IBM Education Assistance for z/OS V2R3

Toolkit REXX support & Toolkit Streaming Send/Receive

Element/Component: z/OS Client Web Enablement Toolkit

Agenda

- Trademarks
- Session Objectives
- Toolkit overview
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Migration & Coexistence Considerations
- Installation
- Session Summary
- Appendix

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.
- Additional Trademarks:
  - None

Session Objectives

- z/OS Client Web Enablement Toolkit quick overview
- REXX support for the toolkit
- HTTP streaming send and receive
- Support for directing HTTP tracing to data sets or zFS files
- JSON parser performance improvements for large JSON text
- Support for the HEAD HTTP request method
- AT-TLS interoperability support

z/OS serving REST APIs – Toolkit Overview

- z/OS platform has for years been labeled “the server of servers” and houses much of the world’s most critical data.
- Enhancements to the z/OS Web serving space through the years have allowed this mammoth workhorse and repository of data to be more easily accessible to other systems.
- Options available to serve countless REST requests coming from just about anywhere into the z/OS mainframe world.
  - IBM HTTP Server powered by Apache
  - WebSphere* Liberty
  - WebSphere Classic
  - z/OS Connect

The z/OS Client Web Enablement Toolkit

The z/OS client web enablement toolkit provides a set of lightweight application programming interfaces (APIs) to enable traditional, native z/OS programs to participate in modern web services applications.

- Pieces of the toolkit:
  - A z/OS HTTP/HTTPS protocol enabler to externalize HTTP and HTTPS client functions in an easy-to-use generic fashion for user's in almost any z/OS environment
  - A z/OS JSON parser which parses JSON coming from any source, builds new JSON text, or adds to existing JSON text.
- The toolkit allows its two parts to be used independently or combined together.
  - Payload processing is separate from communication processing.
- The interfaces are intuitive for people familiar with other HTTP enabling APIs or other parsers
- Easy for newbies

General programming toolkit environment

- Runs in just about any address space
  - Code runs in user's address space
- Supports both authorized and un-authorized callers
- Easy API suite provided
- Multi-language support
  - Include files supplied for C, COBOL, PL/I, Assembler
  - Samples provided for C, COBOL, PL/I and PL/X (internal)

Overview – REXX Support

- Problem Statement / Need Addressed
  - The need for generic JSON parsing and HTTP services on the z/OS platform
  - Toolkit support for COBOL, C, PL/I, Assembler, and PL/X(internally) is great, but adoption of the functionality will not be maximized without support REXX.
- Solution
  - A set of APIs that allow any z/OS application in just about any execution environment to avail themselves of these services
  - A REXX toolkit interface has been created.
- Benefit / Value
  - Almost any application running on z/OS can easily play the role of a client in a client/server web application.
  - Easier to code and execute make support for this language a great win for system programmers and people wanting to whip up an application fast.

Usage & Invocation – REXX Support Host Commands

- Two new host command environments available in REXX
  - HWTJSON
  - HWTHTTP
- Runs in the following REXX environments:
  - TSO
  - ISPF
  - System REXX
  - z/OS UNIX
  - ISV-provided REXX environments
- How to get the host command environment
  - V2R3 and higher (non-ISV-provided environments)
    - host command environments built-in
  - V2R2 and lower (and ISV-provided environments on any release)
    - Host command environment dynamically added by adding hwtcalls('on') to the REXX exec

Usage & Invocation – REXX Support General syntax

- HTTP enabler syntax
  - `address hwthttp 'service_name returncode service_args... diagnostic_stem.'`
- z/OS JSON parser syntax
  - `address hwtjson 'service_name returncode service_args... diagnostic_stem.'`
- Host command return code (special variable rc in REXX) should be 0
  - >0 toolkit service not called, verify command arguments
  - <0 probably an abend code, service may or may not have been called
- The HTTP enabler require z/OS UNIX signals.
  - May wish to issue `call syscalls 'SIGOFF'` depending on your environment

