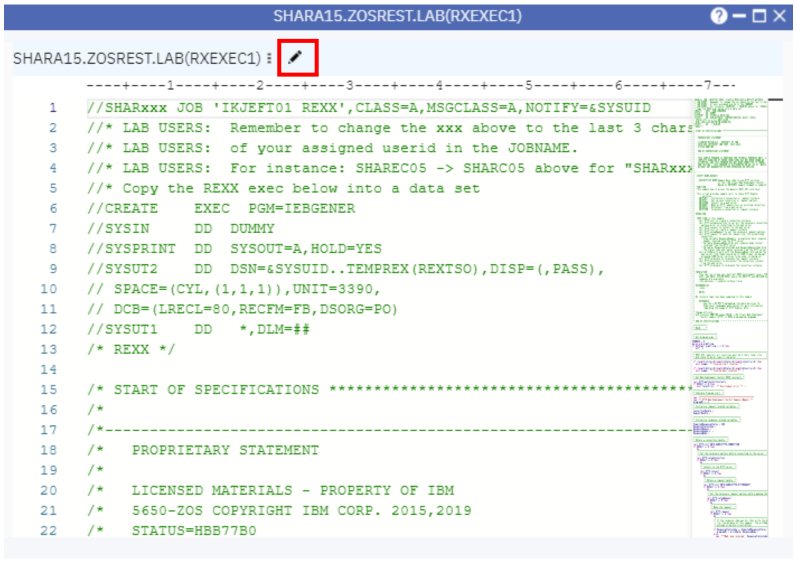

Data Set and File Search
Search / SHARA15.ZOSREST.LAB
Member pattern search
Go
Members for SHARA15.ZOSREST.LAB(3) Items per page: 100 1-3 of 3 items 1 of 1 pages
HLTHCHK
RXEXEC1
RXZOSMF1

Click on the RXEXEC1 member name to open this member.



HLTHCHK
RXEXEC1
RXZOSMF1

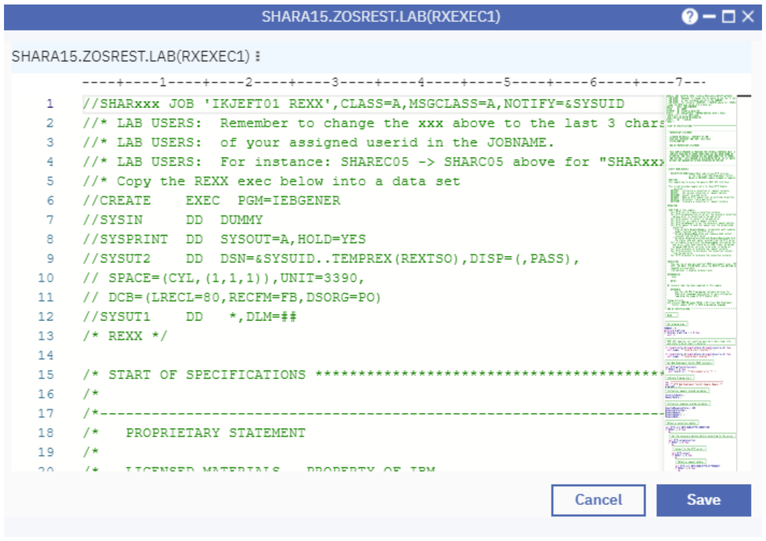
You are now browsing this member. Since we want to edit this file, click on the pen icon above to go into edit mode.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7---
//SHARxxx JOB 'IKJEFT01 REXX',CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//* LAB USERS:  Remember to change the xxx above to the last 3 chars
//* LAB USERS:  of your assigned userid in the JOBNAME.
//* LAB USERS:  For instance: SHAREC05 -> SHARC05 above for "SHARxxx
//* Copy the REXX exec below into a data set
//CREATE    EXEC  PGM=IEBGENER
//SYSIN     DD  DUMMY
//SYSPRINT  DD  SYSOUT=A,HOLD=YES
//SYSUT2    DD  DSN=&SYSUID..TEMPREX(REXTSO),DISP=(,PASS),
// SPACE=(CYL,(1,1,1)),UNIT=3390,
// DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSUT1    DD   *,DLM=##
/* REXX */

/* START OF SPECIFICATIONS ****************************************
/*
/*-----------------------------------------------------------------
/*   PROPRIETARY STATEMENT
/*
/*   LICENSED MATERIALS - PROPERTY OF IBM
/*   5650-ZOS COPYRIGHT IBM CORP. 2015,2019
/*   STATUS=HBB77B0
```

You are now in edit mode. You can tell because: 1) There is a save button on the bottom right; and 2) the Pen icon is gone.



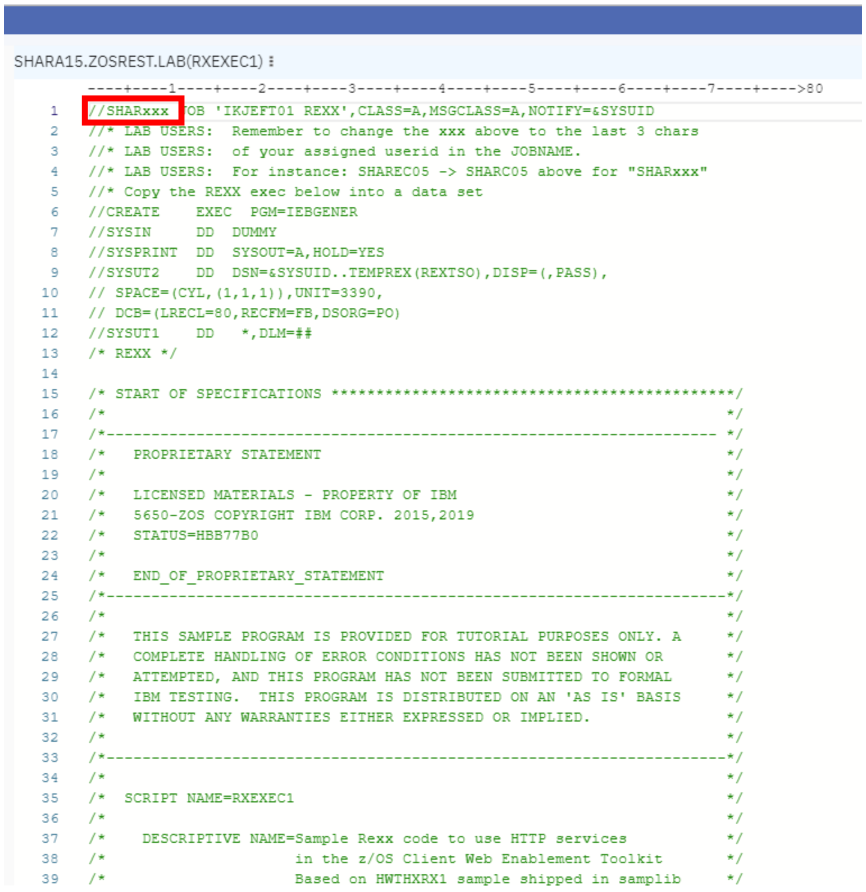
```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7---
//SHARxxx JOB 'IKJEFT01 REXX',CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//* LAB USERS:  Remember to change the xxx above to the last 3 chars
//* LAB USERS:  of your assigned userid in the JOBNAME.
//* LAB USERS:  For instance: SHAREC05 -> SHARC05 above for "SHARxx
//* Copy the REXX exec below into a data set
//CREATE    EXEC  PGM=IEBGENER
//SYSIN     DD  DUMMY
//SYSPRINT  DD  SYSOUT=A,HOLD=YES
//SYSUT2    DD  DSN=&SYSUID..TEMPREX(REXTSO),DISP=(,PASS),
// SPACE=(CYL,(1,1,1)),UNIT=3390,
// DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSUT1    DD   *,DLM=##
/* REXX */

/* START OF SPECIFICATIONS ****************************************
/*
/*----------------------------------------------------------------
/*   PROPRIETARY STATEMENT
/*
/*   LICENSED MATERIALS - PROPERTY OF IBM
```

You are now ready to edit our first program. This REXX exec takes advantage of the toolkit interfaces to interact with the geo services REST API and will be the basis of our first exercise.

To make it easier to view more at once on the screen, you may wish to hit the **maximize** button.

REXX programs can be run as batch programs in our two examples today instead of executed in the user's TSO/E address space. To customize the job, please modify the job card. This will allow you to find your job output easier as the jobname will match your userid. Change the very first line to represent your userid given to you. For example, if you were userid SHARA15, change **SHARxxx** to **SHARA15**.



SHARA15.ZOSREST.LAB(RXEXEC1)

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---->80
//SHARxxx JOB 'IKJEFT01 REXX',CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//* LAB USERS:  Remember to change the xxx above to the last 3 chars
//* LAB USERS:  of your assigned userid in the JOBNAME.
//* LAB USERS:  For instance: SHAREC05 -> SHARC05 above for "SHARxxx"
//* Copy the REXX exec below into a data set
//CREATE    EXEC  PGM=IEBGENER
//SYSIN     DD  DUMMY
//SYSPRINT  DD  SYSOUT=A,HOLD=YES
//SYSUT2    DD  DSN=&SYSUID..TEMPREX(REXTSO),DISP=(,PASS),
// SPACE=(CYL,(1,1,1)),UNIT=3390,
// DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSUT1    DD   *,DLM=##
/* REXX */

/* START OF SPECIFICATIONS ********************************************/
/*                                                                    */
/*------------------------------------------------------------------- */
/*   PROPRIETARY STATEMENT                                            */
/*                                                                    */
/*   LICENSED MATERIALS - PROPERTY OF IBM                             */
/*   5650-ZOS COPYRIGHT IBM CORP. 2015,2019                           */
/*   STATUS=HBB77B0                                                   */
/*                                                                    */
/*   END_OF_PROPRIETARY_STATEMENT                                     */
/*--------------------------------------------------------------------*/
/*                                                                    */
/*   THIS SAMPLE PROGRAM IS PROVIDED FOR TUTORIAL PURPOSES ONLY. A    */
/*   COMPLETE HANDLING OF ERROR CONDITIONS HAS NOT BEEN SHOWN OR      */
/*   ATTEMPTED, AND THIS PROGRAM HAS NOT BEEN SUBMITTED TO FORMAL     */
/*   IBM TESTING.  THIS PROGRAM IS DISTRIBUTED ON AN 'AS IS' BASIS    */
/*   WITHOUT ANY WARRANTIES EITHER EXPRESSED OR IMPLIED.              */
/*                                                                    */
/*--------------------------------------------------------------------*/
/*                                                                    */
/*  SCRIPT NAME=RXEXEC1                                               */
/*                                                                    */
/*    DESCRIPTIVE NAME=Sample Rexx code to use HTTP services          */
/*                     in the z/OS Client Web Enablement Toolkit      */
/*                     Based on HWTHXRX1 sample shipped in samplib    */
```

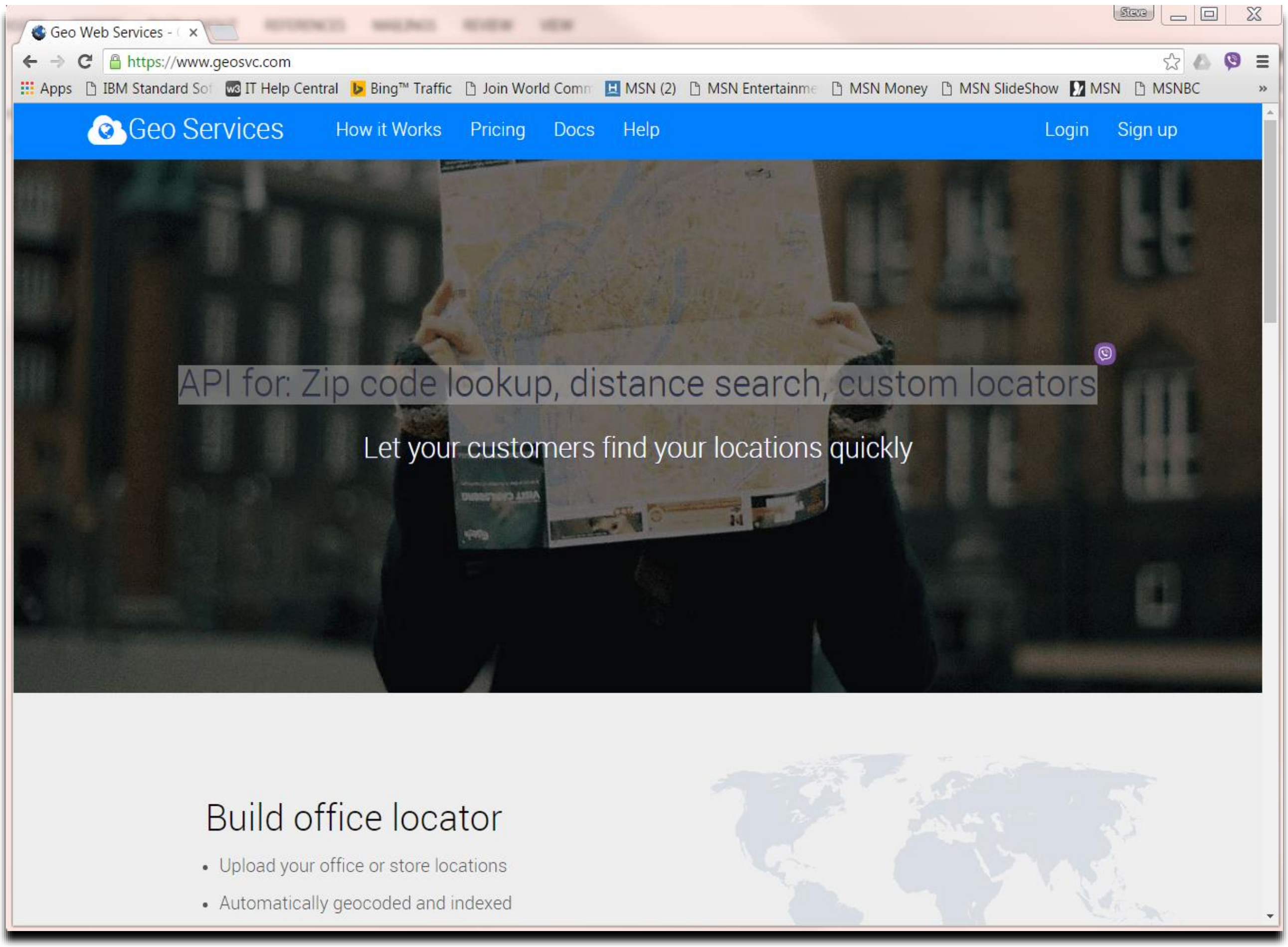
2. Learn how to find a published REST API and to gather the necessary information

Now that we are logged into the system let us first retrieve a public key for use with the RXEXEC1. This key will allow us to use this REST API server.

REST APIs can be found from a number of different locations, including searching on the internet, internal company-published APIs, and API directories. We are going to use the following website.

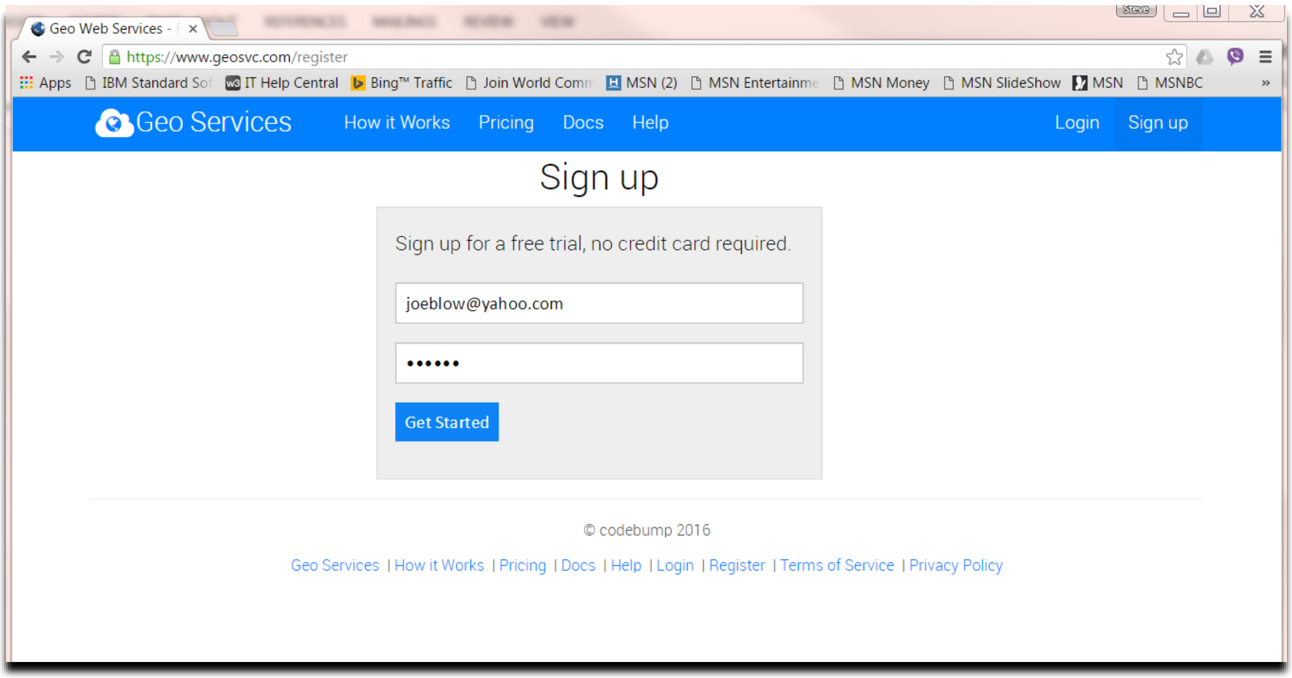
Launch a web browser and type the following as a URI (URL) http://geosvc.com

Geo Services is a published REST API that allows applications to find distances between two points on the map, and to get geographical information about a municipality and its zip codes, area codes, and latitude/longitude coordinates.



In order to regulate how much a single user can use their REST API server in a day, it is required that each user register with the site to obtain a public key. Click **Sign up** in the upper right hand corner.

The only personal information is an email address, which is not required of the website to even be a verified id. After typing an email address and password, click "Get Started"


Geo Web Services
https://www.geosvc.com/register
Apps | IBM Standard Sof | IT Help Central | Bing™ Traffic | Join World Comm | MSN (2) | MSN Entertainme | MSN Money | MSN SlideShow | MSN | MSNBC
Geo Services | How it Works | Pricing | Docs | Help | Login | Sign up

Sign up

Sign up for a free trial, no credit card required.

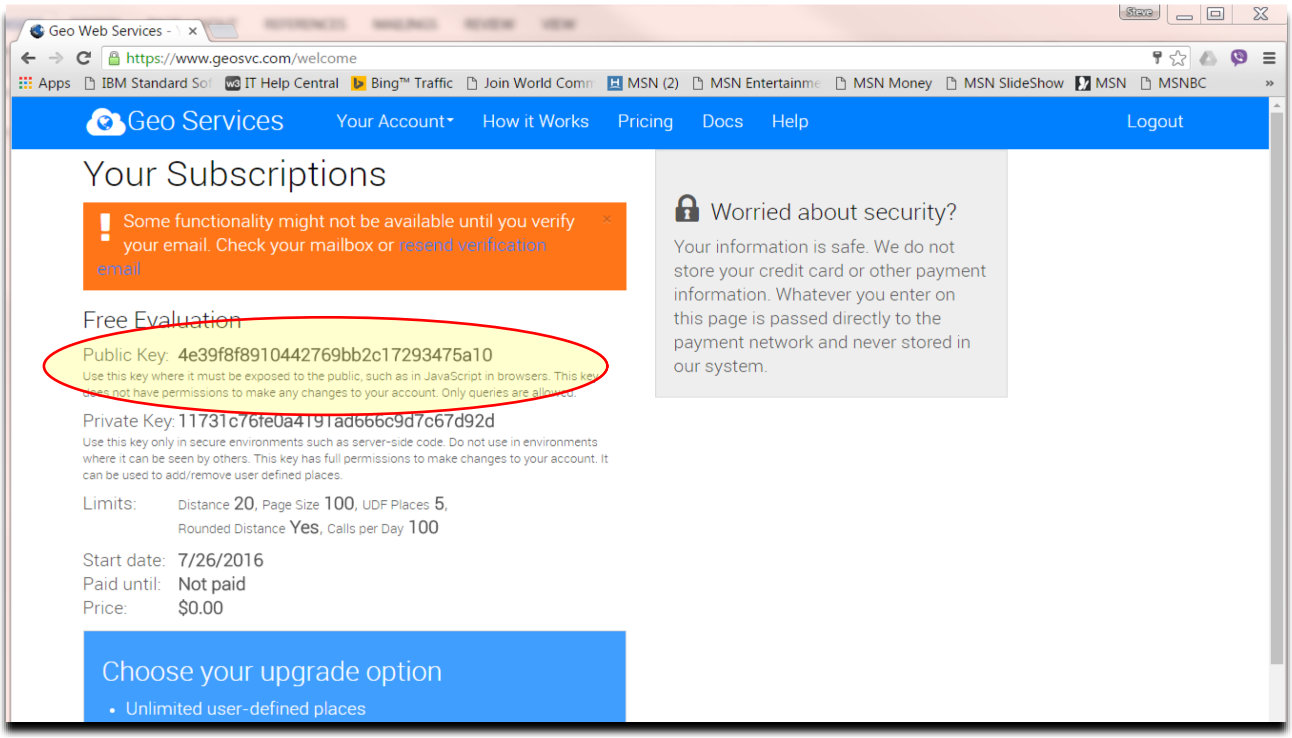
joeblow@yahoo.com

••••••

© codebump 2016

Geo Services | How it Works | Pricing | Docs | Help | Login | Register | Terms of Service | Privacy Policy

You are then given a public key which you will need on all API requests. There is a limit of 20 requests per day. Hopefully, you won't need that many to get it working. Save this public key either by cutting and pasting into a Notepad or into RXEXEC1 as a comment or write it down.


Geo Web Services
https://www.geosvc.com/welcome
Apps | IBM Standard Sof | IT Help Central | Bing™ Traffic | Join World Comm | MSN (2) | MSN Entertainme | MSN Money | MSN SlideShow | MSN | MSNBC
Geo Services | Your Account | How it Works | Pricing | Docs | Help | Logout

Your Subscriptions

Some functionality might not be available until you verify your email. Check your mailbox or resend verification email

Free Evaluation

Public Key: 4e39f8f8910442769bb2c17293475a10
Use this key where it must be exposed to the public, such as in JavaScript in browsers. This key does not have permissions to make any changes to your account. Only queries are allowed.

Private Key: 11731c76fe0a4191ad666c9d7c67d92d
Use this key only in secure environments such as server-side code. Do not use in environments where it can be seen by others. This key has full permissions to make changes to your account. It can be used to add/remove user defined places.

Limits: Distance 20, Page Size 100, UDF Places 5, Rounded Distance Yes, Calls per Day 100

Start date: 7/26/2016
Paid until: Not paid
Price: $0.00

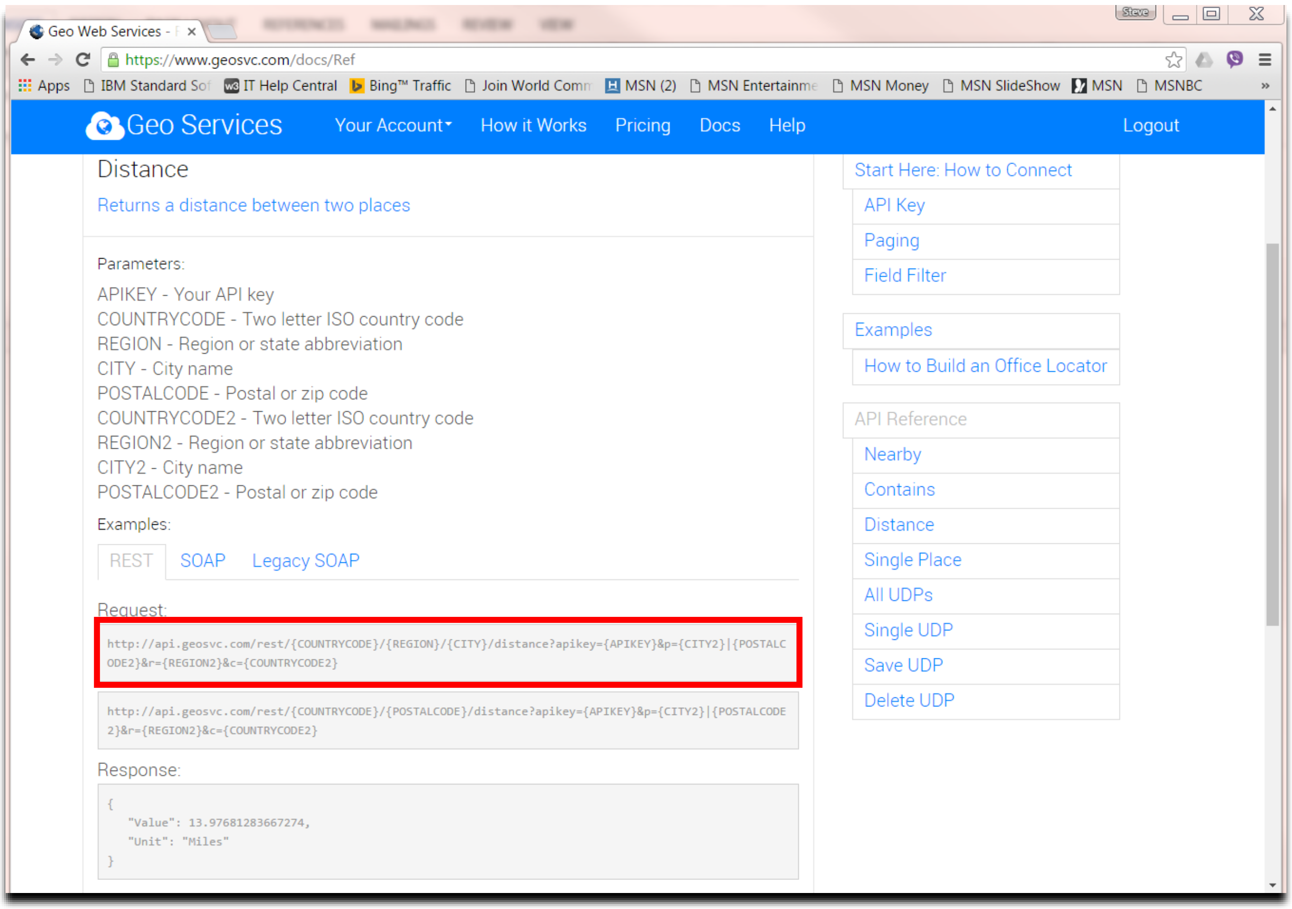
Choose your upgrade option

- Unlimited user-defined places

Worried about security?