Usage & Invocation – REXX Support
New REXX-only services

- HWTCONST (get toolkit constants)
  - Initializes pre-defined variables in the current REXX variable pool
  - Useful for using the symbolic names as defined in the documentation (such as checking for return codes, setting a particular option, etc..)
  - `address hwthttp 'hwtconst returnCode diag.'`
  - The variable HWT_CONSTANTS is set to a string containing the names of all the variables set
    - Useful for procedures to expose names (`procedure x expose(hwt_constants)`)
- HWTJESCT (encode/decode escape sequences in JSON text)
  - Transforms non-conforming JSON text (not properly escaped) into conforming JSON text (encode). Decode does the reverse.
  - `address hwtjson 'hwtjesct returnCode func srcTxt trgTxt diag'`
    - Where func is either `HWTJ_ENCODE` or `HWTJ_DECODE`

Usage & Invocation – REXX Support
Service not available in REXX

- HWTJGNUV (get number value)
  - REXX has no concept of integers or floating point numbers

Usage & Invocation – REXX Support
Example how to init a connection (REXX)

```
HandleType = HWTH_HANDLETYPE_CONNECTION
address hwthttp "hwthinit ",
                "ReturnCode ",
                "HandleType ",
                "ConnectHandle ",
                "DiagArea."
```

- A connection instance has been created.
- More than one connection can be initialized per address space (if the user has set his dubbing defaults to dub process)
  - But only one connection can be active at a time (Setup MVS Signals only allows one signal setup to be allowed per process)

Usage & Invocation – REXX Support
Example how to set some connection options

```
/*********************************************************************/
/* Set URI for connection to the Federal Aviation Administration (FAA) */
/*********************************************************************/
ConnectionUri = 'http://services.faa.gov'
ReturnCode = -1
DiagArea. = ''
address hwthttp "hwthset ",
                "ReturnCode ",
                "ConnectHandle ",
                "HWTH_OPT_URI ",
                "ConnectionUri ",
                "DiagArea."

/*********************************************************************** */
/* 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.                                           */

/*********************************************************************/
ReturnCode = -1
DiagArea. = ''
address hwthttp "hwthset ",
                "ReturnCode ",
                "ConnectHandle ",
                "HWTH_OPT_COOKIETYPE ",
                "HWTH_COOKIETYPE_SESSION ",
               "DiagArea."
```

- All the connection options can be set prior to the connect service
- Once the connect services is issued, most set option calls will have no effect until the connection is disconnected and reconnected again.

Usage & Invocation – REXX Support
Example showing connecting to HTTP server

```
/*********************************/
/* Call the HWTHCONN toolkit api  */
/*********************************/
ReturnCode = -1
DiagArea. = ''
address hwthttp "hwthconn ",
                "ReturnCode ",
                "ConnectHandle ",
                "DiagArea."
```

- The connection stays persistent from a toolkit perspective
  - Timed-out connections (sockets) will automatically be reconnected at the time of a request if necessary
- A disconnect, reset or terminate of the connection will disconnect the established connection

Usage & Invocation – REXX Support 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.
  - Optionally set the HWTH_OPT_RESPONSEHDR_EXIT option to specify a 4-byte address of the exit to receive control
  - Optionally set the HWTH_OPT_RESPONSEHDR_USERDATA to pass a 4-byte address to the exit as userdata.

Usage & Invocation – REXX Support Response Headers

- In REXX, all the HTTP response headers are stored in a REXX stem variable as specified by setting the optional HWTH_OPT_RESPONSEHDR_USERDATA option.
- Upon return from the HWTHRQST (send request service), the following information is returned regarding the response headers received:
  - the number of response headers received is stored in the .0 stem
  - the name of each response header is stored in the stemname.x stem variable, where x is the from 1 to the number of response headers received.
  - 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.