Your information is safe. We do not store your credit card or other payment information. Whatever you enter on this page is passed directly to the payment network and never stored in our system.

Find the format of the URI that is to be sent for the distance API (find distance between two points). Click on the "**Docs**" button (top) on the website. Click on "**API Reference**" (left)and then on "**Distance**" (on left-hand side to expand so you can see **Parameters:" and other details**).



Distance

Returns a distance between two places

Parameters:

APIKEY - Your API key
COUNTRYCODE - Two letter ISO country code
REGION - Region or state abbreviation
CITY - City name
POSTALCODE - Postal or zip code
COUNTRYCODE2 - Two letter ISO country code
REGION2 - Region or state abbreviation
CITY2 - City name
POSTALCODE2 - Postal or zip code

Examples:

Request:

```
http://api.geosvc.com/rest/{COUNTRYCODE}/{REGION}/{CITY}/distance?apikey={APIKEY}&p={CITY2}|{POSTALC
ODE2}&r={REGION2}&c={COUNTRYCODE2}
```

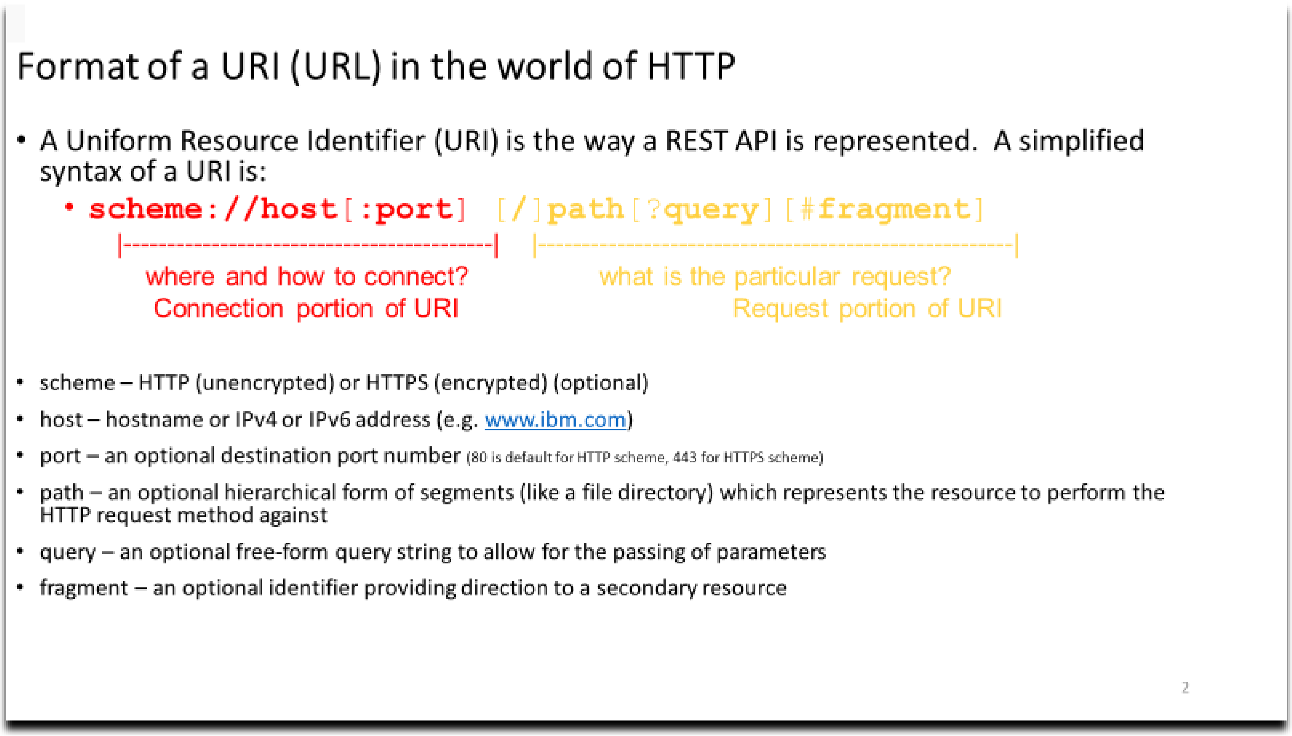
```
http://api.geosvc.com/rest/{COUNTRYCODE}/{POSTALCODE}/distance?apikey={APIKEY}&p={CITY2}|{POSTALCODE
2}&r={REGION2}&c={COUNTRYCODE2}
```

Response:

```
{
   "Value": 13.97681283667274,
   "Unit": "Miles"
}
```

Here is just about everything we need to write our z/OS application. Gather the required information to be entered into the z/OS Client Web Enablement Toolkit REXX exec. Make a note of the following:

- URI (use the REGION URI, instead of the POSTAL CODE URI):
  ____
  (everything up to the question mark)
- HTTP Method: _**GET** (This isn't explicitly stated for this REST API so this is provided for you.)
- Any parameters required: ______________________________
- Response format expected: **JSON** (for this lab). (This information is provided on a different page, but no need to go there for now. Assume we want the data returned in JSON format for this REST API.)



Format of a URI (URL) in the world of HTTP

- A Uniform Resource Identifier (URI) is the way a REST API is represented. A simplified syntax of a URI is:
  - `scheme://host[:port] [/]path[?query][#fragment]`

```
|-----------------------------------------|  |----------------------------------------------------|
      where and how to connect?                       what is the particular request?
      Connection portion of URI                                  Request portion of URI
```

- scheme – HTTP (unencrypted) or HTTPS (encrypted) (optional)
- host – hostname or IPv4 or IPv6 address (e.g. www.ibm.com)
- port – an optional destination port number (80 is default for HTTP scheme, 443 for HTTPS scheme)
- path – an optional hierarchical form of segments (like a file directory) which represents the resource to perform the HTTP request method against
- query – an optional free-form query string to allow for the passing of parameters
- fragment – an optional identifier providing direction to a secondary resource

2

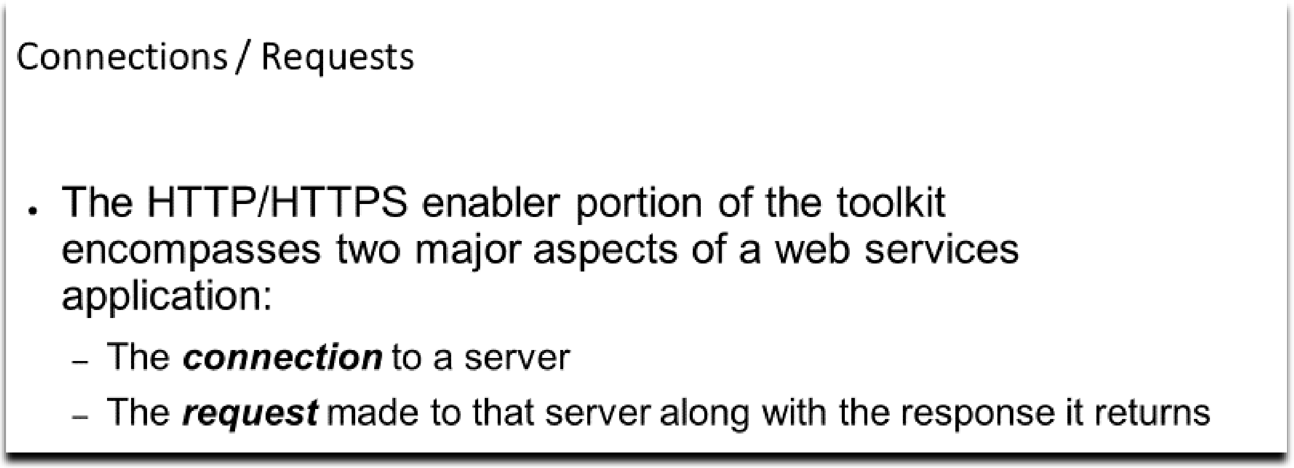
Divide the URI into its two major parts:

Connect portion of URI: ____

Request portion of URI: ____

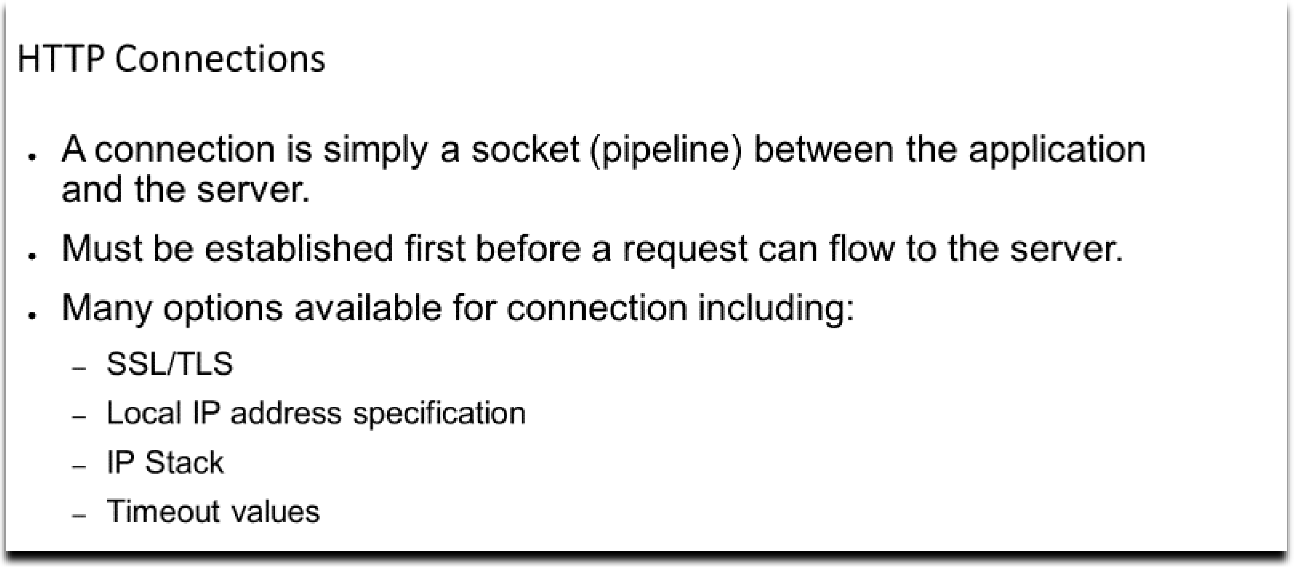
3. Learn how an HTTP toolkit application is laid out

Study the slides below to learn the organizational structure of how a z/OS Client Web Enablement Toolkit HTTP application is formed. Contextually, the information in the previous section about the correct formatting of inputs can be tied to the following information in that is shows which functions are called to execute the inputs.



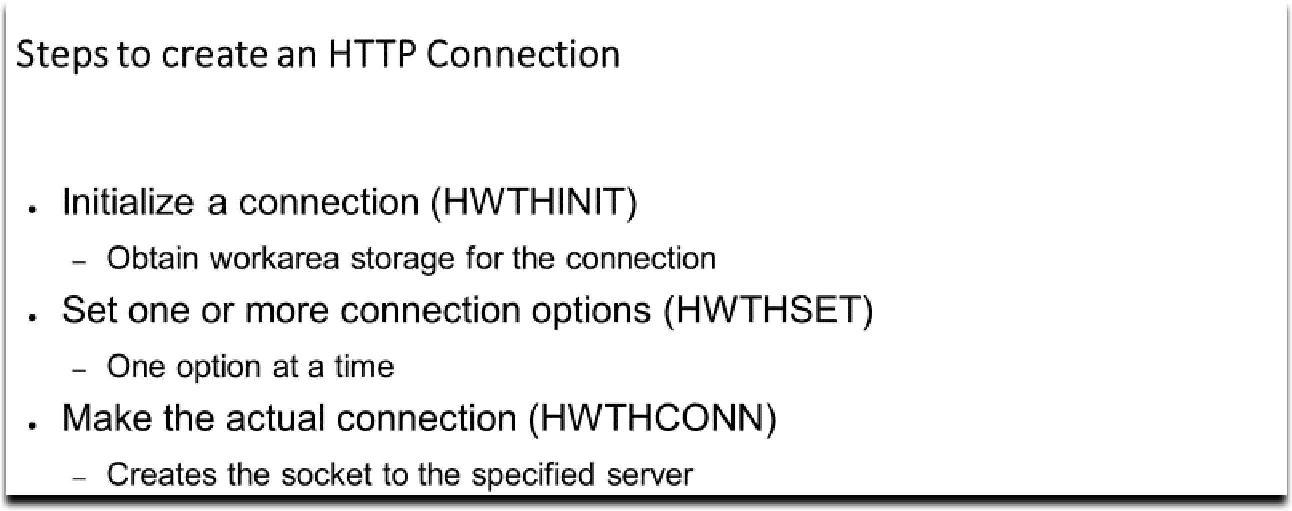
Connections / Requests

- The HTTP/HTTPS enabler portion of the toolkit encompasses two major aspects of a web services application:
  - The ***connection*** to a server
  - The ***request*** made to that server along with the response it returns



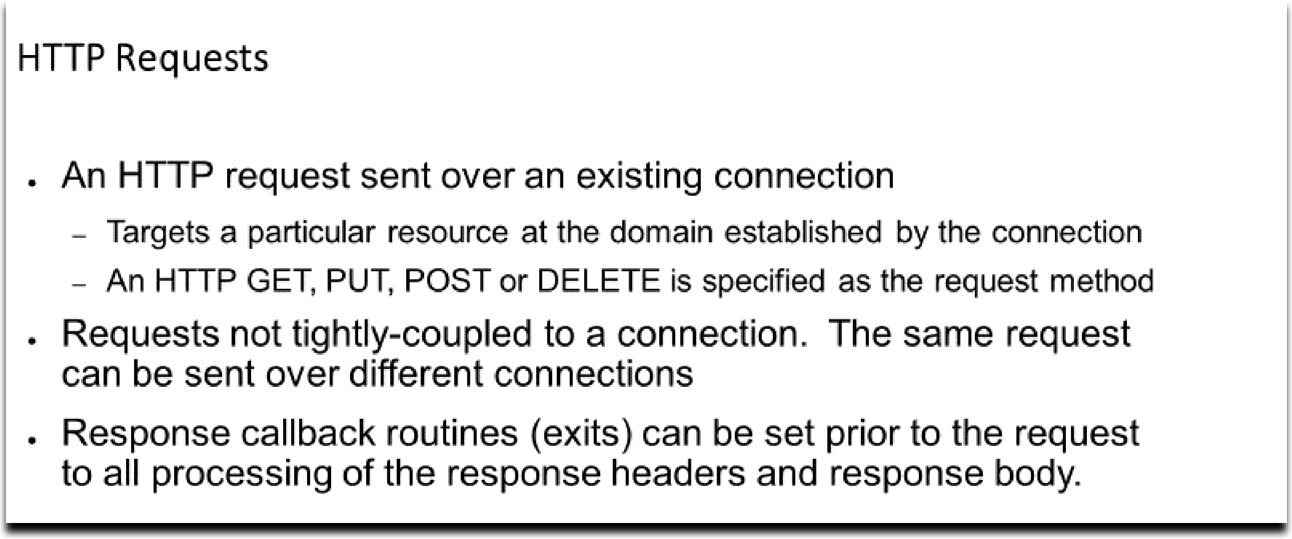
HTTP Connections

- A connection is simply a socket (pipeline) between the application and the server.
- Must be established first before a request can flow to the server.
- Many options available for connection including:
  - SSL/TLS
  - Local IP address specification
  - IP Stack
  - Timeout values



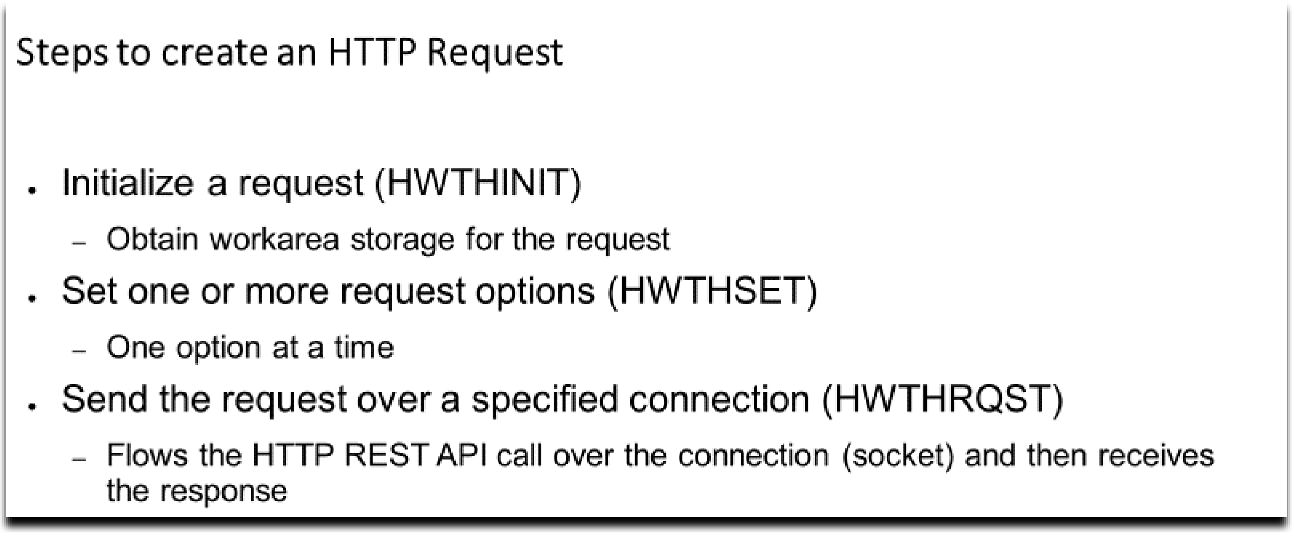
Steps to create an HTTP Connection

- Initialize a connection (HWTHINIT)
  - Obtain workarea storage for the connection
- Set one or more connection options (HWTHSET)
  - One option at a time
- Make the actual connection (HWTHCONN)
  - Creates the socket to the specified server



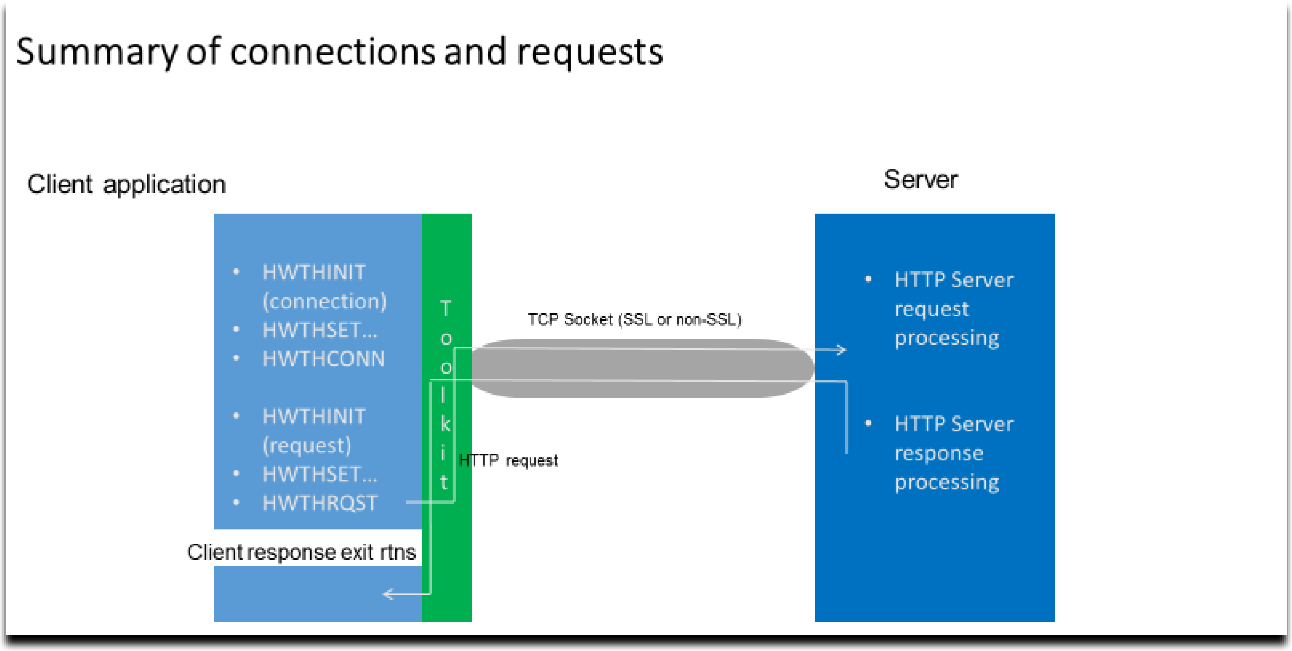
HTTP Requests

- An HTTP request sent over an existing connection
  - Targets a particular resource at the domain established by the connection
  - An HTTP GET, PUT, POST or DELETE is specified as the request method
- Requests not tightly-coupled to a connection. The same request can be sent over different connections
- Response callback routines (exits) can be set prior to the request to all processing of the response headers and response body.



Steps to create an HTTP Request

- Initialize a request (HWTHINIT)
  - Obtain workarea storage for the request
- Set one or more request options (HWTHSET)
  - One option at a time
- Send the request over a specified connection (HWTHRQST)
  - Flows the HTTP REST API call over the connection (socket) and then receives the response

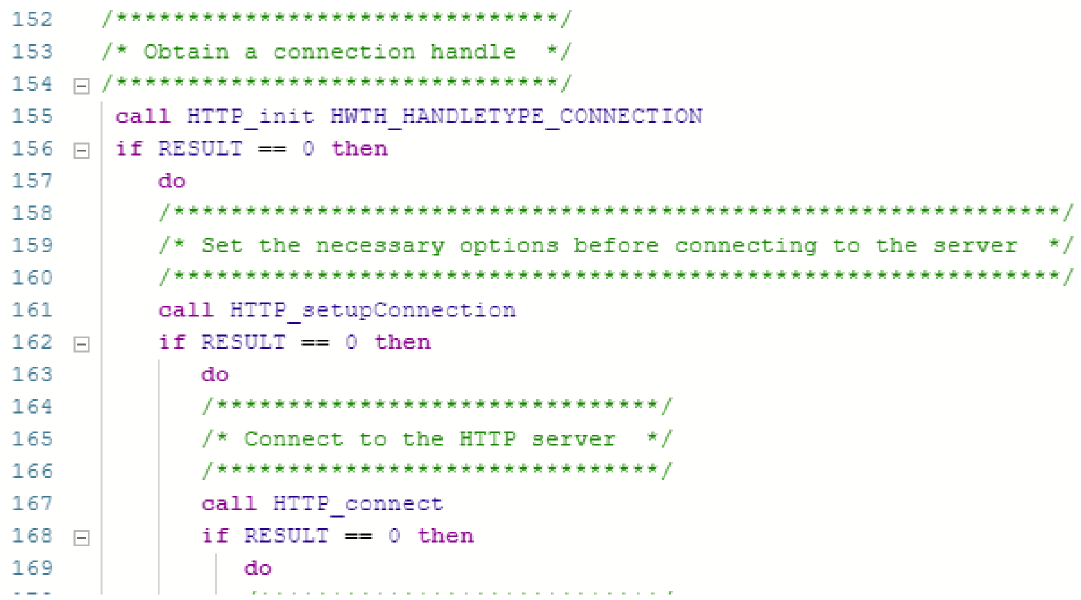


Summary of connections and requests

4. Specify the URIs in the toolkit application

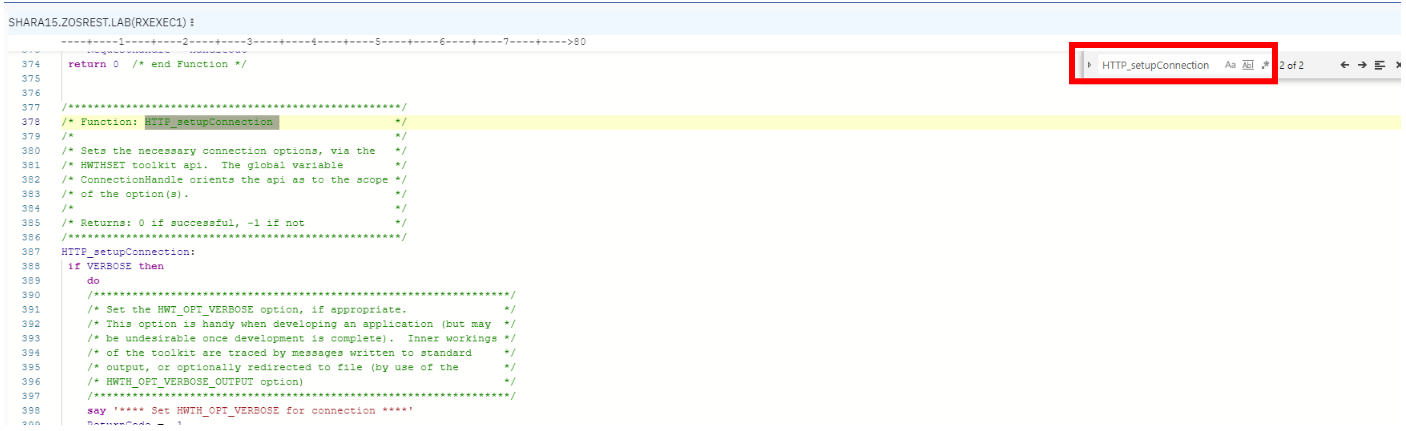
Go back to your z/OSMF edit session in your RXEXEC1 PDS member. Scroll down about 100 lines or so to find the mainline of the program. At about line 155, the toolkit portion of the application begins:

- HTTP_init subroutine initializes the connection
- HTTP_setupConnection sets all the options required by the connection
    - We will specify the connection portion on the OPT_URI connection option inside this subroutine
- HTTP_connect connects to the HTTP server (establishes a socket)



```
152   /********************************/
153   /* Obtain a connection handle  */
154   /********************************/
155    call HTTP_init HWTH_HANDLETYPE_CONNECTION
156    if RESULT == 0 then
157       do
158       /****************************************************************/
159       /* Set the necessary options before connecting to the server  */
160       /****************************************************************/
161       call HTTP_setupConnection
162       if RESULT == 0 then
163          do
164          /********************************/
165          /* Connect to the HTTP server  */
166          /********************************/
167          call HTTP_connect
168          if RESULT == 0 then
169             do
```

Find the HTTP_setupConnection subroutine by hitting **Ctrl-F** and typing **HTTP_setupConnection** in the find bar in the upper right of the edit session. It is around line 387.



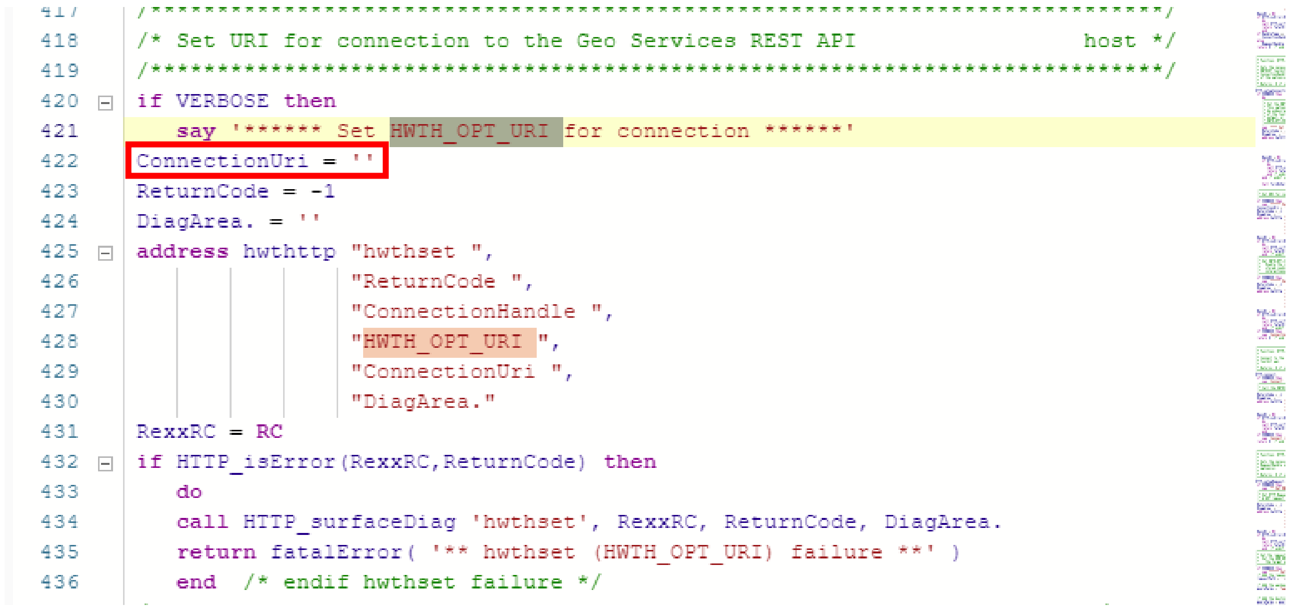
SHARA15.ZOSREST.LAB(RXEXEC1)

HTTP_setupConnection 2 of 2

```
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---->80
374  return 0  /* end Function */
375
376
377 /****************************************************/
378 /* Function: HTTP_setupConnection                   */
379 /*                                                  */
380 /* Sets the necessary connection options, via the   */
381 /* HWTHSET toolkit api.  The global variable        */
382 /* ConnectionHandle orients the api as to the scope */
383 /* of the option(s).                                */
384 /*                                                  */
385 /* Returns: 0 if successful, -1 if not              */
386 /****************************************************/
387 HTTP_setupConnection:
388  if VERBOSE then
389     do
390     /******************************************************************/
391     /* Set the HWT_OPT_VERBOSE option, if appropriate.                */
392     /* This option is handy when developing an application (but may  */
393     /* be undesirable once development is complete).  Inner workings */
394     /* of the toolkit are traced by messages written to standard      */
395     /* output, or optionally redirected to file (by use of the        */
396     /* HWTH_OPT_VERBOSE_OUTPUT option)                                */
397     /******************************************************************/
398     say '**** Set HWTH_OPT_VERBOSE for connection ****'
```

Find the call to HWTHSET that sets the target host/server using the **HWTH_OPT_URI** option:

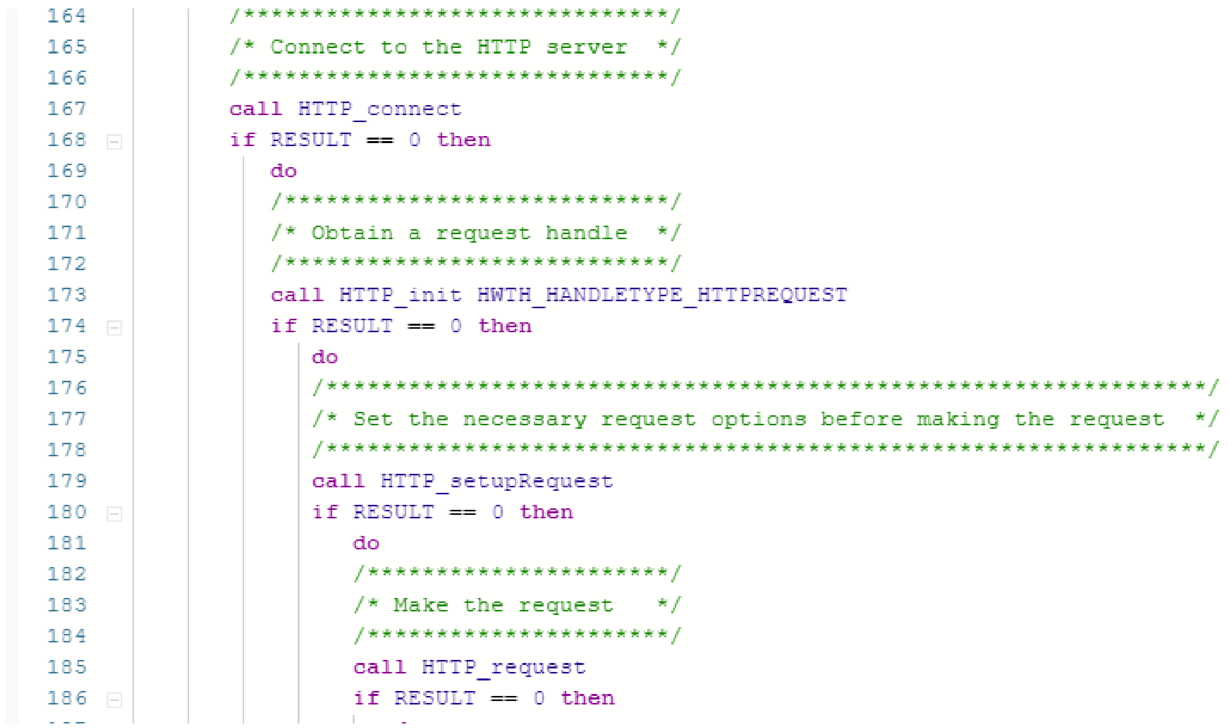
Right now the ConnectionUri REXX variable is set to nulls (‘’). Change this to the “Connect portion of the URI” that you deduced earlier. For example: ConnectionUri = ‘http://api.abcd.com’. (Note: See the answer key near the back of the handout if you need help).



```
/*************************************************************************/
/* Set URI for connection to the Geo Services REST API              host */
/*************************************************************************/
if VERBOSE then
   say '****** Set HWTH_OPT_URI for connection ******'
ConnectionUri = ''
ReturnCode = -1
DiagArea. = ''
address hwthttp "hwthset ",
                "ReturnCode ",
                "ConnectionHandle ",
                "HWTH_OPT_URI ",
                "ConnectionUri ",
                "DiagArea."
RexxRC = RC
if HTTP_isError(RexxRC,ReturnCode) then
   do
   call HTTP_surfaceDiag 'hwthset', RexxRC, ReturnCode, DiagArea.
   return fatalError( '** hwthset (HWTH_OPT_URI) failure **' )
   end  /* endif hwthset failure */
```

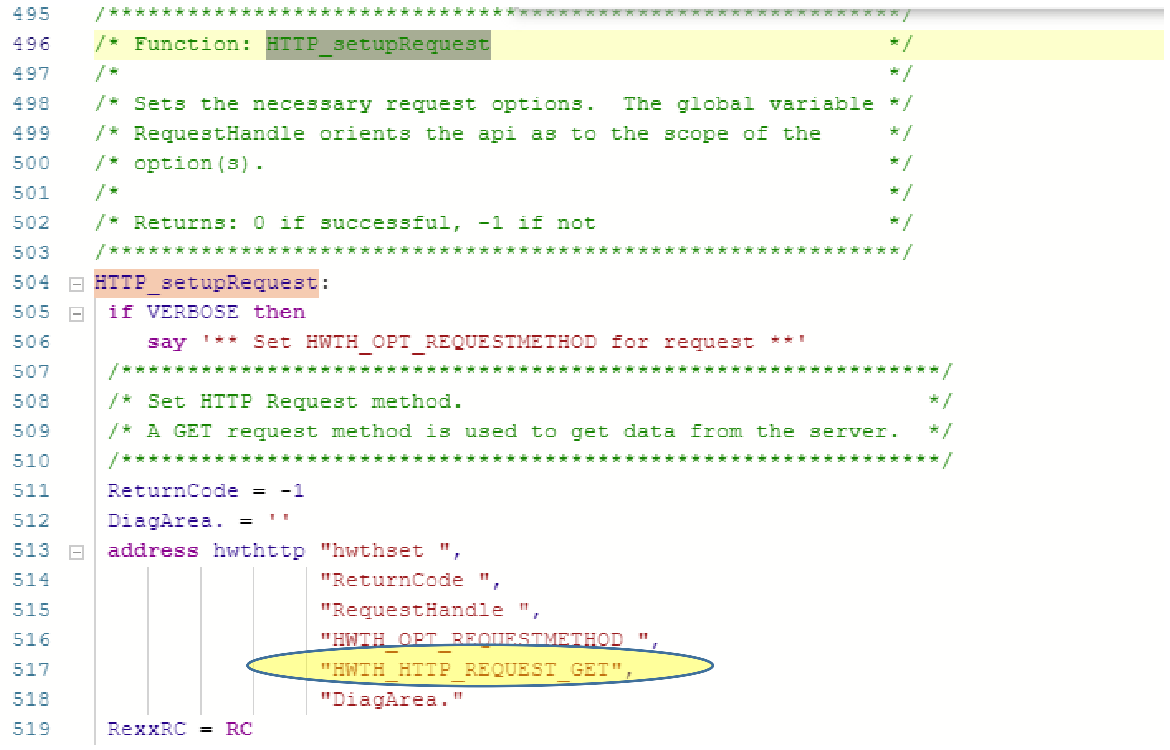
Now go back to the top of the exec. Scroll back down to the mainline for us to see more of this simple program. After a successful connect (HTTP_connect), the code does an:

- HTTP_init for a request (around line 173)
- HTTP_setupRequest to set the request options required (around line 179)
    - We will specify the request portion on the OPT_URI connection option inside this subroutine
- HTTP_request to send the HTTP request to the server (around line 185)



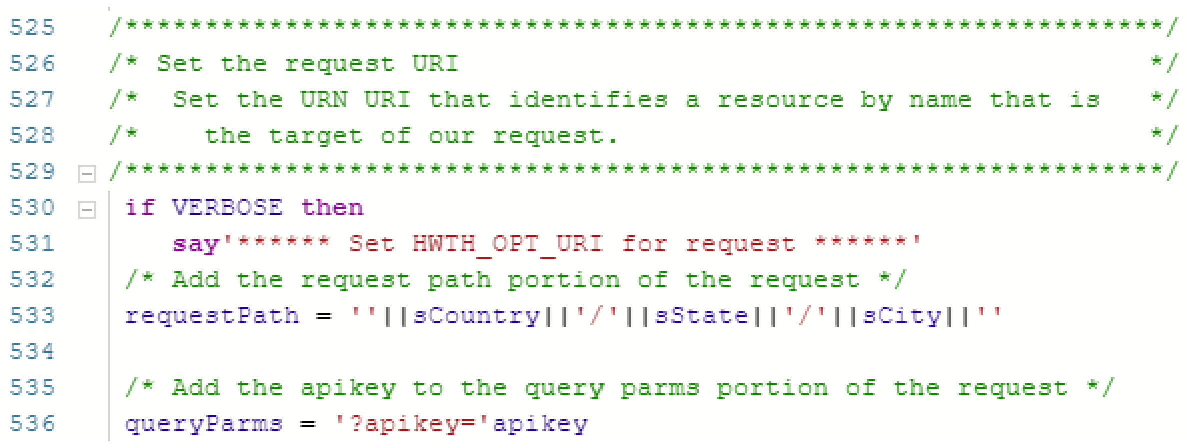
```
164        /********************************/
165        /* Connect to the HTTP server  */
166        /********************************/
167        call HTTP_connect
168        if RESULT == 0 then
169          do
170          /*****************************/
171          /* Obtain a request handle  */
172          /*****************************/
173          call HTTP_init HWTH_HANDLETYPE_HTTPREQUEST
174          if RESULT == 0 then
175            do
176            /******************************************************************/
177            /* Set the necessary request options before making the request  */
178            /******************************************************************/
179            call HTTP_setupRequest
180            if RESULT == 0 then
181              do
182              /***********************/
183              /* Make the request   */
184              /***********************/
185              call HTTP_request
186              if RESULT == 0 then
```

Find the **HTTP_setupRequest** subroutine. (You may need to hit the right arrow on the find bar a few times to find it (around line 496). The HWTH_OPT_REQUESTMETHOD value is set to GET which matches the required HTTP GET method for the service.



```
495  /***************************************************************/
496  /* Function: HTTP_setupRequest                                 */
497  /*                                                             */
498  /* Sets the necessary request options.  The global variable */
499  /* RequestHandle orients the api as to the scope of the     */
500  /* option(s).                                                  */
501  /*                                                             */
502  /* Returns: 0 if successful, -1 if not                         */
503  /***************************************************************/
504  HTTP_setupRequest:
505   if VERBOSE then
506      say '** Set HWTH_OPT_REQUESTMETHOD for request **'
507   /***************************************************************/
508   /* Set HTTP Request method.                                    */
509   /* A GET request method is used to get data from the server.  */
510   /***************************************************************/
511   ReturnCode = -1
512   DiagArea. = ''
513   address hwthttp "hwthset ",
514                   "ReturnCode ",
515                   "RequestHandle ",
516                   "HWTH_OPT_REQUESTMETHOD ",
517                   "HWTH_HTTP_REQUEST_GET",
518                   "DiagArea."
519   RexxRC = RC
```

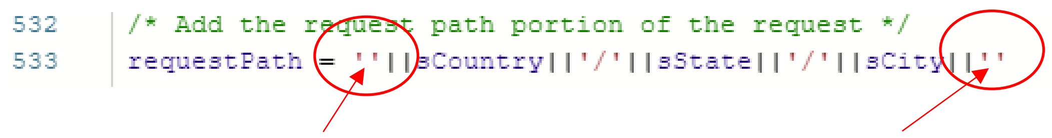
Find the call to HWTHSET that sets the request target URI using the **HWTH_OPT_URI** option (around line 533):



```
525  /*****************************************************************/
526  /* Set the request URI                                           */
527  /*  Set the URN URI that identifies a resource by name that is   */
528  /*    the target of our request.                                 */
529  /*****************************************************************/
530   if VERBOSE then
531      say'****** Set HWTH_OPT_URI for request ******'
532   /* Add the request path portion of the request */
533   requestPath = ''||sCountry||'/'||sState||'/'||sCity||''
534
535   /* Add the apikey to the query parms portion of the request */
536   queryParms = '?apikey='apikey
```

Right now, the requestPath REXX variable is missing two strings on either end (circled in red) . The required country, state, and city the code passed into the exec via the input variables and you don't need to touch those.

Change the requestPath to insert the correct "Request portion of the URI" that you deduced earlier in these two places.



```
/* Add the request path portion of the request */
requestPath = ''||sCountry||'/'||sState||'/'||sCity||''
```

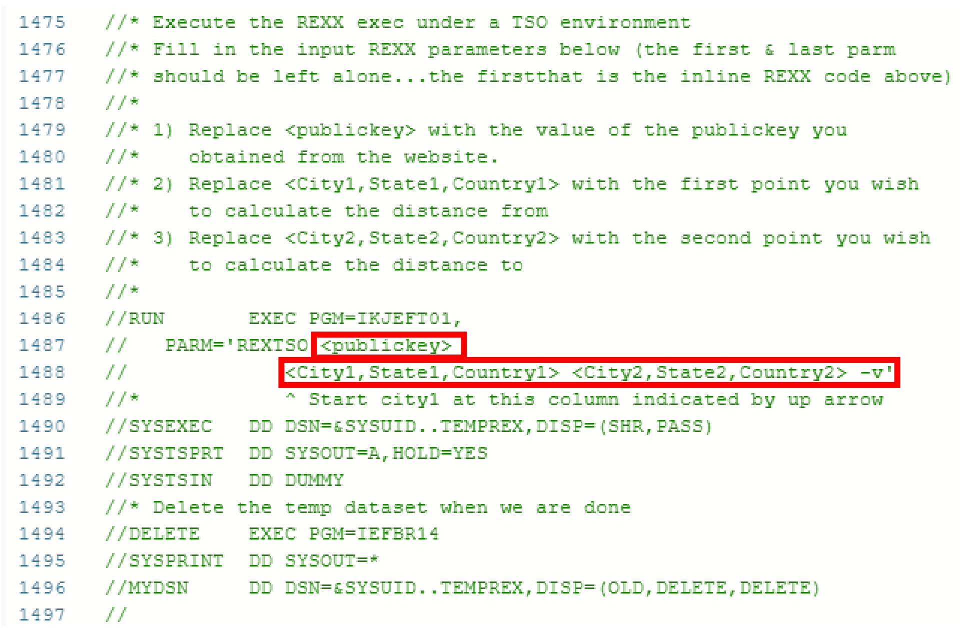
For example: requestPath = '*/INPUT1/*'||sCountry||'/'||sState||'/'||sCity||'***/INPUT2***' where:

- You fill in INPUT1, as the part you noted from the geosvc.com reference page. **:** **DON'T FORGET the leading and trailing "/" for INPUT1 .**
- You fill in INPUT2, as the part you noted from the geosvc.com reference page. **DON'T' FORGET the leading "/" for INPUT2.**
- Leave the middle part alone, but do notice the format and order of the values specified to ensure it follows the "Request portion of the URI" that you deduced. Remember, the apikey portion will be handled later in this exec.
- Keep the part you added in enclosed quotes.

(Note: See the answer key near the back of the handout if you need help).

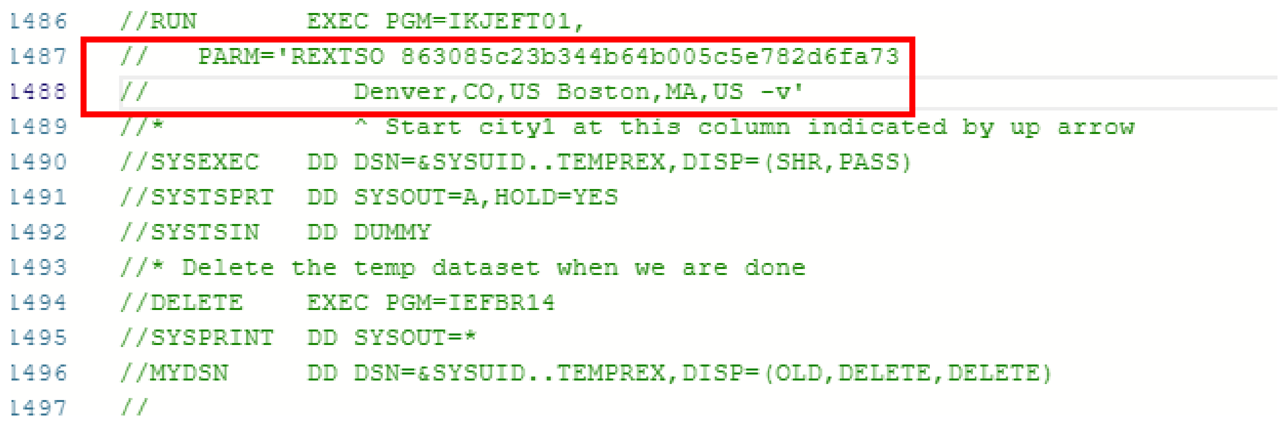
After you filled in the missing pieces in the requestPath, we are now ready to try executing this REXX exec. But before we save this file, we need to specify the input parameters. Normally, if invoking the REXX exec from TSO, we could specify the parameters at the time of invocation on the TSO EXECUTE command. However, we are going to submit this JCL to run the REXX exec. Therefore, we need to type the parameters in the JCL surrounding this REXX exec.

Go all the way to the bottom of the file. Follow the instructions in the JCL to replace the <publickey>, <City1,State1,Country1> and <City2,State2,Country2> with the appropriate fields. Make sure to leave a space between REXTSO and your public key that you cut and paste from the website. Also make sure that you start the City1 in column 16 as indicated.