Usage & Invocation – REXX Support Request Body

- The main data sent on a PUT or POST method to an HTTP REST server is usually sent in the request body
- In non-REXX, the HWTH_OPT_REQUESTBODY option specifies a 4-byte address of a buffer to be sent. The toolkit keeps a binding to the specified address until the HWTHRQST service has completed.
- In REXX, a simple name of a variable is set for the HWTH_OPT_REQUESTBODY option that sets the variable where the response body is stored

Usage & Invocation – REXX Support 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 address of the callback routine is set by specifying a 4–byte address for the HWTH_OPT_RESPONSEBODY_EXIT option.
- An optional HWTH_OPT_RESPONSEBODY_USERDATA specifies a 4-byte value for an input parameter to the exit.
- The body exit can process this data and return back to the toolkit when completed.
- In REXX, the optional HWTH_OPT_RESPONSEBODY_USERDATA option is set to specify the name of a simple name of a variable where the response body is stored

Interactions & Dependencies – REXX Support

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None

Migration & Coexistence Considerations – REXX Support

- None

Installation – REXX Support

- Rolled down to V2R1 and V2R2 via APAR OA50659.

Overview – HTTP streaming

- Problem Statement / Need Addressed
  - Very large request or response bodies may not be able to fit into:
    - contiguous storage, even if the toolkit supported 64-bit
    - any storage available to the address space.
  - De-chunking of very large response bodies can be very inefficient the larger the data is.
- Solution
  - New streaming send/receive functionality
- Benefit / Value
  - Allows virtually an unlimited amount of data to be sent and received by toolkit applications
  - Minimization of data moves of paramount importance
  - More chunked encoding options for application.

Usage & Invocation – HTTP Streaming
Review of sending request body prior to streaming

- For POST and PUT HTTP methods, set HTTP options service (HWTHSET) is used to set HWTH_OPT_REQUESTBODY to a single contiguous buffer address containing the request body
- Send request service (HWTHRQST) sends the HTTP header, request headers and request body all at once
- Fine for small data sent that can fit in a single buffer
- Data cannot be “streamed” to the server

Usage & Invocation – HTTP Streaming
Review of receiving response body prior to streaming

- HWTH_OPT_RESPONSEBODY_EXIT option set (HWTHSET) prior to actual send request (HWTHRQST)
- HWTH_OPT_RESPONSEBODY_USERDATA can also be optionally set
- Data comes back and is delivered to user in their exit all at once
  - Chunked encoded response bodies automatically de-chunked before delivered to the exit.
    - Data movement becomes more expensive as body size increases

Usage & Invocation – HTTP Streaming New Streaming Send (Request Body)

- New Streaming Send Exit option settable by HWTHSET
  - If valid exit address is set for `HWTH_OPT_STREAM_SEND_EXIT`, streaming send will be in effect
- New Streaming user data option settable by HWTHSET
  - Set `HWTH_OPT_REQUESTBODY_USERDATA` to provide exit with an optional buffer of user data to be passed to the streamed send callback routine.
- The toolkit will drive the streaming send exit in a loop to process the data
  - Send exit will specify:
    - address and length of the data buffers to be sent
    - A status value to communicate the state of the data being sent

Usage & Invocation – HTTP Streaming New Streaming Send (Request Body)

- Program can set options via HWTHSET prior to sending the HTTP request to alter streaming behavior
  - **Total length of data (if known).**. A Content-Length other than zero will result in the toolkit setting the request header Content-Length to the proper size.
  - If the Content-Length is set to zero, streaming will interpret this as chunked encoding and auto-chunk the outgoing data.
- Subsequent calls to send exit will allow address and length to be customized on each call
  - Each call makes no assumptions about the buffers used on the previous call. The same buffers can be reused or entirely new buffers supplied each time it is invoked
- Last send will set indicator that there is no more data to send
- Send exit can abort send at any time

Usage & Invocation – HTTP Streaming New Streaming Receive (Response Body)