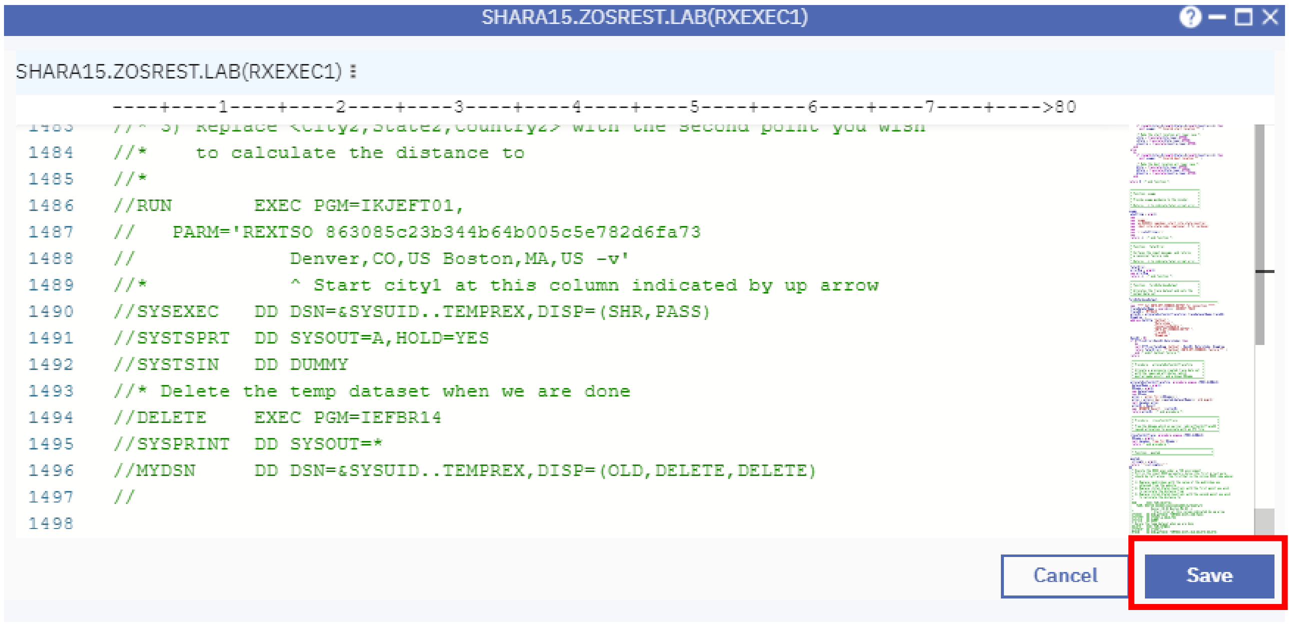
```
//* Execute the REXX exec under a TSO environment
//* Fill in the input REXX parameters below (the first & last parm
//* should be left alone...the firstthat is the inline REXX code above)
//*
//* 1) Replace <publickey> with the value of the publickey you
//*    obtained from the website.
//* 2) Replace <City1,State1,Country1> with the first point you wish
//*    to calculate the distance from
//* 3) Replace <City2,State2,Country2> with the second point you wish
//*    to calculate the distance to
//*
//RUN       EXEC PGM=IKJEFT01,
//   PARM='REXTSO <publickey>
//             <City1,State1,Country1> <City2,State2,Country2> -v'
//*            ^ Start city1 at this column indicated by up arrow
//SYSEXEC   DD DSN=&SYSUID..TEMPREX,DISP=(SHR,PASS)
//SYSTSPRT  DD SYSOUT=A,HOLD=YES
//SYSTSIN   DD DUMMY
//* Delete the temp dataset when we are done
//DELETE    EXEC PGM=IEFBR14
//SYSPRINT  DD SYSOUT=*
//MYDSN     DD DSN=&SYSUID..TEMPREX,DISP=(OLD,DELETE,DELETE)
//
```

Here is an example of how you might fill it in. Note that there are no spaces between city,state,country. Hint…for now, please choose a city name that has no spaces in it, a two letter state and the two letter country for the United States (US).



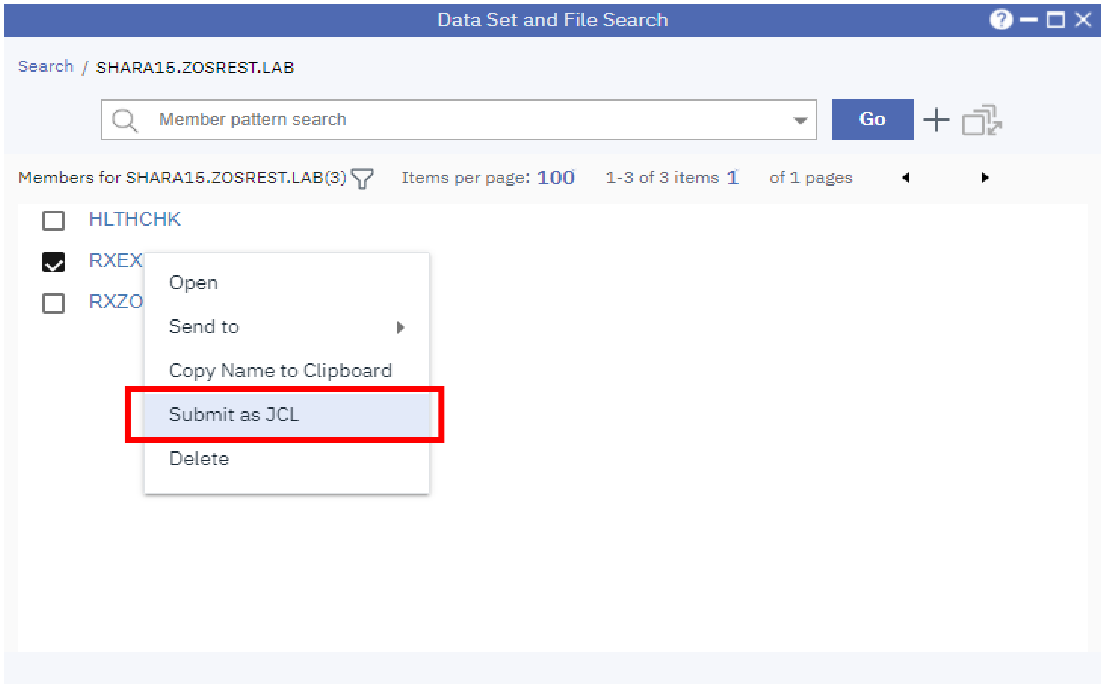
```
//RUN       EXEC PGM=IKJEFT01,
//   PARM='REXTSO 863085c23b344b64b005c5e782d6fa73
//             Denver,CO,US Boston,MA,US -v'
//*            ^ Start city1 at this column indicated by up arrow
//SYSEXEC   DD DSN=&SYSUID..TEMPREX,DISP=(SHR,PASS)
//SYSTSPRT  DD SYSOUT=A,HOLD=YES
//SYSTSIN   DD DUMMY
//* Delete the temp dataset when we are done
//DELETE    EXEC PGM=IEFBR14
//SYSPRINT  DD SYSOUT=*
//MYDSN     DD DSN=&SYSUID..TEMPREX,DISP=(OLD,DELETE,DELETE)
//
```

Now we want to save this file and try to run it. Click save in the bottom right.

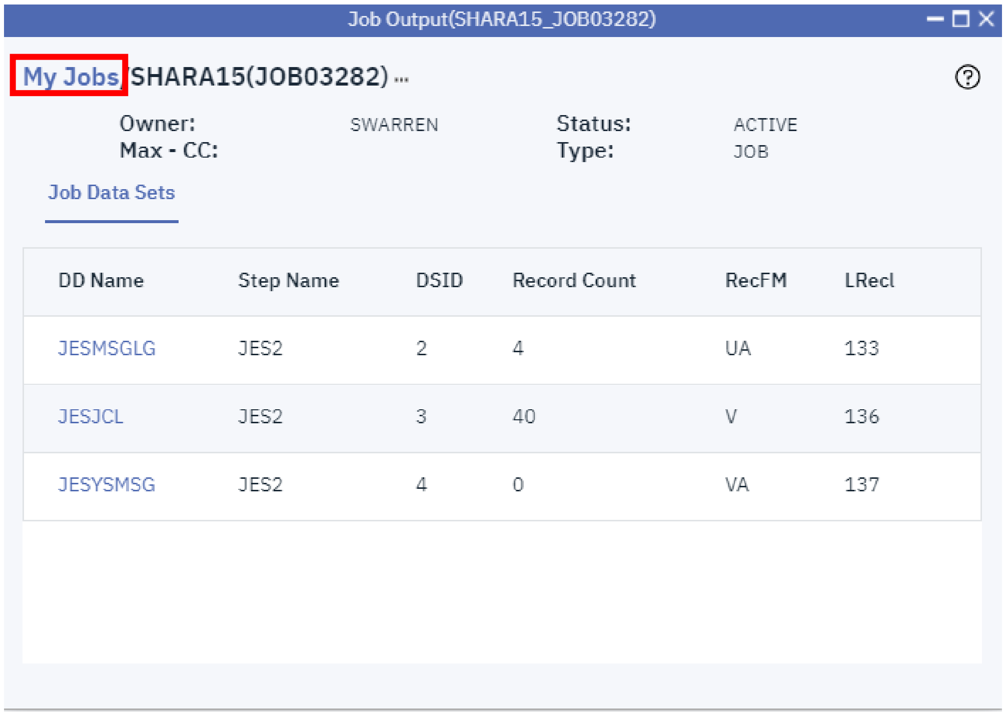


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---->80
//* 3) Replace <city2,state2,country2> with the second point you wish
//*    to calculate the distance to
//*
//RUN       EXEC PGM=IKJEFT01,
//     PARM='REXTSO 863085c23b344b64b005c5e782d6fa73
//               Denver,CO,US Boston,MA,US -v'
//*              ^ Start city1 at this column indicated by up arrow
//SYSEXEC   DD DSN=&SYSUID..TEMPREX,DISP=(SHR,PASS)
//SYSTSPRT  DD SYSOUT=A,HOLD=YES
//SYSTSIN   DD DUMMY
//* Delete the temp dataset when we are done
//DELETE    EXEC PGM=IEFBR14
//SYSPRINT  DD SYSOUT=*
//MYDSN     DD DSN=&SYSUID..TEMPREX,DISP=(OLD,DELETE,DELETE)
//
```

After getting back to the member list by X'ing the browse window, we are ready to run this baby. **Right click** on the **RXEXEC1** member name and select **Submit as JCL**.



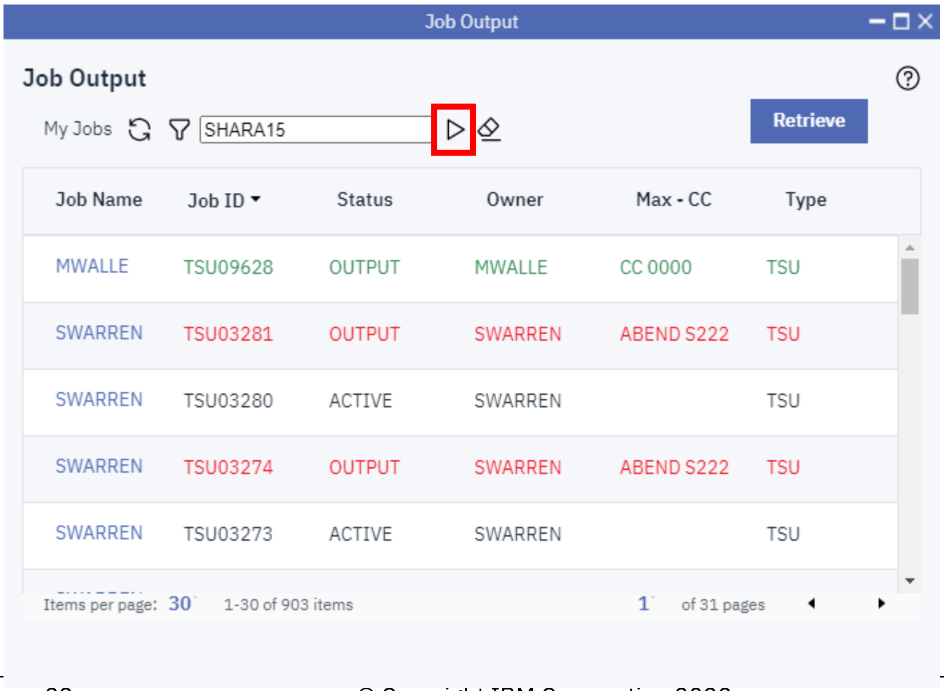
You will see the job output screen. If you have any JCL errors, it will show up here. You can click on the various DDs below to see any error messages that may have occurred. If we are ready to look at the job output, click on **My Jobs**:



Owner: SWARREN Status: ACTIVE
Max - CC: Type: JOB

DD Name	Step Name	DSID	Record Count	RecFM	LRecl
JESMSGLG	JES2	2	4	UA	133
JESJCL	JES2	3	40	V	136
JESYSMSG	JES2	4	0	VA	137

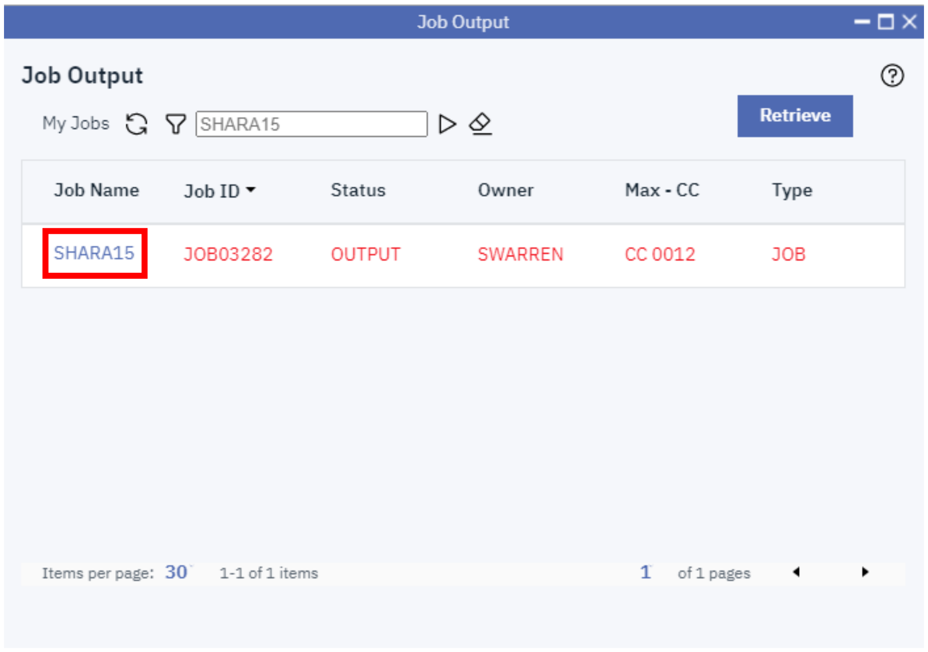
In the filter box, type the name of your JOB that you changed the JCL to: SHARxxx. (e.g. If your userid is SHARA15, then the jobname should also be SHARA15. Then click on the "Do Filter" icon to process the filter.



Job Name	Job ID	Status	Owner	Max - CC	Type
MWALLE	TSU09628	OUTPUT	MWALLE	CC 0000	TSU
SWARREN	TSU03281	OUTPUT	SWARREN	ABEND S222	TSU
SWARREN	TSU03280	ACTIVE	SWARREN		TSU
SWARREN	TSU03274	OUTPUT	SWARREN	ABEND S222	TSU
SWARREN	TSU03273	ACTIVE	SWARREN		TSU

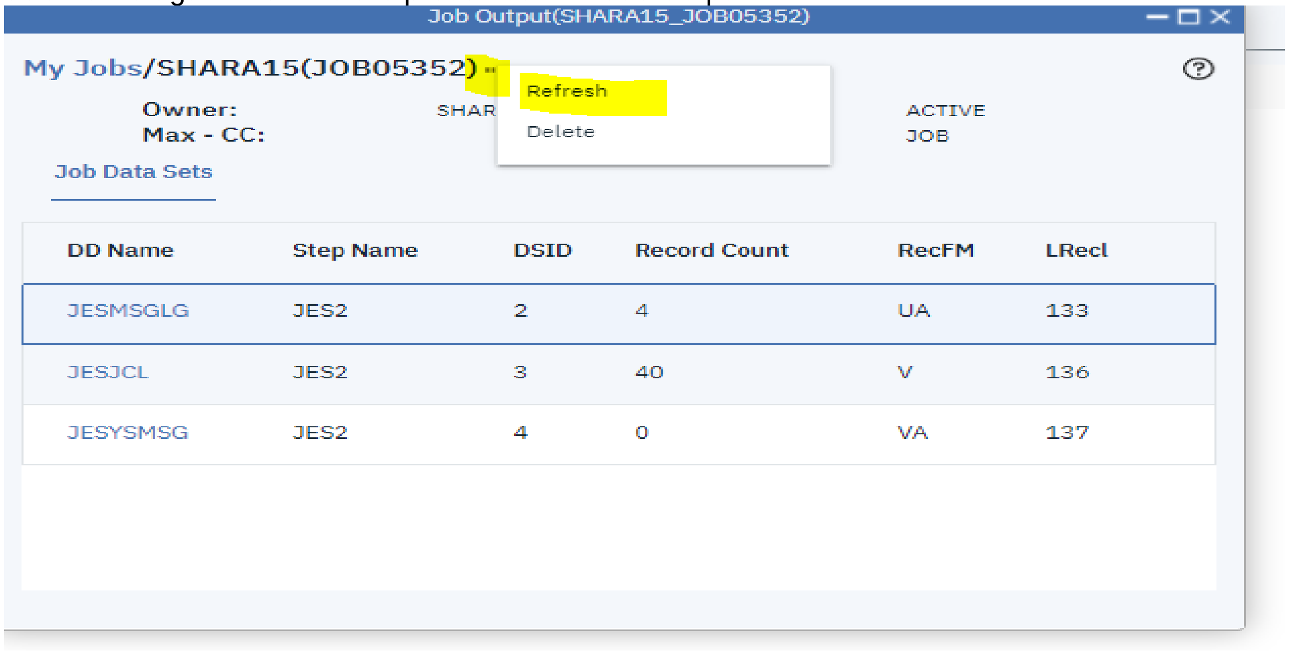
Items per page: 30 1-30 of 903 items 1 of 31 pages

Ok great, we found our job. Oh no! Max CC (condition code) is not 0000. It's 0012. Something didn't run quite right. Let's investigate what could be wrong. Click on the Job Name to investigate.



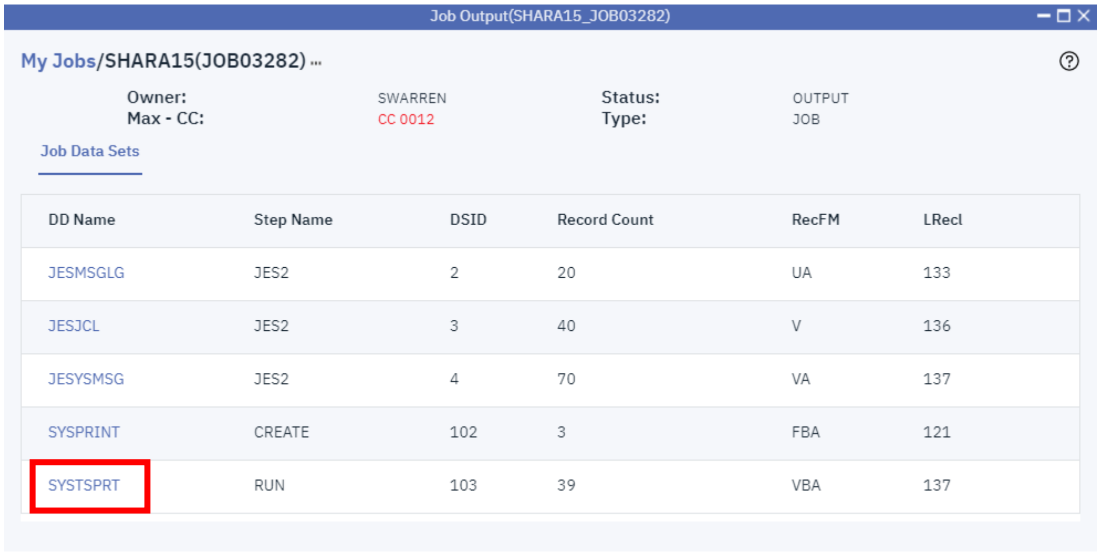
Job Name	Job ID	Status	Owner	Max - CC	Type
SHARA15	JOB03282	OUTPUT	SWARREN	CC 0012	JOB

Note if you don't see the SYSPRINT and SYSTSPRT DD names in the job output, click the refresh button at the top to populate the Job Output with these DD names (.. --> Refresh). Now click the **Job Name** again. Now we are presented with Job Output.



DD Name	Step Name	DSID	Record Count	RecFM	LRecl
JESMSGLG	JES2	2	4	UA	133
JESJCL	JES2	3	40	V	136
JESYSMSG	JES2	4	0	VA	137

Ok now let's see what went wrong. Click on the SYSTSPRT DD name to see the output from the REXX exec.



Job Output(SHARA15_JOB03282)

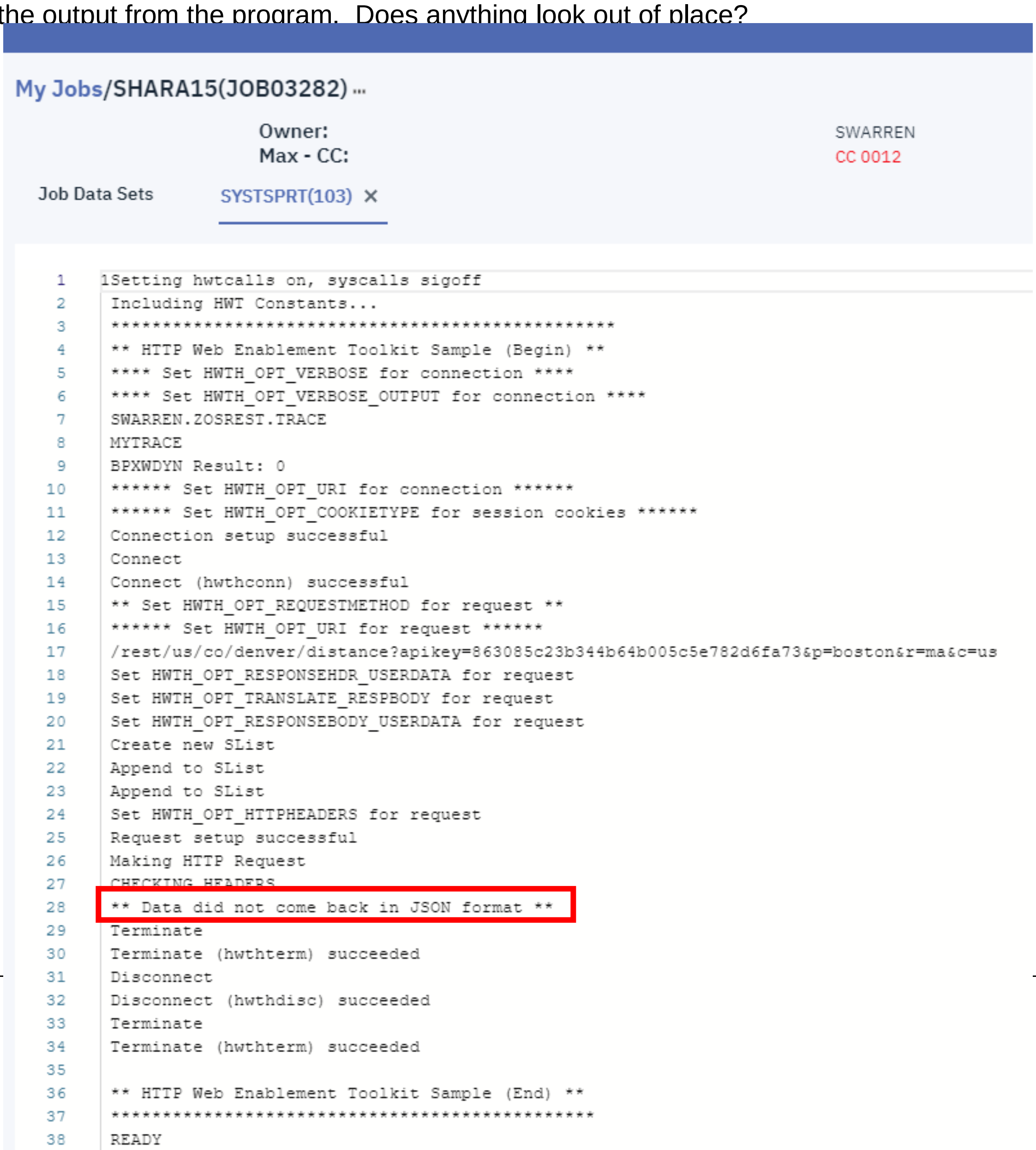
My Jobs/SHARA15(JOB03282) ...

Owner: SWARREN
Max - CC: CC 0012
Status: OUTPUT
Type: JOB

Job Data Sets

DD Name	Step Name	DSID	Record Count	RecFM	LRecl
JESMSGLG	JES2	2	20	UA	133
JESJCL	JES2	3	40	V	136
JESYSMSG	JES2	4	70	VA	137
SYSPRINT	CREATE	102	3	FBA	121
SYSTSPRT	RUN	103	39	VBA	137

Scan the output from the program. Does anything look out of place?



My Jobs/SHARA15(JOB03282) ...

Owner: SWARREN
Max - CC: CC 0012

Job Data Sets   SYSTSPRT(103) ✕

```
1Setting hwtcalls on, syscalls sigoff
 Including HWT Constants...
 *************************************************
 ** HTTP Web Enablement Toolkit Sample (Begin) **
 **** Set HWTH_OPT_VERBOSE for connection ****
 **** Set HWTH_OPT_VERBOSE_OUTPUT for connection ****
 SWARREN.ZOSREST.TRACE
 MYTRACE
 BPXWDYN Result: 0
 ****** Set HWTH_OPT_URI for connection ******
 ****** Set HWTH_OPT_COOKIETYPE for session cookies ******
 Connection setup successful
 Connect
 Connect (hwthconn) successful
 ** Set HWTH_OPT_REQUESTMETHOD for request **
 ****** Set HWTH_OPT_URI for request ******
 /rest/us/co/denver/distance?apikey=863085c23b344b64b005c5e782d6fa73&p=boston&r=ma&c=us
 Set HWTH_OPT_RESPONSEHDR_USERDATA for request
 Set HWTH_OPT_TRANSLATE_RESPBODY for request
 Set HWTH_OPT_RESPONSEBODY_USERDATA for request
 Create new SList
 Append to SList
 Append to SList
 Set HWTH_OPT_HTTPHEADERS for request
 Request setup successful
 Making HTTP Request
 CHECKING HEADERS
 ** Data did not come back in JSON format **
 Terminate
 Terminate (hwthterm) succeeded
 Disconnect
 Disconnect (hwthdisc) succeeded
 Terminate
 Terminate (hwthterm) succeeded

 ** HTTP Web Enablement Toolkit Sample (End) **
 ************************************************
 READY
```

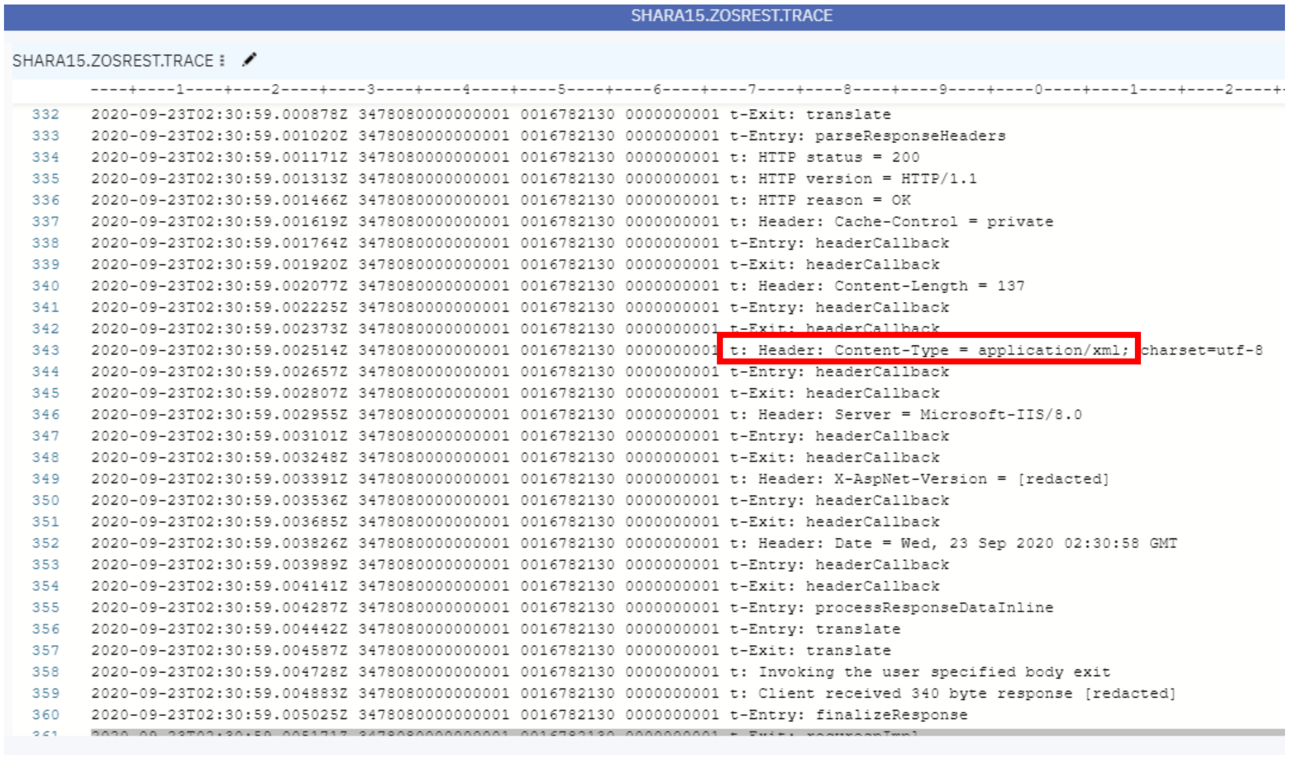
The program did not return the expected information… time to debug!

. Attempt to debug error in the program by tracing the output

Tracing in the HTTP portion of the toolkit is easy. By simply turning on the connection option HWTH_OPT_VERBOSE, this will cause the toolkit to trace its connection and data flows between us and the server. A secondary option HWTH_OPT_VERBOSE_OUTPUT allows this trace output to be captured to either a standard sequential MVS data set or a zFS file. This sample program has both of these options already turned on for us.

Let's go take a look at the trace data set. This particular sample allocates a previously defined sequential variable data set with the name the **'*labuserid*.ZOSREST.TRACE'** (Do you remember how to look at the contents of a data set? Look above if you forgot the steps.) *Trace output is appended to the end of this data set, so it will contain multiple "runs" of the REXX. Look at the bottom for the most recent!*

Sampling of the response coming back from the REST API server shows that the data coming back is not in the in the format we expected to parse as JSON. It's XML! Look at your trace data set:

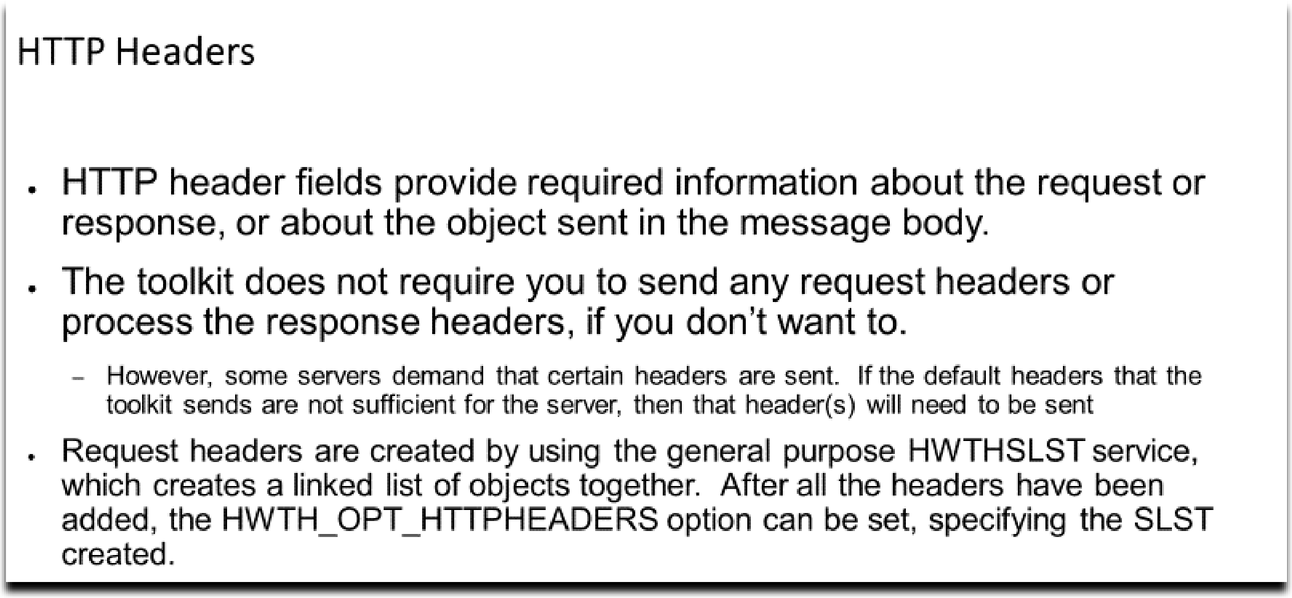


```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0----+----1----+----2----+
2020-09-23T02:30:59.000878Z 347808000000001 0016782130 0000000001 t-Exit: translate
2020-09-23T02:30:59.001020Z 347808000000001 0016782130 0000000001 t-Entry: parseResponseHeaders
2020-09-23T02:30:59.001171Z 347808000000001 0016782130 0000000001 t: HTTP status = 200
2020-09-23T02:30:59.001313Z 347808000000001 0016782130 0000000001 t: HTTP version = HTTP/1.1
2020-09-23T02:30:59.001466Z 347808000000001 0016782130 0000000001 t: HTTP reason = OK
2020-09-23T02:30:59.001619Z 347808000000001 0016782130 0000000001 t: Header: Cache-Control = private
2020-09-23T02:30:59.001764Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.001920Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.002077Z 347808000000001 0016782130 0000000001 t: Header: Content-Length = 137
2020-09-23T02:30:59.002225Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.002373Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.002514Z 347808000000001 0016782130 0000000001 t: Header: Content-Type = application/xml; charset=utf-8
2020-09-23T02:30:59.002657Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.002807Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.002955Z 347808000000001 0016782130 0000000001 t: Header: Server = Microsoft-IIS/8.0
2020-09-23T02:30:59.003101Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.003248Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.003391Z 347808000000001 0016782130 0000000001 t: Header: X-AspNet-Version = [redacted]
2020-09-23T02:30:59.003536Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.003685Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.003826Z 347808000000001 0016782130 0000000001 t: Header: Date = Wed, 23 Sep 2020 02:30:58 GMT
2020-09-23T02:30:59.003989Z 347808000000001 0016782130 0000000001 t-Entry: headerCallback
2020-09-23T02:30:59.004141Z 347808000000001 0016782130 0000000001 t-Exit: headerCallback
2020-09-23T02:30:59.004287Z 347808000000001 0016782130 0000000001 t-Entry: processResponseDataInline
2020-09-23T02:30:59.004442Z 347808000000001 0016782130 0000000001 t-Entry: translate
2020-09-23T02:30:59.004587Z 347808000000001 0016782130 0000000001 t-Exit: translate
2020-09-23T02:30:59.004728Z 347808000000001 0016782130 0000000001 t: Invoking the user specified body exit
2020-09-23T02:30:59.004883Z 347808000000001 0016782130 0000000001 t: Client received 340 byte response [redacted]
2020-09-23T02:30:59.005025Z 347808000000001 0016782130 0000000001 t-Entry: finalizeResponse
```

6. Learn how to specify an HTTP request header

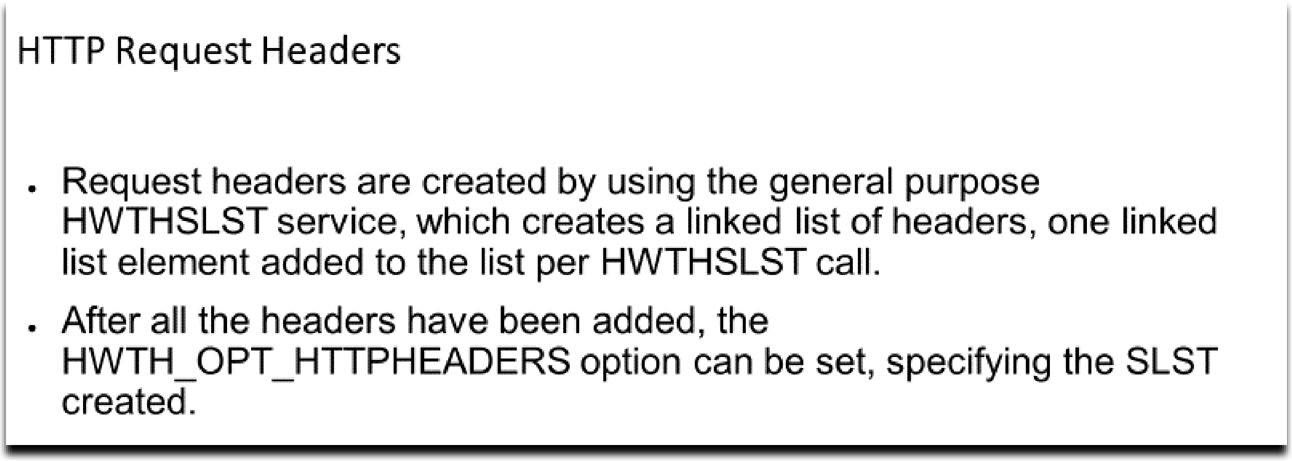
This data *could* be sent to z/OS XML System Services to be parsed, but we want to illustrate the JSON parsing available with the toolkit. We need to change the Accept request header to be JSON, since the website did say that it would return data back in either XML or JSON. (What you noted as “Response formats”.)

Let’s talk about headers and response bodies from the server first.



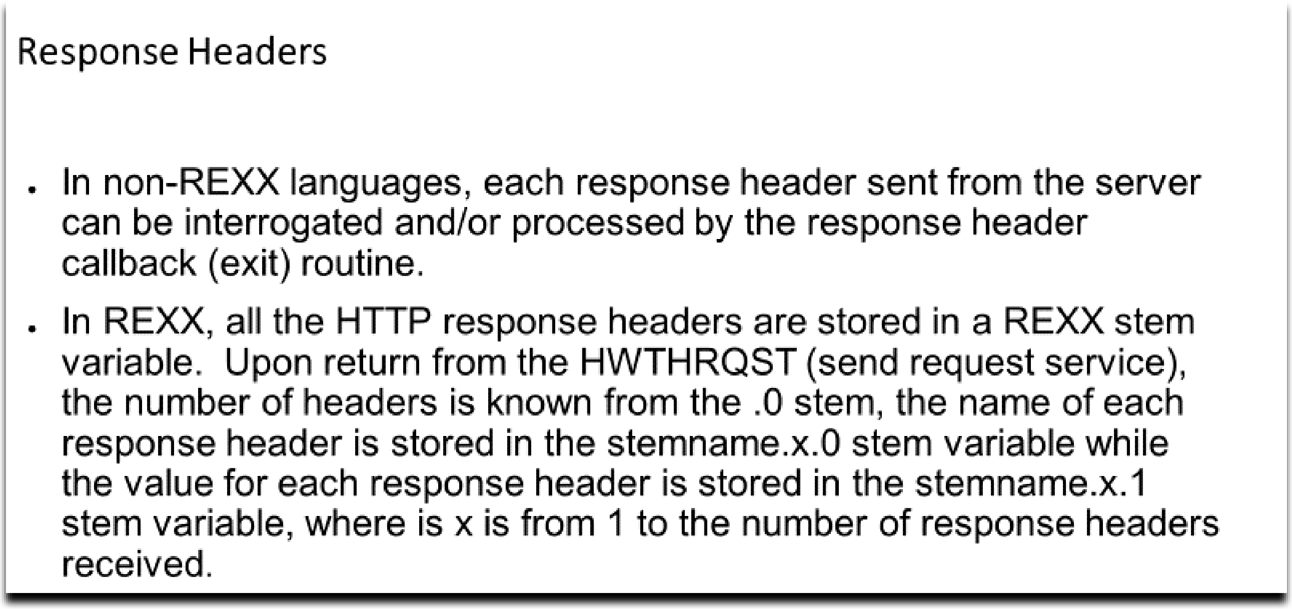
HTTP Headers

- HTTP header fields provide required information about the request or response, or about the object sent in the message body.
- The toolkit does not require you to send any request headers or process the response headers, if you don’t want to.
  - However, some servers demand that certain headers are sent. If the default headers that the toolkit sends are not sufficient for the server, then that header(s) will need to be sent
- Request headers are created by using the general purpose HWTHSLST service, which creates a linked list of objects together. After all the headers have been added, the HWTH_OPT_HTTPHEADERS option can be set, specifying the SLST created.



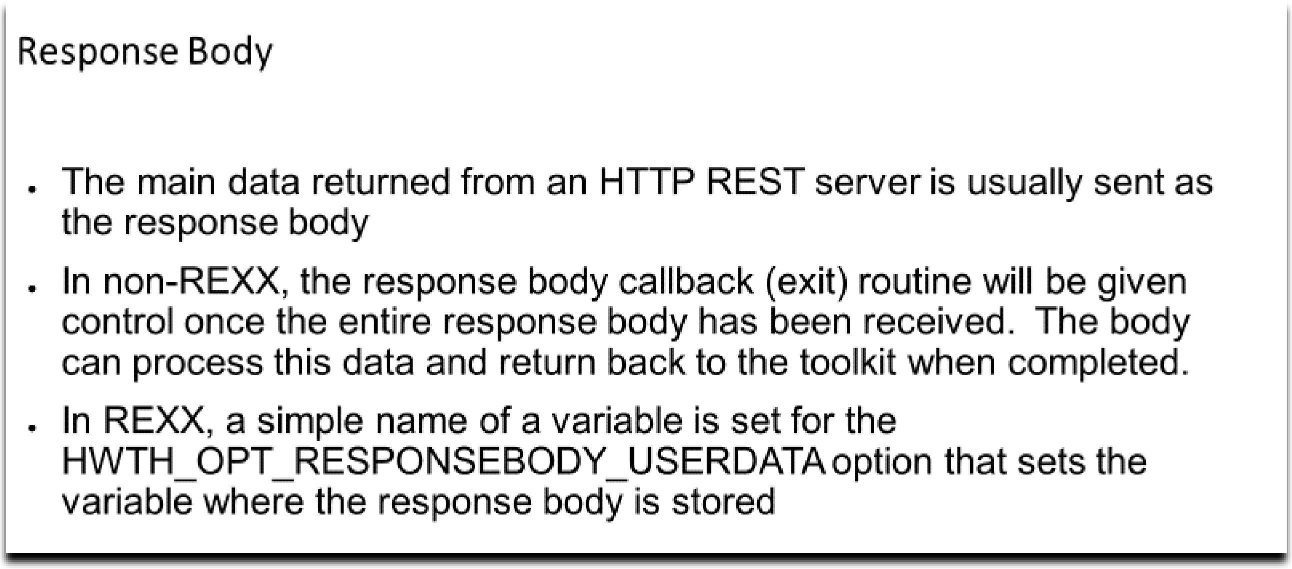
HTTP Request Headers

- Request headers are created by using the general purpose HWTHSLST service, which creates a linked list of headers, one linked list element added to the list per HWTHSLST call.
- After all the headers have been added, the HWTH_OPT_HTTPHEADERS option can be set, specifying the SLST created.



Response Headers

- In non-REXX languages, each response header sent from the server can be interrogated and/or processed by the response header callback (exit) routine.
- In REXX, all the HTTP response headers are stored in a REXX stem variable. Upon return from the HWTHRQST (send request service), the number of headers is known from the .0 stem, the name of each response header is stored in the stemname.x.0 stem variable while the value for each response header is stored in the stemname.x.1 stem variable, where is x is from 1 to the number of response headers received.

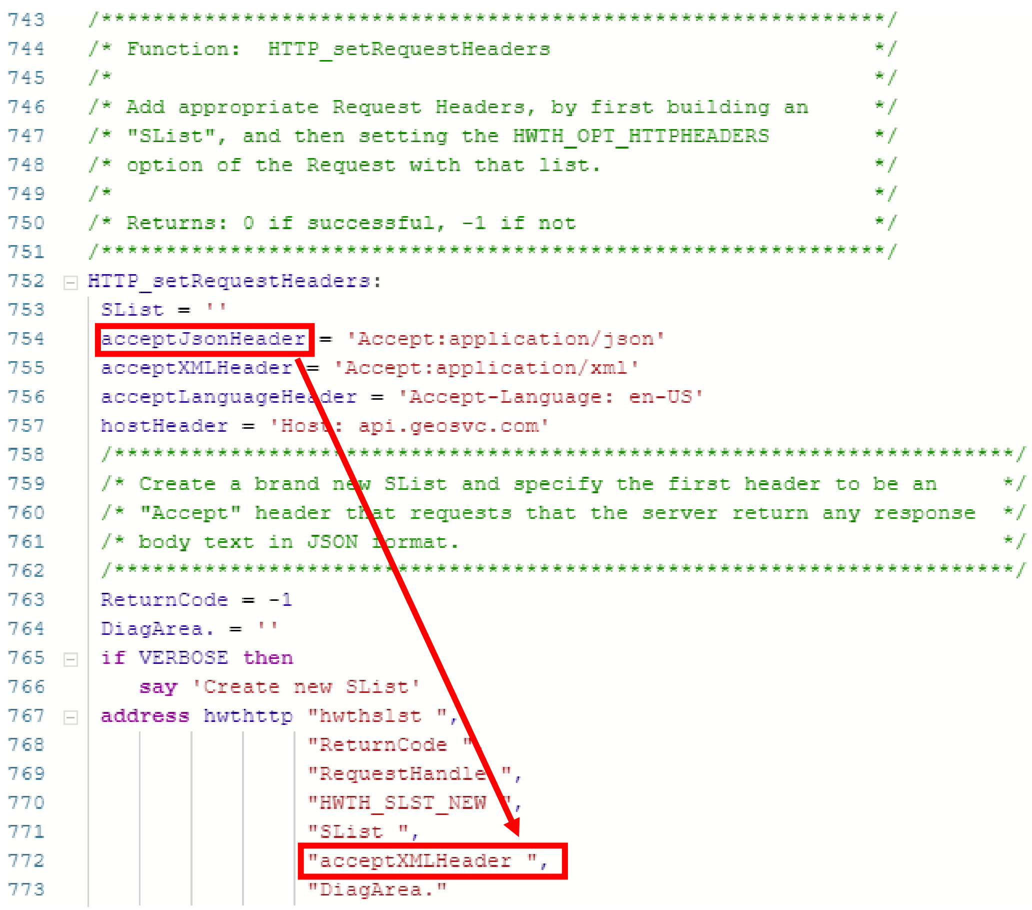


Response Body

- The main data returned from an HTTP REST server is usually sent as the response body
- In non-REXX, the response body callback (exit) routine will be given control once the entire response body has been received. The body can process this data and return back to the toolkit when completed.
- In REXX, a simple name of a variable is set for the HWTH_OPT_RESPONSEBODY_USERDATA option that sets the variable where the response body is stored

7. Fix the programming error

So to change the behavior of the server to return JSON instead of XML, we need to change the Accept header. Find the HTTP_setRequestHeaders subroutine (around line 741). Scroll down and find where the Accept Header is added to the linked list.

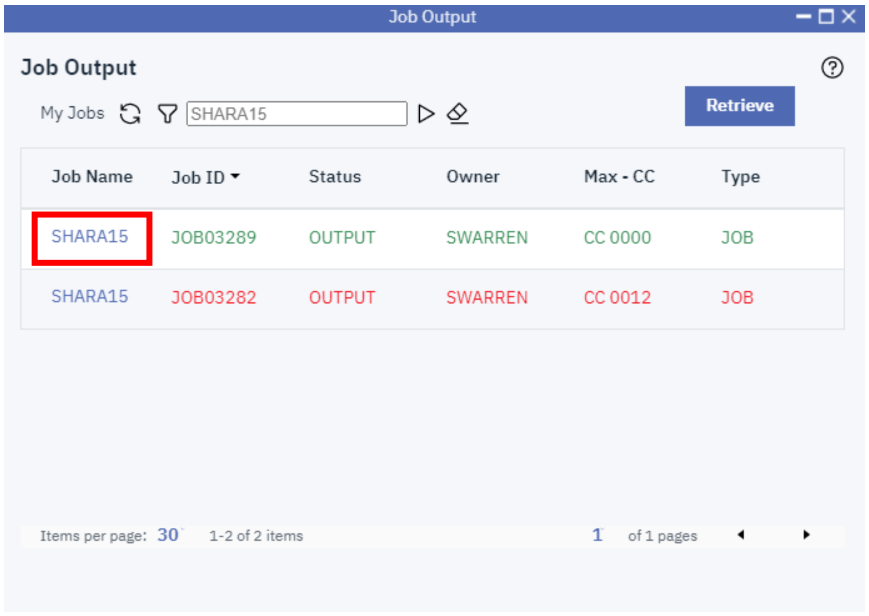