- New settable options for Streaming Receive exit and Streaming Receive exit user data
  - Note that the `HWTH_OPT_STREAM_RECEIVE_EXIT` is mutually exclusive with the existing `HWTH_OPT_RESPONSEBODY_EXIT` option.
- When configured for Streaming, streaming exit is driven in a loop:
  - Call streaming receive exit the first time with no data and ask for the first set of target buffers and their lengths
  - Subsequent call to exit delivers the data to fill the supplied buffers with as much as was received from the server over the socket at that (streaming favors giving the exit control as fast as possible rather than waiting until the entire buffers are full)
  - Return list describes what data was returned
  - Exit always returns to toolkit with next set of buffers for the toolkit to attempt to fill
  - Exit is notified thru special indicator when all the data has been received
- Receive streaming exit can abort receive at any time

Usage & Invocation – HTTP Streaming New Streaming Receive (Response Body)

- Received chunked data will:
  - Fill the user-supplied buffer as much as possible
  - Give aA vector list in a compatible format with the writev() z/OS XL C/C++ library function will be returned
    - Each chunk's starting address and length is identified
    - Allows caller to easily harden data without any additional data moves
- For non-chunked data, the vector list will contain <= the number of buffers supplied by the caller.

Usage & Invocation – HTTP Streaming
Data structures used by the streaming exits

- Progress descriptor structure:
  - Provides an orientation for the exit by specifying:
    - URI of request, total bytes, userdata (optional), and the HTTP status line information.
- Send/Receive list:
  - List of buffers and their lengths to be sent from or written into by the streaming exits

Usage & Invocation – HTTP Streaming
Send exit parameter list and send states

- Streaming send exit:
  - Progress descriptor structure
  - Send State
  - List of populated send buffers to send to server along with their lengths
  - The number of buffers being sent on this send

- List of possible send states:
  - HWTH_STREAM_SEND_CONTINUE
  - HWTH_STREAM_SEND_EOD
  - HWTH_STREAM_SEND_ABORT
  - HWTH_STREAM_SEND_COMPLETE
  - HWTH_STREAM_SEND_ERROR

Usage & Invocation – HTTP Streaming Receive exit parameter list

- Streaming receive exit:
  - Progress descriptor structure
  - Receive State
  - Supply list
    - Array of address and length of the next pieces of the response body should be placed (owned by the exit)
  - Supply list dimension
    - How many buffers the supply list contains
  - Return list
    - Array of address and lengths which describe the pieces of the response body data already written. (owned by the toolkit)
  - Return list dimension
    - How many chunks of data in the return list

Usage & Invocation – HTTP Streaming Receive exit states

- List of possible receive states:
  - HWTH_STREAM_RECEIVE_CONTINUE
  - HWTH_STREAM_RECEIVE_EOD
  - HWTH_STREAM_RECEIVE_ABORT
  - HWTH_STREAM_RECEIVE_COMPLETE
  - HWTH_STREAM_RECEIVE_ERROR

Usage & Invocation – HTTP Stream
Things not included with Streaming

- No REXX support for streaming in initial release
  - Additional coding required to support the REXX language
  - Large data transfers probably not a perfect fit to be written in REXX
- No samples shipped at GA
  - Possible to ship a sample soon after

Interactions & Dependencies – HTTP Streaming

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None
-

Migration & Coexistence Considerations – HTTP Streaming

- None

Installation – HTTP Streaming

- None

Overview – HTTP tracing enhancements

- Problem Statement / Need Addressed
  - HTTP tracing goes only to standard out.
  - Not practical for many programming environments
- Solution
  - Provide a mechanism to direct the trace output to a data set or zFS file.
- Benefit / Value
  - Additional flexibility to have an application gather necessary tracing information to aid in the problem determination for REST API interactions

Usage & Invocation – Review of HTTP Enabler Problem Determination

- **Return Code from service**
  - Specific return code can give explanation
- **DiagArea**
  - Many times provides detailed explanation.
- **Status values returned in callback routines**
  - Provides HTTP status values from server
- **HWTH_VERBOSE set option**
  - Toolkit directs many trace-like messages to the standard output of the application. Useful during debugging.
  - Can be directed to standard output or to a preallocated MVS data set or zFS file thru use of the HWTH_OPT_VERBOSE_OUTPUT option.
- **SOCKAPI CTRACE option**
- **System SSL tracing**