```
743  /*************************************************************/
744  /* Function:  HTTP_setRequestHeaders                         */
745  /*                                                           */
746  /* Add appropriate Request Headers, by first building an     */
747  /* "SList", and then setting the HWTH_OPT_HTTPHEADERS        */
748  /* option of the Request with that list.                     */
749  /*                                                           */
750  /* Returns: 0 if successful, -1 if not                       */
751  /*************************************************************/
752  HTTP_setRequestHeaders:
753   SList = ''
754   acceptJsonHeader = 'Accept:application/json'
755   acceptXMLHeader = 'Accept:application/xml'
756   acceptLanguageHeader = 'Accept-Language: en-US'
757   hostHeader = 'Host: api.geosvc.com'
758   /*********************************************************************/
759   /* Create a brand new SList and specify the first header to be an    */
760   /* "Accept" header that requests that the server return any response  */
761   /* body text in JSON format.                                         */
762   /*********************************************************************/
763   ReturnCode = -1
764   DiagArea. = ''
765   if VERBOSE then
766      say 'Create new SList'
767   address hwthttp "hwthslst ",
768                   "ReturnCode ",
769                   "RequestHandle ",
770                   "HWTH_SLST_NEW ",
771                   "SList ",
772                   "acceptXMLHeader ",
773                   "DiagArea."
```

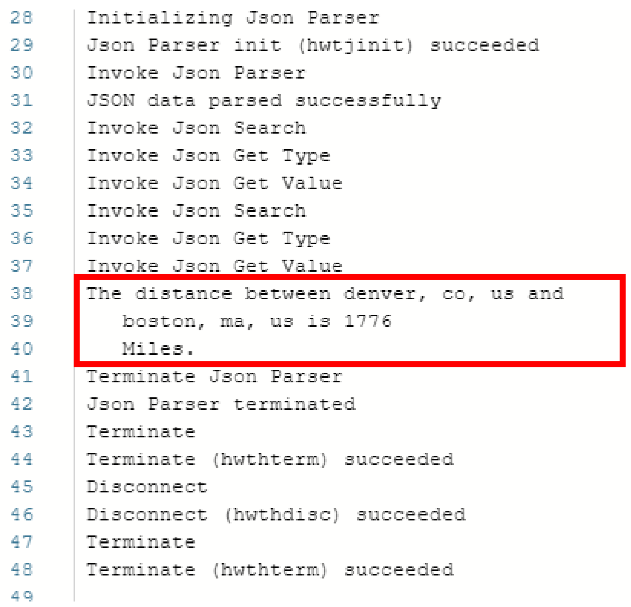
Replace the **acceptXMLHeader** with the **acceptJsonHeade**r on the hwthslst call. Save and re-run RXEXEC1.

Do you get formatted results of current distance between the two locations you specified?

Select the Job Name with the COND CODE of 0000 after clicking the refresh button to get the SYSTSPRT DD.



Job Name	Job ID	Status	Owner	Max - CC	Type
SHARA15	JOB03289	OUTPUT	SWARREN	CC 0000	JOB
SHARA15	JOB03282	OUTPUT	SWARREN	CC 0012	JOB



```
Initializing Json Parser
Json Parser init (hwtjinit) succeeded
Invoke Json Parser
JSON data parsed successfully
Invoke Json Search
Invoke Json Get Type
Invoke Json Get Value
Invoke Json Search
Invoke Json Get Type
Invoke Json Get Value
The distance between denver, co, us and
   boston, ma, us is 1776
   Miles.
Terminate Json Parser
Json Parser terminated
Terminate
Terminate (hwthterm) succeeded
Disconnect
Disconnect (hwthdisc) succeeded
Terminate
Terminate (hwthterm) succeeded
```

Congratulations! You have coded a REST client!

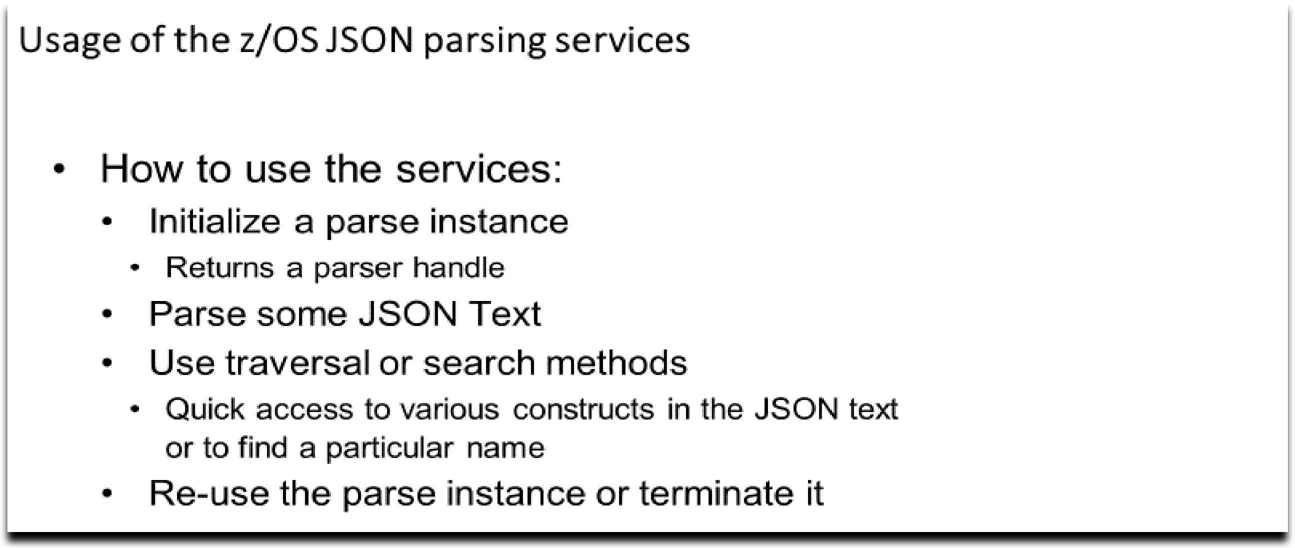
8. Analyze the JSON API calls

Look at the writeData subroutine. Even though we asked for the data to be returned in JSON format earlier, it doesn't necessarily mean that the server will return JSON. The code checks to make sure that the Content Type of the data coming back is JSON by consulting the responseHeader. If the data returned is what we expect, then the JSON parser is invoked. How are these checks made? The variable name set (also called a REXX stem variable) on the HWTHSET for the HWTH_OPT_RESPONSEHDR_USERDATA contains the following upon return:

- stem.0 is the number of elements.
- stem.x (where x is between 1 and stem.0) contains the response header name
- stem.x.1 (where x is between 1 and stem.0) contains the response header value

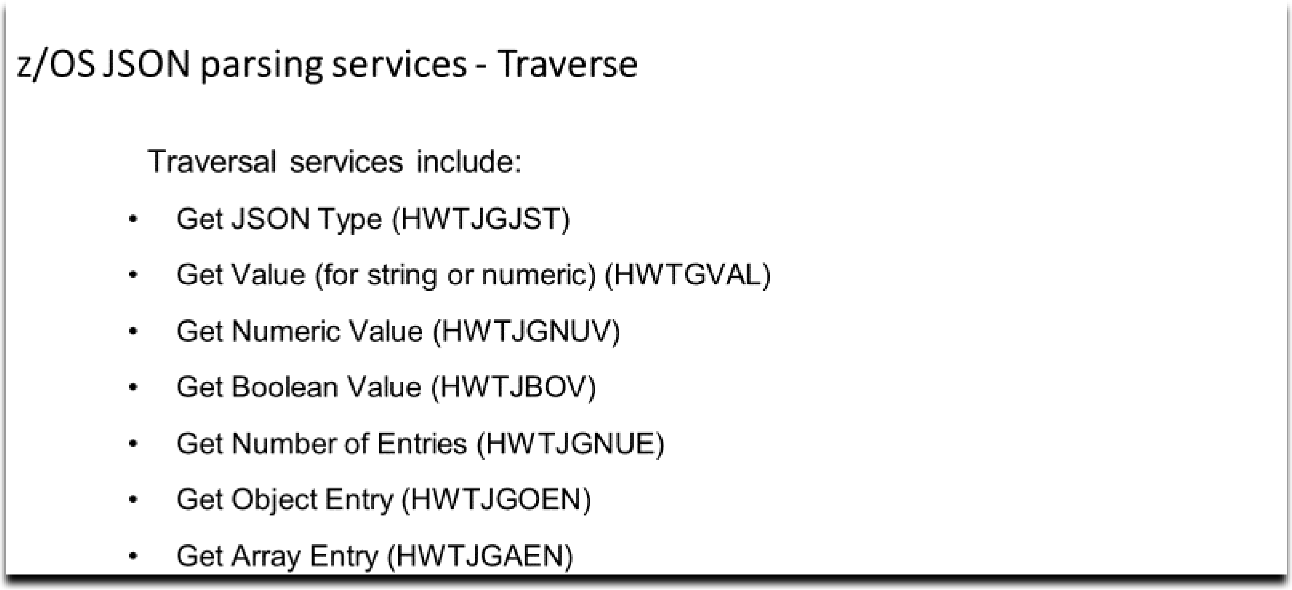
In other words, keep looking thru the response headers. If any of them are Content-Type, then is the value "application/json"? If so, then we should invoke the JSON parser.

How does the JSON parser work?



Usage of the z/OS JSON parsing services

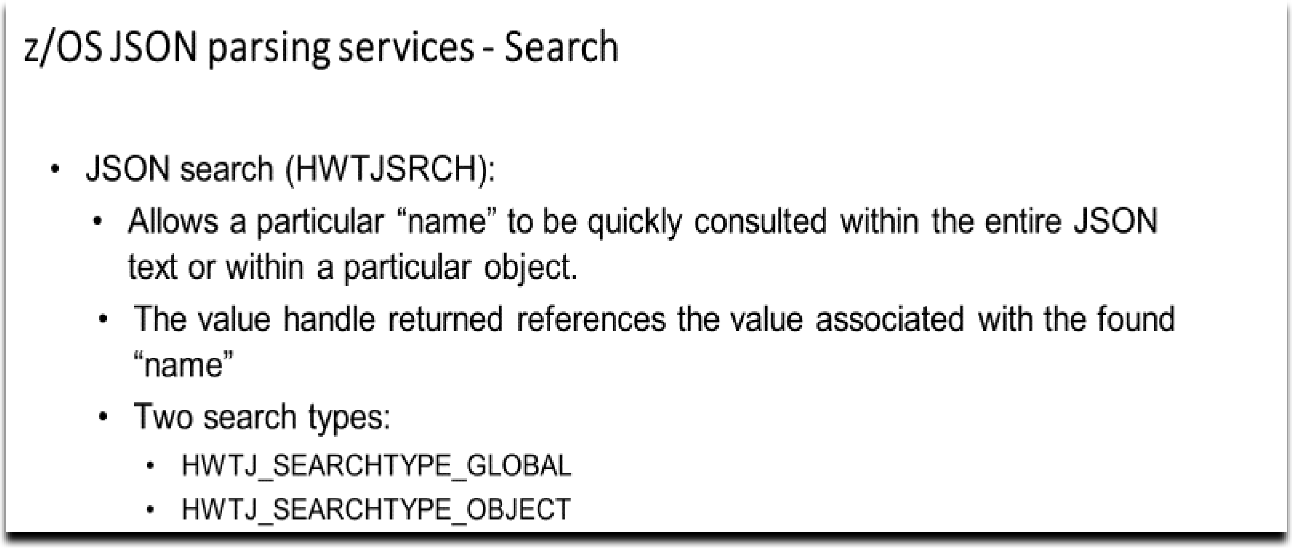
- How to use the services:
  - Initialize a parse instance
    - Returns a parser handle
  - Parse some JSON Text
  - Use traversal or search methods
    - Quick access to various constructs in the JSON text or to find a particular name
  - Re-use the parse instance or terminate it



z/OS JSON parsing services - Traverse

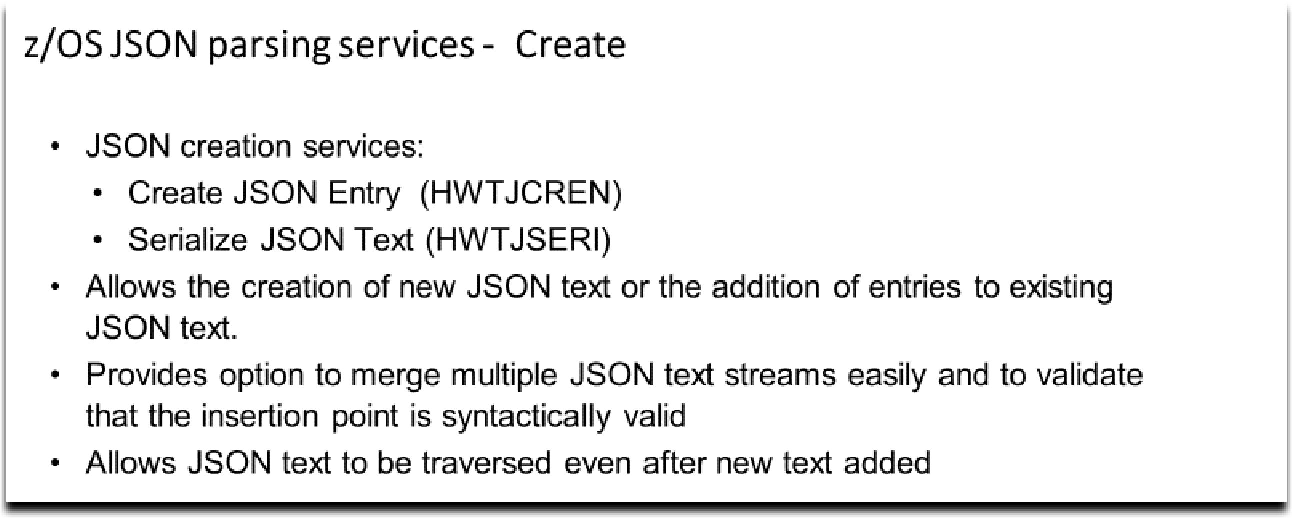
Traversal services include:

- Get JSON Type (HWTJGJST)
- Get Value (for string or numeric) (HWTGVAL)
- Get Numeric Value (HWTJGNUV)
- Get Boolean Value (HWTJBOV)
- Get Number of Entries (HWTJGNUE)
- Get Object Entry (HWTJGOEN)
- Get Array Entry (HWTJGAEN)



z/OS JSON parsing services - Search

- JSON search (HWTJSRCH):
  - Allows a particular “name” to be quickly consulted within the entire JSON text or within a particular object.
  - The value handle returned references the value associated with the found “name”
  - Two search types:
    - HWTJ_SEARCHTYPE_GLOBAL
    - HWTJ_SEARCHTYPE_OBJECT



z/OS JSON parsing services - Create

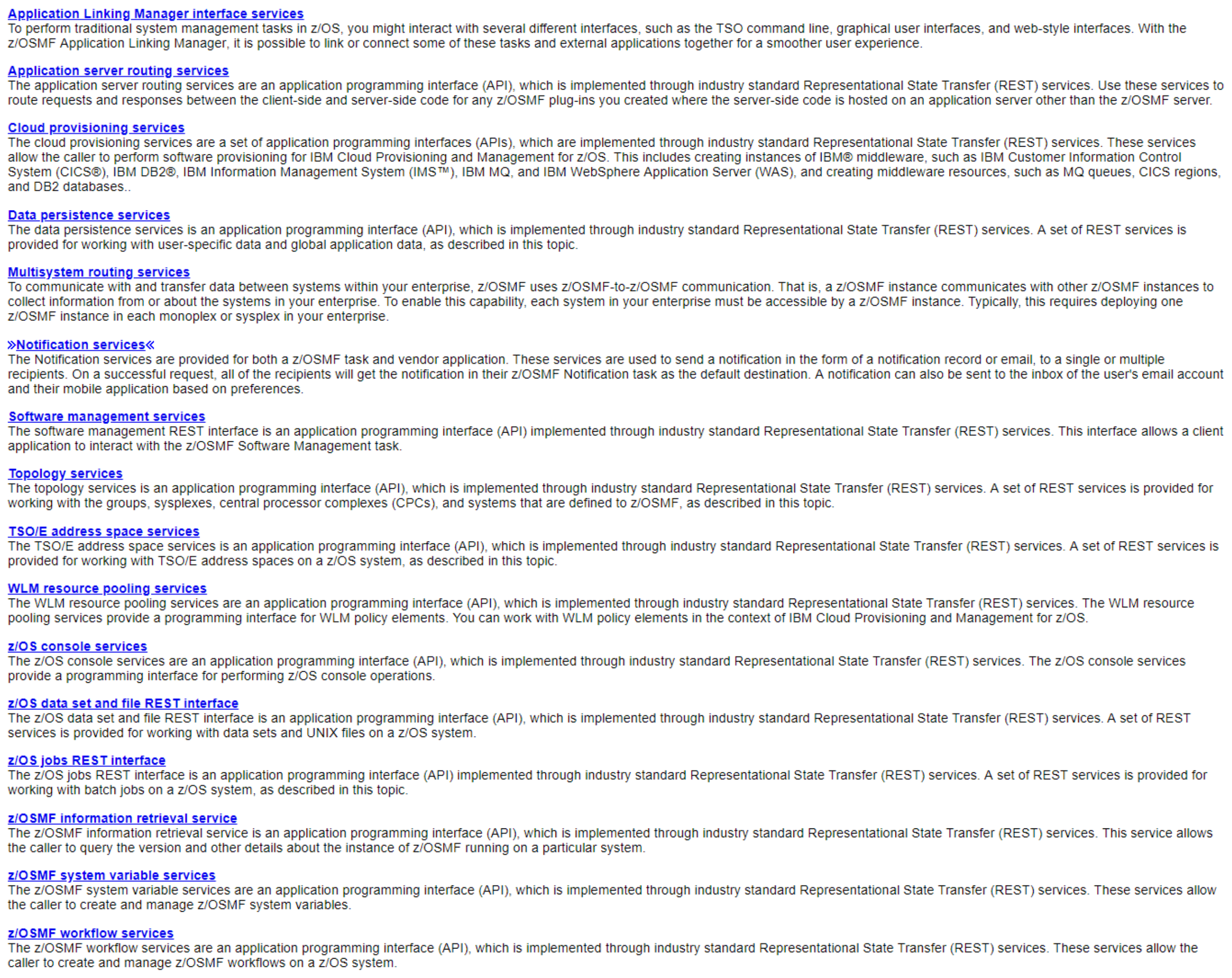
- JSON creation services:
  - Create JSON Entry (HWTJCREN)
  - Serialize JSON Text (HWTJSERI)
- Allows the creation of new JSON text or the addition of entries to existing JSON text.
- Provides option to merge multiple JSON text streams easily and to validate that the insertion point is syntactically valid
- Allows JSON text to be traversed even after new text added

9. Example 2: Leveraging z/OSMF REST APIs

The first example, like many other external REST APIs, may have some value in your shop. But what about a z/OS application executing z/OS functions on another system in your enterprise, regardless of location? z/OSMF (short for z/OS Management Facility) provides a rich set of REST APIs that allow your application to perform many different types of tasks including working with jobs, data sets, provisioning z/OS Middleware, Notification services, TSO/E functions, z/OS Console and much, much more. A list is provided in this book:

IBM z/OS Management Facility Programming Guide.

Alternatively, you can just Google "**Using the z/OSMF REST Services".** Or even OpenAPI(Swagger) for many z/OSMF REST APIs. Click the first link suggested by Google (this will take you to the z/OSMF Programming Guide) and scroll down to the bottom where the z/OSMF REST services are described.



Application Linking Manager interface services
To perform traditional system management tasks in z/OS, you might interact with several different interfaces, such as the TSO command line, graphical user interfaces, and web-style interfaces. With the z/OSMF Application Linking Manager, it is possible to link or connect some of these tasks and external applications together for a smoother user experience.

Application server routing services
The application server routing services are an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. Use these services to route requests and responses between the client-side and server-side code for any z/OSMF plug-ins you created where the server-side code is hosted on an application server other than the z/OSMF server.

Cloud provisioning services
The cloud provisioning services are a set of application programming interfaces (APIs), which are implemented through industry standard Representational State Transfer (REST) services. These services allow the caller to perform software provisioning for IBM Cloud Provisioning and Management for z/OS. This includes creating instances of IBM® middleware, such as IBM Customer Information Control System (CICS®), IBM DB2®, IBM Information Management System (IMS™), IBM MQ, and IBM WebSphere Application Server (WAS), and creating middleware resources, such as MQ queues, CICS regions, and DB2 databases..

Data persistence services
The data persistence services is an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with user-specific data and global application data, as described in this topic.

Multisystem routing services
To communicate with and transfer data between systems within your enterprise, z/OSMF uses z/OSMF-to-z/OSMF communication. That is, a z/OSMF instance communicates with other z/OSMF instances to collect information from or about the systems in your enterprise. To enable this capability, each system in your enterprise must be accessible by a z/OSMF instance. Typically, this requires deploying one z/OSMF instance in each monoplex or sysplex in your enterprise.

»Notification services«
The Notification services are provided for both a z/OSMF task and vendor application. These services are used to send a notification in the form of a notification record or email, to a single or multiple recipients. On a successful request, all of the recipients will get the notification in their z/OSMF Notification task as the default destination. A notification can also be sent to the inbox of the user's email account and their mobile application based on preferences.

Software management services
The software management REST interface is an application programming interface (API) implemented through industry standard Representational State Transfer (REST) services. This interface allows a client application to interact with the z/OSMF Software Management task.

Topology services
The topology services is an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with the groups, sysplexes, central processor complexes (CPCs), and systems that are defined to z/OSMF, as described in this topic.

TSO/E address space services
The TSO/E address space services is an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with TSO/E address spaces on a z/OS system, as described in this topic.

WLM resource pooling services
The WLM resource pooling services are an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. The WLM resource pooling services provide a programming interface for WLM policy elements. You can work with WLM policy elements in the context of IBM Cloud Provisioning and Management for z/OS.

z/OS console services
The z/OS console services are an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. The z/OS console services provide a programming interface for performing z/OS console operations.

z/OS data set and file REST interface
The z/OS data set and file REST interface is an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with data sets and UNIX files on a z/OS system.

z/OS jobs REST interface
The z/OS jobs REST interface is an application programming interface (API) implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with batch jobs on a z/OS system, as described in this topic.

z/OSMF information retrieval service
The z/OSMF information retrieval service is an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. This service allows the caller to query the version and other details about the instance of z/OSMF running on a particular system.

z/OSMF system variable services
The z/OSMF system variable services are an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. These services allow the caller to create and manage z/OSMF system variables.

z/OSMF workflow services
The z/OSMF workflow services are an application programming interface (API), which is implemented through industry standard Representational State Transfer (REST) services. These services allow the caller to create and manage z/OSMF workflows on a z/OS system.

Our second exercise will attempt to illustrate a more real-life scenario leveraging some of the z/OSMF REST APIs listed above. Imagine that your shop would like a certain job on a remote system to be run on a regular basis. If the job output shows a problem, then the appropriate person would be notified via email of the problem. This is all very easy to do using the z/OSMF REST API suite!

Our program will perform the following steps:

1. Submit a simple z/OS healthcheck job to print the results of a healthcheck (using the z/OSMF Submit job REST API)

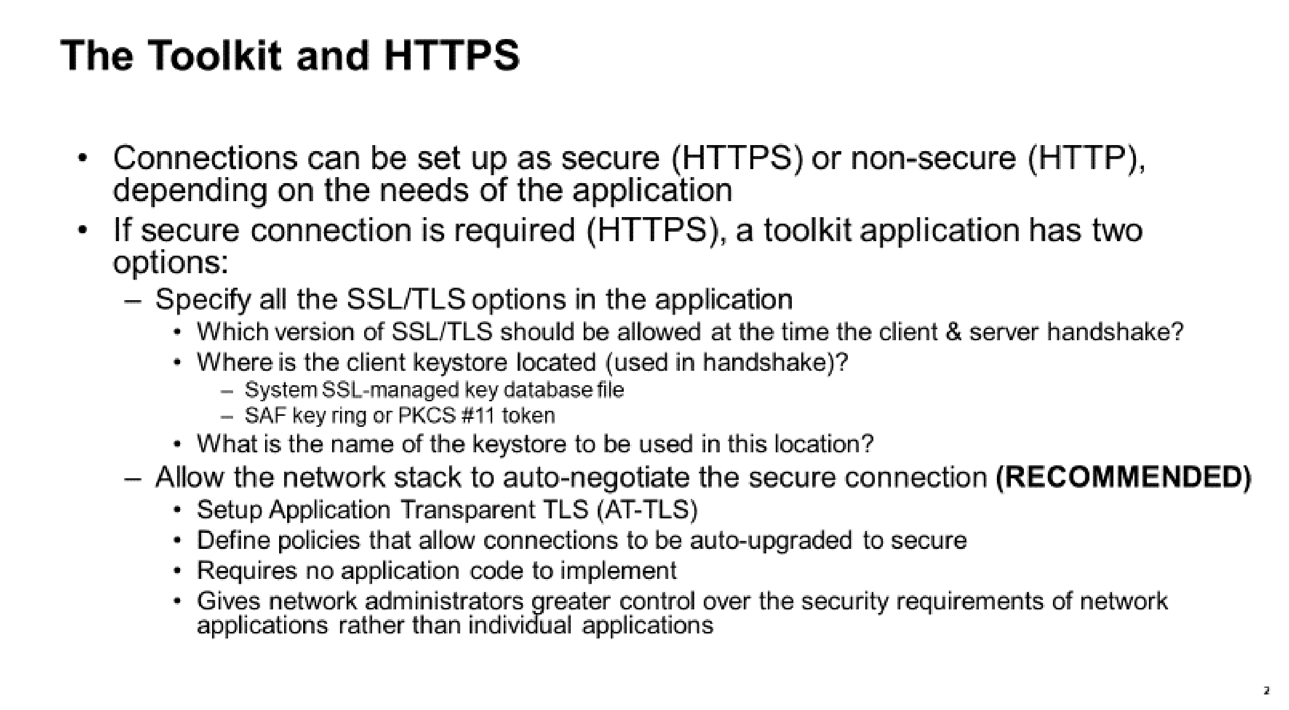
2. Check the status of the job to make sure it has completed successfully. (using the z/OSMF job status retrieval REST API)

3. If the job completed successfully, it will retrieve the output written to a data set (using the z/OSMF data set retrieval REST API)

4. It will then analyze the output. If the IBM z/OS Health Check failed in any way, then it will send the output to an email address (using the z/OSMF notification REST API)

10. The toolkit and secure connections (HTTPS and AT-TLS)

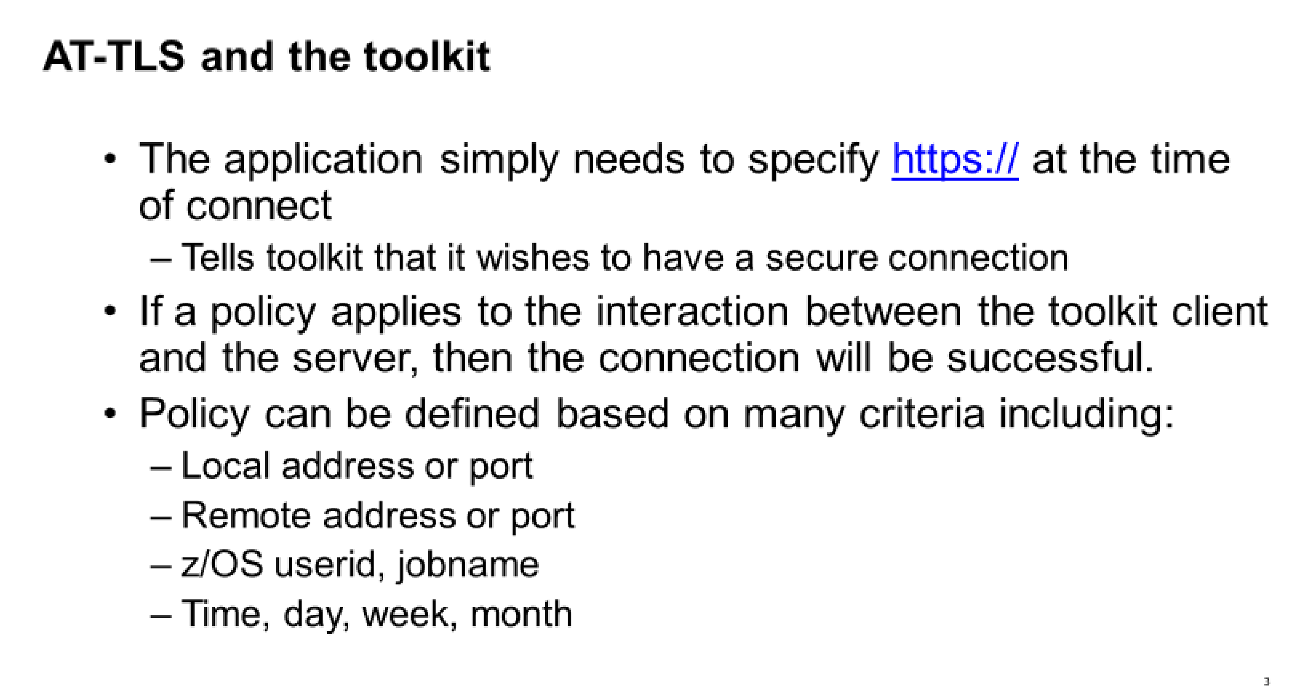
Unlike our first example, which only required an HTTP connection (that doesn't require the data to be encrypted from the client to the server and back again), the z/OSMF REST APIs require an HTTPS connection. While this could make the application slightly more difficult to code, we will show you that there is a way to have an HTTPS connection with no additional code! Study the following slides to learn more.



The Toolkit and HTTPS

- Connections can be set up as secure (HTTPS) or non-secure (HTTP), depending on the needs of the application
- If secure connection is required (HTTPS), a toolkit application has two options:
  - Specify all the SSL/TLS options in the application
    - Which version of SSL/TLS should be allowed at the time the client & server handshake?
    - Where is the client keystore located (used in handshake)?
      - System SSL-managed key database file
      - SAF key ring or PKCS #11 token
    - What is the name of the keystore to be used in this location?
  - Allow the network stack to auto-negotiate the secure connection **(RECOMMENDED)**
    - Setup Application Transparent TLS (AT-TLS)
    - Define policies that allow connections to be auto-upgraded to secure
    - Requires no application code to implement
    - Gives network administrators greater control over the security requirements of network applications rather than individual applications

2



AT-TLS and the toolkit

- The application simply needs to specify https:// at the time of connect
  - Tells toolkit that it wishes to have a secure connection
- If a policy applies to the interaction between the toolkit client and the server, then the connection will be successful.
- Policy can be defined based on many criteria including:
  - Local address or port
  - Remote address or port
  - z/OS userid, jobname
  - Time, day, week, month

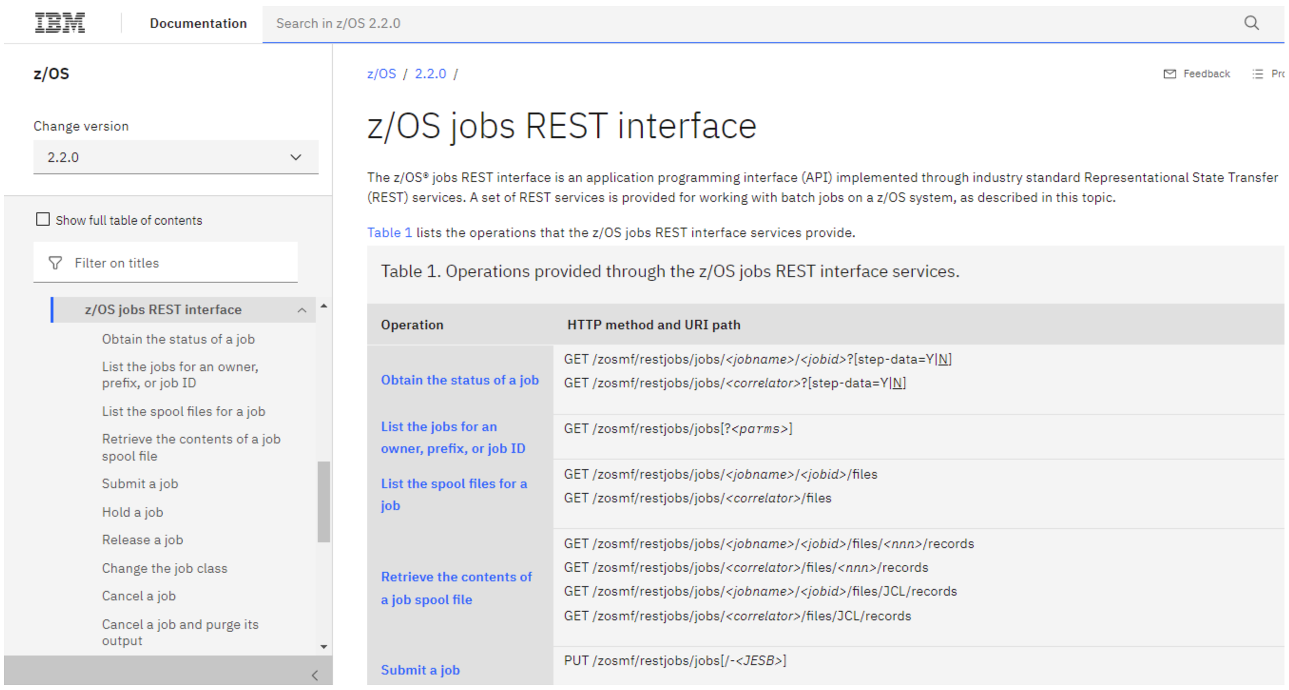
3

!! In our AT-TLS configuration on the system we are using, we have a very simple policy defined. Any job prefixed with the letters TKT will result in the TCP stack to call System SSL to auto-negotiate a TLS 1.2 connection !!

11. Specify options to submit the job using z/OSMF submit job REST API request

As in the first example, we need the gather some information in order to specify the correct options to the toolkit for the connection and request. The connection has already been taken care of by the sample program and the connection should automatically be upgraded to a secure HTTPS connection by AT-TLS when we run the program. We need a bunch of information for the submit job request.

Let's go to the z/OSMF jobs API documentation online. You can get by clicking the **"z/OS jobs REST interface"** from the previous z/OSMF documentation page. Here's what you should see:



Documentation | Search in z/OS 2.2.0

z/OS

Change version

2.2.0

Show full table of contents

Filter on titles

- z/OS jobs REST interface
  - Obtain the status of a job
  - List the jobs for an owner, prefix, or job ID
  - List the spool files for a job
  - Retrieve the contents of a job spool file
  - Submit a job
  - Hold a job
  - Release a job
  - Change the job class
  - Cancel a job
  - Cancel a job and purge its output

z/OS / 2.2.0 /

Feedback

z/OS jobs REST interface

The z/OS® jobs REST interface is an application programming interface (API) implemented through industry standard Representational State Transfer (REST) services. A set of REST services is provided for working with batch jobs on a z/OS system, as described in this topic.

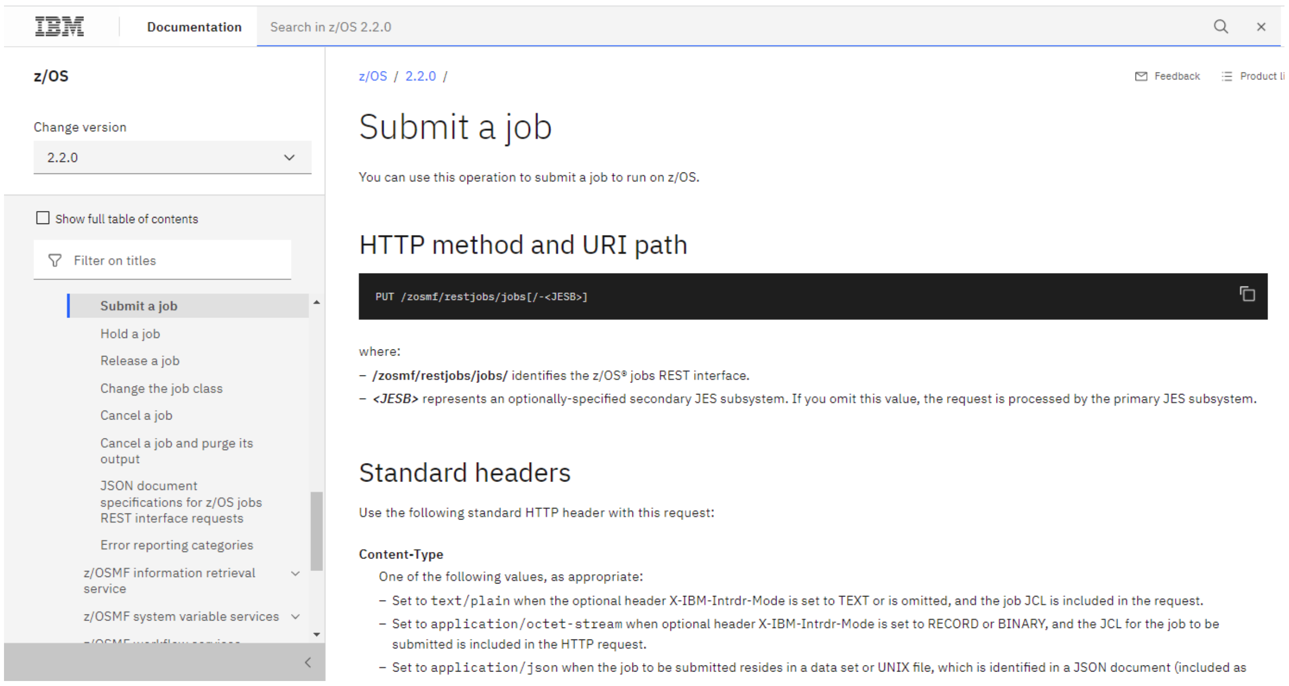
Table 1 lists the operations that the z/OS jobs REST interface services provide.

Table 1. Operations provided through the z/OS jobs REST interface services.

Operation	HTTP method and URI path
Obtain the status of a job	GET /zosmf/restjobs/jobs/*<jobname>*/*<jobid>*?[step-data=Y|N] GET /zosmf/restjobs/jobs/*<correlator>*?[step-data=Y|N]
List the jobs for an owner, prefix, or job ID	GET /zosmf/restjobs/jobs[?*<parms>*]
List the spool files for a job	GET /zosmf/restjobs/jobs/*<jobname>*/*<jobid>*/files GET /zosmf/restjobs/jobs/*<correlator>*/files
Retrieve the contents of a job spool file	GET /zosmf/restjobs/jobs/*<jobname>*/*<jobid>*/files/*<nnn>*/records GET /zosmf/restjobs/jobs/*<correlator>*/files/*<nnn>*/records GET /zosmf/restjobs/jobs/*<jobname>*/*<jobid>*/files/JCL/records GET /zosmf/restjobs/jobs/*<correlator>*/files/JCL/records
Submit a job	PUT /zosmf/restjobs/jobs[/-*<JESB>*]

Note all of the types of actions you can take against a job. Pretty neat stuff. Click on the **Submit a job** link to get more detailed information on submitting a job using this REST API.

You should see this:


IBM
Documentation
Search in z/OS 2.2.0
z/OS
Change version
2.2.0
Show full table of contents
Filter on titles
Submit a job
Hold a job
Release a job
Change the job class
Cancel a job
Cancel a job and purge its output
JSON document specifications for z/OS jobs REST interface requests
Error reporting categories
z/OSMF information retrieval service
z/OSMF system variable services
z/OS / 2.2.0 /
Feedback
Product li

Submit a job

You can use this operation to submit a job to run on z/OS.

HTTP method and URI path

```
PUT /zosmf/restjobs/jobs[/-<JESB>]
```

where:

- **/zosmf/restjobs/jobs/** identifies the z/OS® jobs REST interface.
- ***<JESB>*** represents an optionally-specified secondary JES subsystem. If you omit this value, the request is processed by the primary JES subsystem.

Standard headers

Use the following standard HTTP header with this request:

Content-Type

One of the following values, as appropriate:

- Set to `text/plain` when the optional header X-IBM-Intrdr-Mode is set to TEXT or is omitted, and the job JCL is included in the request.
- Set to `application/octet-stream` when optional header X-IBM-Intrdr-Mode is set to RECORD or BINARY, and the JCL for the job to be submitted is included in the HTTP request.
- Set to `application/json` when the job to be submitted resides in a data set or UNIX file, which is identified in a JSON document (included as

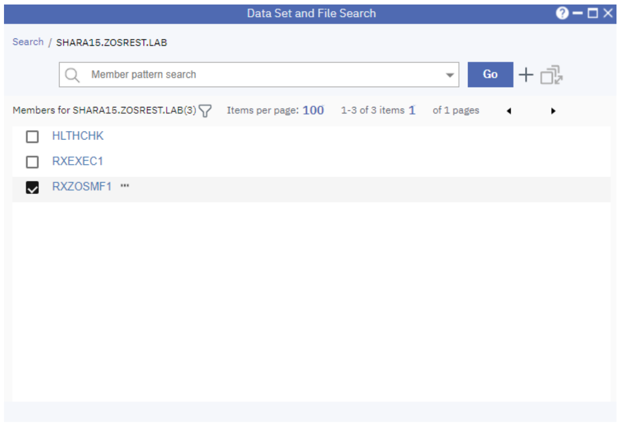
From this page, you should be able to get a bunch of information that you will need to update in the sample program including:

Request URI : ________________________________

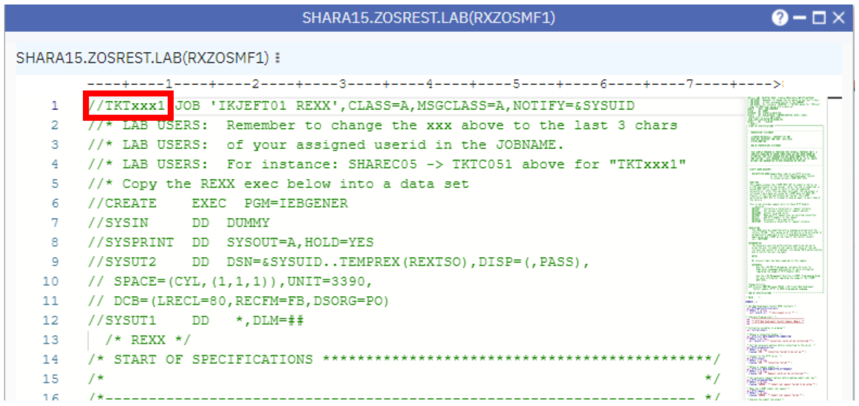
Request method (e.g. GET, PUT, POST, DELETE): __________________________

Your job will be a member in a Partitioned Data Set (PDS). Scroll down and find out what form this dataset needs to be specified in the program on the web page.

Now that we have the basics needed to submit the job, let's go into our program. From your *labuserid*.ZOSREST.LAB dataset, edit the RXZOSMF1 member:



NOTE: Don't forget to click the Pen icon after entering browse so that you can edit this program.



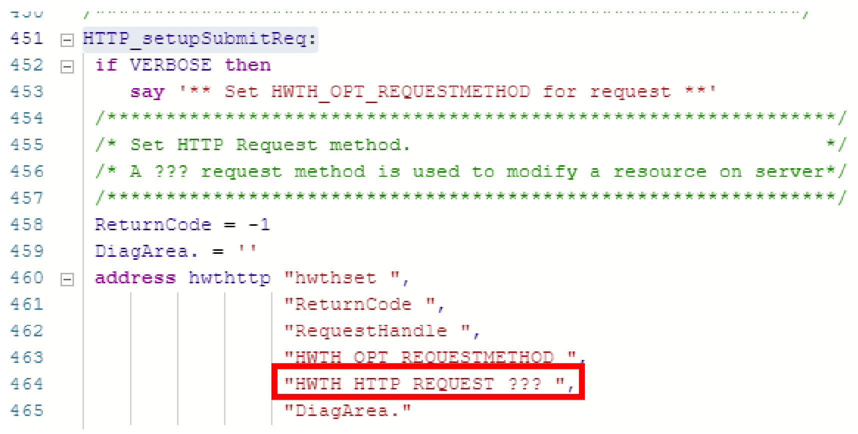
```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---->
//TKTxxx1 JOB 'IKJEFT01 REXX',CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//* LAB USERS:  Remember to change the xxx above to the last 3 chars
//* LAB USERS:  of your assigned userid in the JOBNAME.
//* LAB USERS:  For instance: SHAREC05 -> TKTC051 above for "TKTxxx1"
//* Copy the REXX exec below into a data set
//CREATE    EXEC  PGM=IEBGENER
//SYSIN     DD  DUMMY
//SYSPRINT  DD  SYSOUT=A,HOLD=YES
//SYSUT2    DD  DSN=&SYSUID..TEMPREX(REXTSO),DISP=(,PASS),
// SPACE=(CYL,(1,1,1)),UNIT=3390,
// DCB=(LRECL=80,RECFM=FB,DSORG=PO)
//SYSUT1    DD   *,DLM=##
  /* REXX */
/* START OF SPECIFICATIONS ********************************************/
/*                                                                    */
/*------------------------------------------------------------------- */
```

Notice the jobname highlighted above. It has the name TKTxxx1. As we learned from the AT-TLS policy definitions on this test system, any jobname prefixed with TKT will automatically be attempted to upgrade to HTTPS.

Change the xxx to the last 3 characters of your assigned userid. For example, for userid SHARA15, the jobname should be TKTA151.

The layout of this program is similar to the first example we worked on. A connection is initialized, several connection options are set and then the connect service is called to establish the connection. After this step and with the correct AT-TLS policy in effect, we should have a secure connection.

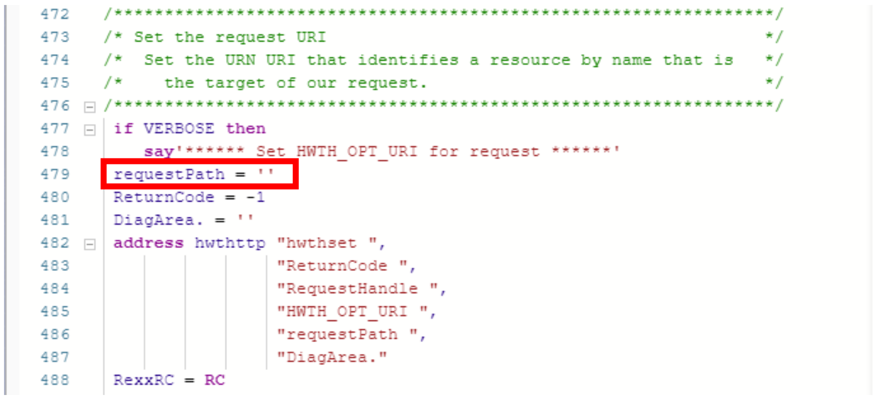
The submit request is then setup. We need to fill in some pieces with the information gathered from the z/OSMF documentation. Let's find **HTTP_setupSubmitReq:** from the find bar in the z/OSMF editor (Reminder: **Ctrl-F**, then type what you are searching for (around line 450). This will find the beginning of the setup routine as shown below.



```
HTTP_setupSubmitReq:
 if VERBOSE then
    say '** Set HWTH_OPT_REQUESTMETHOD for request **'
 /*************************************************************/
 /* Set HTTP Request method.                                  */
 /* A ??? request method is used to modify a resource on server*/
 /*************************************************************/
 ReturnCode = -1
 DiagArea. = ''
 address hwthttp "hwthset ",
                 "ReturnCode ",
                 "RequestHandle ",
                 "HWTH_OPT_REQUESTMETHOD ",
                 "HWTH_HTTP_REQUEST_??? ",
                 "DiagArea."
```

Fill in the request method required for the submit job REST API where the ??? is located.

Then just scroll down a few lines (around line 478) to find where the URI is set (HWTH_OPT_URI).

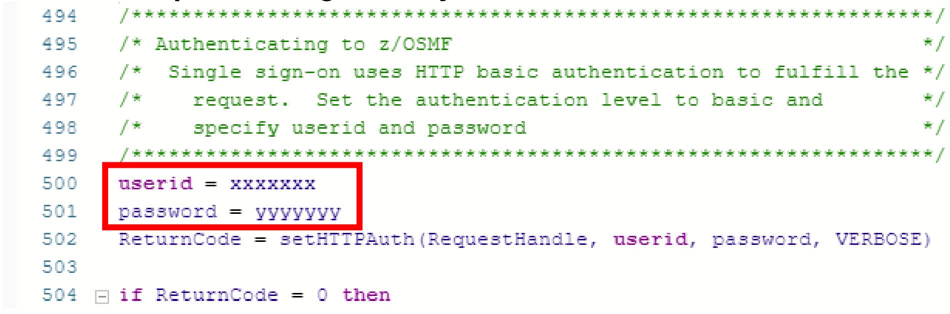


```
472  /*****************************************************************/
473  /* Set the request URI                                           */
474  /*  Set the URN URI that identifies a resource by name that is   */
475  /*    the target of our request.                                 */
476  /*****************************************************************/
477   if VERBOSE then
478      say'****** Set HWTH_OPT_URI for request ******'
479   requestPath = ''
480   ReturnCode = -1
481   DiagArea. = ''
482   address hwthttp "hwthset ",
483                   "ReturnCode ",
484                   "RequestHandle ",
485                   "HWTH_OPT_URI ",
486                   "requestPath ",
487                   "DiagArea."
488   RexxRC = RC
```

Notice that the requestPath is not set (see line circled above). Fill in the requestPath the value you found in the Programming Guide.

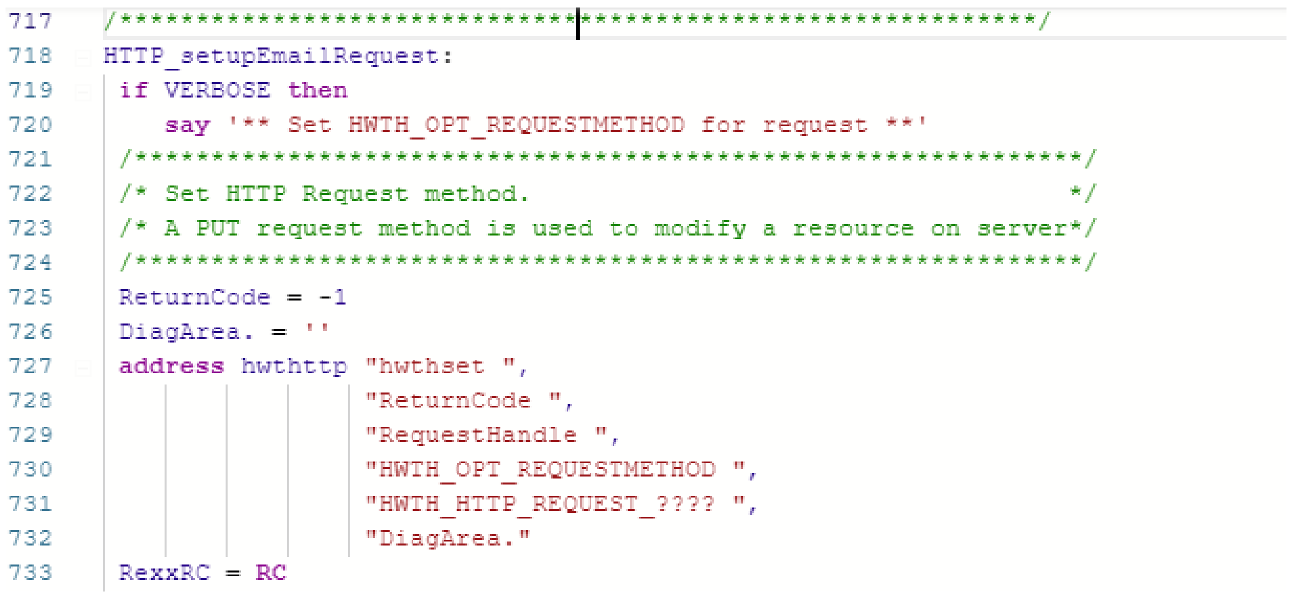
Thanks to AT-TLS, we have an encrypted connection between our application and z/OSMF on the target system. However, z/OSMF on the target system has no idea who we are. We need to send our login credentials to z/OSMF to allow z/OSMF to authenticate each request we make. This encrypted connection allows us to send the userid and password of the target system (in our simple exercise, it is the same userid and password that you logged onto) on our request securely. This is a one-time operation. Once we have sent these credentials, z/OSMF sends back a security token that the toolkit cookie manager takes care of for us. On all subsequent requests, the token will be automatically added to the request, and z/OSMF will know who we are.

Scroll down a little bit more (around line 500) and fill in the userid and password fields with your lab userid and password given to you:



```
494  /*****************************************************************/
495  /* Authenticating to z/OSMF                                      */
496  /*  Single sign-on uses HTTP basic authentication to fulfill the */
497  /*    request.  Set the authentication level to basic and        */
498  /*    specify userid and password                                */
499  /*****************************************************************/
500  userid = xxxxxxx
501  password = yyyyyyy
502  ReturnCode = setHTTPAuth(RequestHandle, userid, password, VERBOSE)
503
504  if ReturnCode = 0 then
```

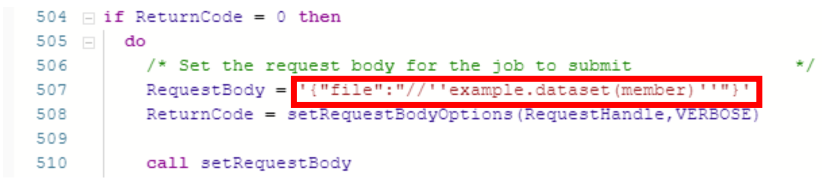
We need to fill in one more value for the submit job request: the dataset containing the job. From the Programming Guide, we learned how we specify a PDS member to z/OSMF (in the request body).



```
/****************************************************************/
HTTP_setupEmailRequest:
 if VERBOSE then
    say '** Set HWTH_OPT_REQUESTMETHOD for request **'
 /***************************************************************/
 /* Set HTTP Request method.                                   */
 /* A PUT request method is used to modify a resource on server*/
 /***************************************************************/
 ReturnCode = -1
 DiagArea. = ''
 address hwthttp "hwthset ",
                 "ReturnCode ",
                 "RequestHandle ",
                 "HWTH_OPT_REQUESTMETHOD ",
                 "HWTH_HTTP_REQUEST_???? ",
                 "DiagArea."
 RexxRC = RC
```

Scroll down just a few lines (around line 506) and fill in the dataset and member name of the healthcheck job member (**HLTHCHK**) we are going to submit that you saved above. **The location of your healthcheck job JCL is: SHARxnn.ZOSREST.LAB(HLTHCHK) (fill in your assigned userid) In other words, replace example.dataset(member) with the location of your job.**

Note: Don't remove any of the quotes listed here or you will likely get a REXX syntax error.

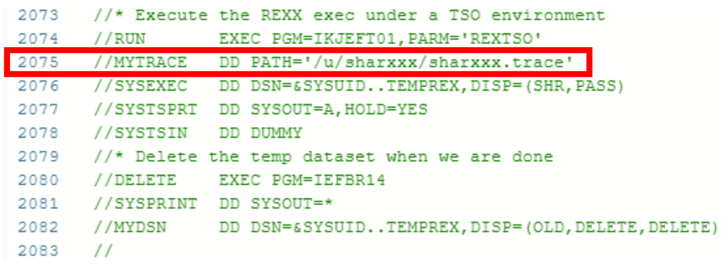


```
504  if ReturnCode = 0 then
505    do
506      /* Set the request body for the job to submit                */
507      RequestBody = '{"file":"//''example.dataset(member)''"}'
508      ReturnCode = setRequestBodyOptions(RequestHandle,VERBOSE)
509
510      call setRequestBody
```

Now our request is complete.

12. Specify location for toolkit trace output

As you know very well, the first time we run something, there could be minor issues with our program. We need to have toolkit tracing turned on. This time, we are going to direct the trace output to a zFS file. The toolkit VERBOSE OUTPUT attribute requires that a DD representing the zFS file is specified. To find out the name of that DD, scroll all the way to the bottom of the member.

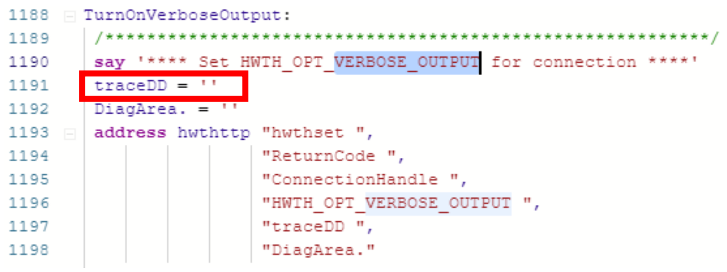


```
//* Execute the REXX exec under a TSO environment
//RUN       EXEC PGM=IKJEFT01,PARM='REXTSO'
//MYTRACE   DD PATH='/u/sharxxx/sharxxx.trace'
//SYSEXEC   DD DSN=&SYSUID..TEMPREX,DISP=(SHR,PASS)
//SYSTSPRT  DD SYSOUT=A,HOLD=YES
//SYSTSIN   DD DUMMY
//* Delete the temp dataset when we are done
//DELETE    EXEC PGM=IEFBR14
//SYSPRINT  DD SYSOUT=*
//MYDSN     DD DSN=&SYSUID..TEMPREX,DISP=(OLD,DELETE,DELETE)
//
```

As you can see the DD name is called **MYTRACE**. Replace the xxx in both locations to the last 3 characters of your assigned userid (in lowercase). E.g. If you are SHAREA15, then the Path value will be '/u/shara15/shara15.trace'. This path name is case sensitive.

Trace file name: ______________________________ (you will need to know this later)

Now go back to the top of the program and find VERBOSE_OUTPUT a couple of times until you find where the VERBOSE_OUTPUT is being set.



```
TurnOnVerboseOutput:
 /*********************************************************/
 say '**** Set HWTH_OPT_VERBOSE_OUTPUT for connection ****'
 traceDD = ''
 DiagArea. = ''
 address hwthttp "hwthset ",
                 "ReturnCode ",
                 "ConnectionHandle ",
                 "HWTH_OPT_VERBOSE_OUTPUT ",
                 "traceDD ",
                 "DiagArea."
```

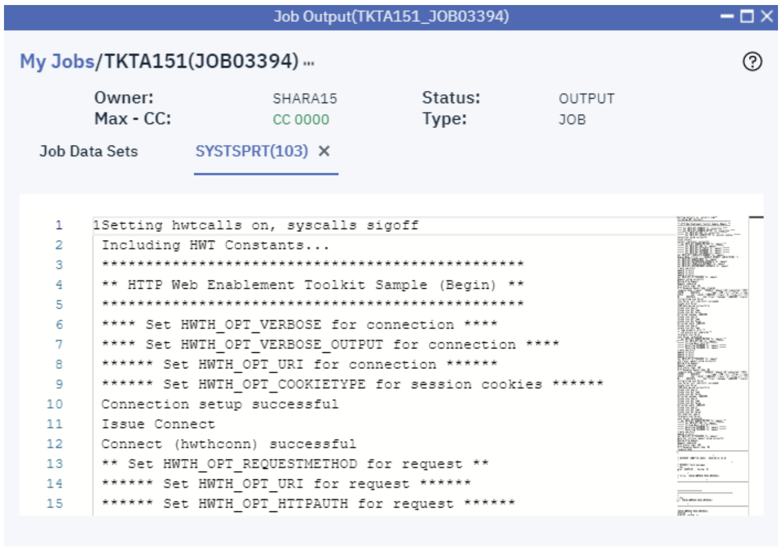
Set the traceDD variable to the name of the DD you just found out.

13. Read the rest of the program

Other subsequent steps have already been written for you. Take a moment and browse the code in checkJobStatus, HTTP_setupDataSetRetrieveReq and parseZOSMFDataSetOutput routines.

14. Run the program

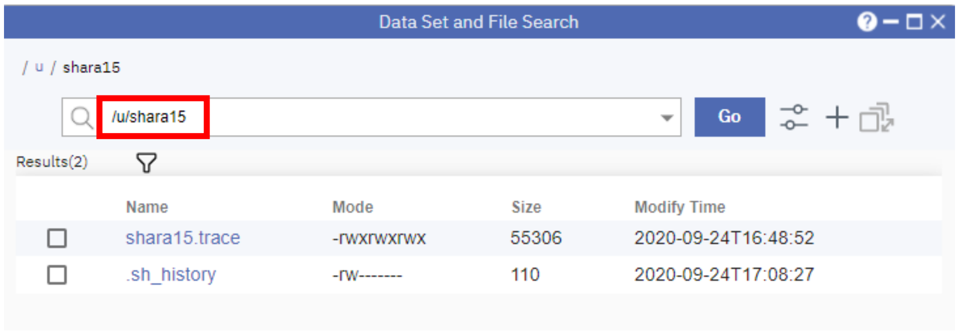
Ok..now we are ready to run this program. **First, save the file.** Since again this is JCL, we will submit this using z/OSMF. Do you remember how to submit the job? (If you forgot, look thru the exercise. Now let's take a look at the job output (remember to go into the Job Output task in z/OSMF). You can sort on jobs with your jobname prefix or owned by your SHARxnn userid. For example, if your jobname is TKTA151, you can type **owner SHARA15. Don't forget to also hit the refresh button if you don't see the SYSTSPRT DD name in the output.**



Owner: SHARA15 Status: OUTPUT
Max - CC: CC 0000 Type: JOB

```
1Setting hwtcalls on, syscalls sigoff
 Including HWT Constants...
 ************************************************
 ** HTTP Web Enablement Toolkit Sample (Begin) **
 ************************************************
 **** Set HWTH_OPT_VERBOSE for connection ****
 **** Set HWTH_OPT_VERBOSE_OUTPUT for connection ****
 ****** Set HWTH_OPT_URI for connection ******
 ****** Set HWTH_OPT_COOKIETYPE for session cookies ******
 Connection setup successful
 Issue Connect
 Connect (hwthconn) successful
 ** Set HWTH_OPT_REQUESTMETHOD for request **
 ****** Set HWTH_OPT_URI for request ******
 ****** Set HWTH_OPT_HTTPAUTH for request ******
```

To look at the trace output in the zFS to help debug toolkit issues, there are many ways. To use z/OSMF, go into Data Set and File Search (magnifying glass in lower right-hand corner of z/OSMF desktop). Type the **name of the directory** that you specified in the JCL at the bottom of your REXX program. The below shows the example for the SHARA15 userid.



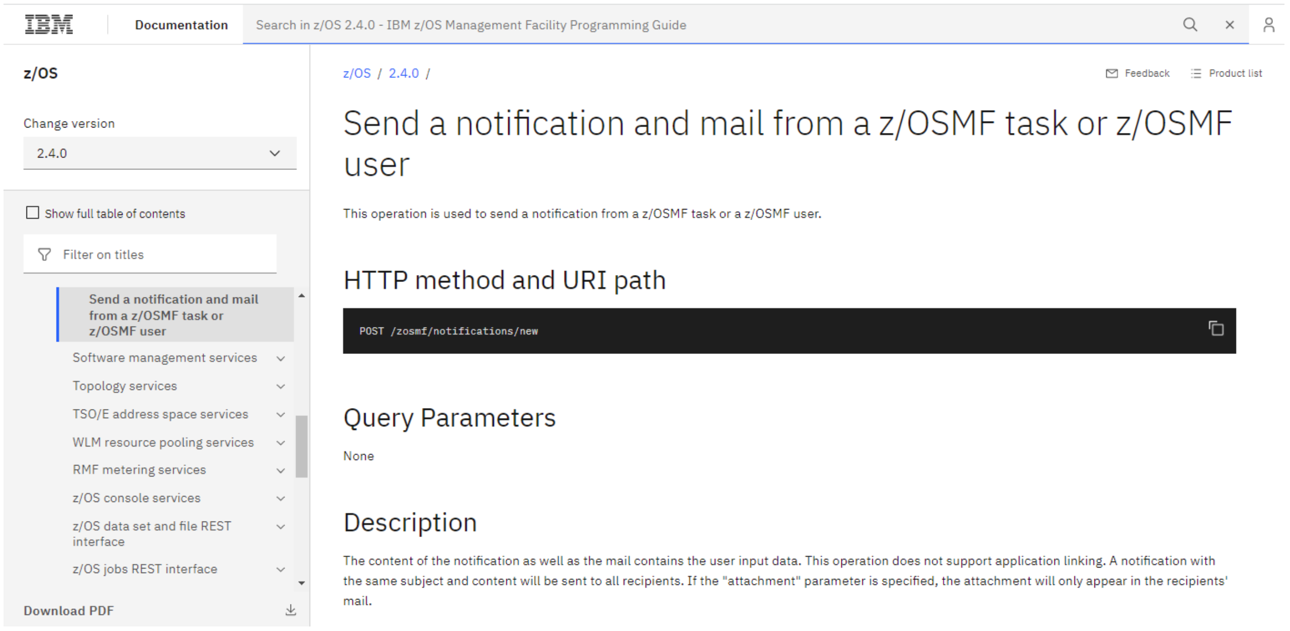
	Name	Mode	Size	Modify Time
☐	shara15.trace	-rwxrwxrwx	55306	2020-09-24T16:48:52
☐	.sh_history	-rw-------	110	2020-09-24T17:08:27

Now just click on the trace file name to browse the zFS file. You will likely need to scroll over to the right to see the trace contents and not just the timestamp prefix for the trace messages.

15. Run email portion of program

Once you get the program running, besides submitting the job (which will have a jobname called **LABUSERH**), the program should have verified that the job ended successful and then retrieved the job output from a dataset.

We need to now send an email if the data indicates a healthcheck error. First, go back to z/OSMF REST API documentation for the notification REST API. You can find it easily by googling **"Send a notification and mail from a z/OSMF task"**


IBM
Documentation
Search in z/OS 2.4.0 - IBM z/OS Management Facility Programming Guide
z/OS
Change version
2.4.0
Show full table of contents
Filter on titles
Send a notification and mail from a z/OSMF task or z/OSMF user
Software management services
Topology services
TSO/E address space services
WLM resource pooling services
RMF metering services
z/OS console services
z/OS data set and file REST interface
z/OS jobs REST interface
Download PDF
z/OS / 2.4.0 /
Feedback
Product list

Send a notification and mail from a z/OSMF task or z/OSMF user

This operation is used to send a notification from a z/OSMF task or a z/OSMF user.

HTTP method and URI path

```
POST /zosmf/notifications/new
```

Query Parameters

None

Description

The content of the notification as well as the mail contains the user input data. This operation does not support application linking. A notification with the same subject and content will be sent to all recipients. If the "attachment" parameter is specified, the attachment will only appear in the recipients' mail.

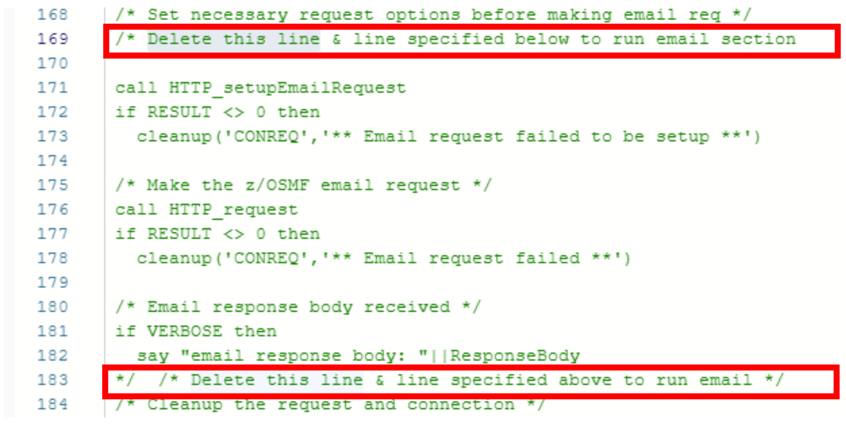
Note the following information:

Request URI: ________________________________

Request Method: ________________________________

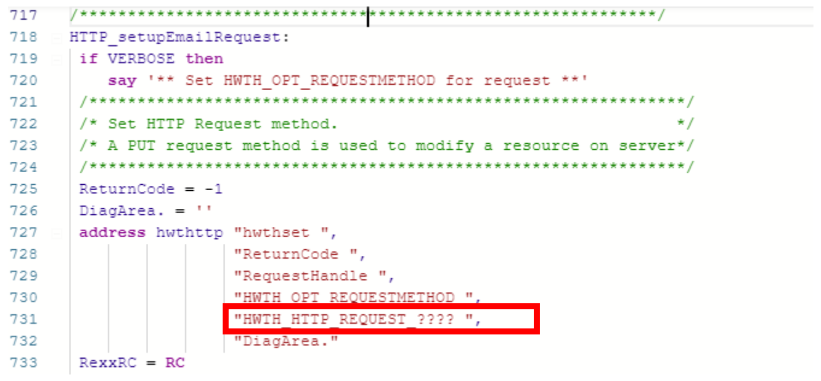
Take a moment and read thru this page. Now go back into your application.

Uncomment the previously commented out email program (around lines 169 and 183). Delete the two lines that say: “delete this line to “uncomment” the email section of the program.

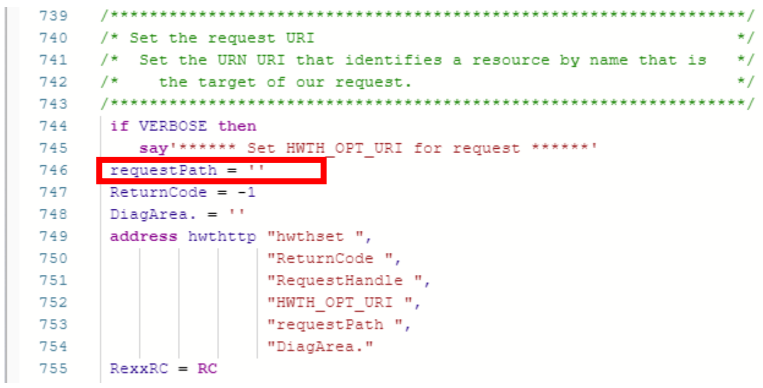


```
168  /* Set necessary request options before making email req */
169  /* Delete this line & line specified below to run email section
170
171  call HTTP_setupEmailRequest
172  if RESULT <> 0 then
173    cleanup('CONREQ','** Email request failed to be setup **')
174
175  /* Make the z/OSMF email request */
176  call HTTP_request
177  if RESULT <> 0 then
178    cleanup('CONREQ','** Email request failed **')
179
180  /* Email response body received */
181  if VERBOSE then
182    say "email response body: "||ResponseBody
183  */  /* Delete this line & line specified above to run email */
184  /* Cleanup the request and connection */
```

Find **HTTP_setupEmailRequest** and make changes to the request options, including request method and URI.

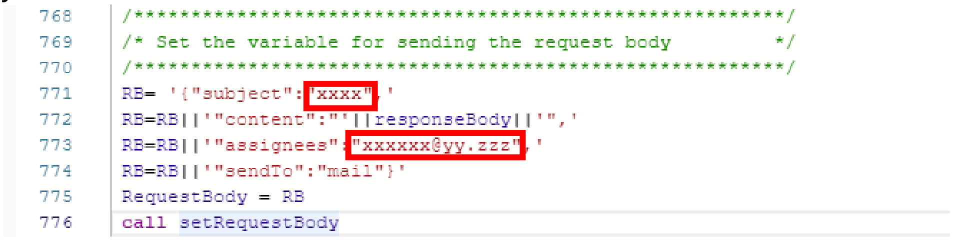


```
717 /***************************************************************/
718 HTTP_setupEmailRequest:
719  if VERBOSE then
720     say '** Set HWTH_OPT_REQUESTMETHOD for request **'
721  /**************************************************************/
722  /* Set HTTP Request method.                                   */
723  /* A PUT request method is used to modify a resource on server*/
724  /**************************************************************/
725  ReturnCode = -1
726  DiagArea. = ''
727  address hwthttp "hwthset ",
728                  "ReturnCode ",
729                  "RequestHandle ",
730                  "HWTH_OPT_REQUESTMETHOD ",
731                  "HWTH_HTTP_REQUEST_???? ",
732                  "DiagArea."
733  RexxRC = RC
```



```
739 /*******************************************************************/
740 /* Set the request URI                                             */
741 /*  Set the URN URI that identifies a resource by name that is    */
742 /*    the target of our request.                                   */
743 /*******************************************************************/
744  if VERBOSE then
745     say'****** Set HWTH_OPT_URI for request ******'
746  requestPath = ''
747  ReturnCode = -1
748  DiagArea. = ''
749  address hwthttp "hwthset ",
750                  "ReturnCode ",
751                  "RequestHandle ",
752                  "HWTH_OPT_URI ",
753                  "requestPath ",
754                  "DiagArea."
755  RexxRC = RC
```

Finally, after updating the request path and request method, set the email options. Scroll down just a little bit and replace the subject and assignees field with an email subject and email address respectively.



```
/*********************************************************/
/* Set the variable for sending the request body         */
/*********************************************************/
RB= '{"subject":"xxxx",'
RB=RB||'"content":"'||responseBody||'",'
RB=RB||'"assignees":"xxxxxx@yy.zzz",'
RB=RB||'"sendTo":"mail"}'
RequestBody = RB
call setRequestBody
```

Save your work and try to run the program again by submitting it. See if you get the email delivered to you!

CONGRATULATIONS! You completed your task!

Extra Credit Sample (Exercise 1)

What other changes can you easily make?

What about outputting the response headers coming back?

What about tweaking this sample to do one of the other request types on the API Reference? (i.e. Nearby, Contains)

Answer key:

GeoServices exercise:

Connect portion of URI: http://api.geosvc.com (notice there is no last '/')

Request portion of URI (in full from website):

/rest/{COUNTRYCODE}/{REGION}/{CITY}/distance?apikey={APIKEY}&p={CITY2}|{POSTALCODE2}&r={REGION2}&c={COUNTRYCODE2}

Request portion of URI (in for using in this lab):
/rest/{COUNTRYCODE}/{REGION}/{CITY}/distance

```
requestPath = '/rest/'||sCountry||'/'||sState||'/'||sCity||'/distance'
```

z/OSMF exercise

Request method for submit job REST call: **PUT**

Submit job requestPath: **/zosmf/restjobs/jobs**

tracedd: **MYTRACE**

Email request method: **POST**

Email request URI: **/zosmf/notifications/new**

Reference Material

- **Lab source exercises with answers filled in (plus bonus material)!!!!**
  - **https://github.com/IBM/zOS-Client-Web-Enablement-Toolkit**
- **z/OS 2.4 MVS Programming: Callable Services for High-Level Languages**
  - Complete toolkit documentation
- **z/OS 2.4 MVS System Messages, Volume 6 (GOS – IEA)**
  - Toolkit message documentation
- **z/OS 2.4 MVS System Codes**
  - Toolkit abend '04D'x documentation