Usage & Invocation – HTTP Tracing Enhancements

- HWTH_OPT_RESPONSEBODY_EXIT option set (HWTHSET) prior to actual send request (HWTHRQST)
- HWTH_OPT_RESPONSEBODY_USERDATA can also be optionally set
- Data comes back and is delivered to user in their exit all at once
  - Chunked encoded response bodies automatically de-chunked before delivered to the exit.
    - Data movement becomes more expensive as body size increases

Usage & Invocation – HTTP Tracing Enhancements

```
address hwthttp "hwthset ",
                "ReturnCode ",
                "ConnHandle ",
                "HWTH_OPT_VERBOSE ",
                "HWTH_VERBOSE_ON ",
                "DiagArea."

traceDD = 'MYTRACE'
/* Allocate the data set here */
address hwthttp "hwthset ",
                "ReturnCode ",
                "ConnectionHandle ",
                "HWTH_OPT_VERBOSE_OUTPUT ",
                "traceDD ",
                "DiagArea."
```

- The DD name above must represent either:
  - a pre-allocated traditional z/OS data set which is a physical sequential (DSORG=PS) with a record format of unblocked variable (RECFM=V) or Undefined (RECFM=U) and expandable (non-zero primary and secondary extents). The DD must also specify a DISP=OLD disposition.
  - a zFS or HFS file.
- When a connect, sendRequest or disconnect are issued, detailed trace will be written to the data set specified by the MYTRACE DD.

Interactions & Dependencies – HTTP Tracing Enhancements

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None

Migration & Coexistence Considerations – HTTP Tracing Enhancements

- None

Installation – HTTP Tracing Enhancements

- Rolled down to V2R1 and V2R2 via APAR OA49002.

Overview – z/OS JSON parser performance improvement

- Problem Statement / Need Addressed
  - z/OS JSON parser is a very quick parser
  - As the JSON text grows in size, the parse and create operations decline in performance
- Solution
  - Allow the parser to adapt to the size of the JSON text better
- Benefit / Value
  - Even very large JSON text streams parse much faster (HWTJPARS) than before and create entry requests (HWTJCREN) occur significantly faster than before.

Usage & Invocation – z/OS JSON parser performance improvement

- No external changes.
- The z/OS JSON parser is much smarter about the management of the internal JSON mapping of the JSON text stream.
- For very large JSON bodies, the internal representation can span multiple obtained storage areas.
- The insertion point for adding an new entry to an object or array is learned significantly faster than before, making insertion orders of magnitude faster than before when there are many entries already in the object or array.

Interactions & Dependencies – z/OS JSON parser performance improvement

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None

Migration & Coexistence Considerations – z/OS JSON parser performance improvement

- None

Installation – z/OS JSON parser performance improvement

- Rolled down to V2R1 and V2R2 via APAR OA50192.

Overview – Support for HEAD HTTP request method

- Problem Statement / Need Addressed
  - The HTTP enabler supports GET, PUT, POST and DELETE (the 4 required CRUD methods required for REST API interaction
  - Sometimes, the user simply wants to retrieve the response headers without the actual data in the response body for performance reasons. Currently there is no way to do this operation
- Solution
  - Support the HTTP HEAD request method
- Benefit / Value
  - Programs wishing to obtain only the meta data information on the GET without actually incurring the expense of sending the actual data can optimize this flow for better performance.

Usage & Invocation – Support for HEAD HTTP request method

- HWTHSET can now be invoked using a new constant value `HWTH_HTTP_REQUEST_HEAD` can be specified for setting the `HWTH_OPT_REQUESTMETHOD` option.
- The HEAD method is identical to GET except that the server does not return a message-body in the response. The metadata contained in the HTTP headers in response to a HEAD request will be identical to the information sent in response to a GET

Interactions & Dependencies – Support for HEAD HTTP request method

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None

Migration & Coexistence Considerations – Support for HEAD HTTP request method

- None

Installation – Support for HEAD HTTP request method

- Rolled down to V2R1 and V2R2 via APAR OA51673.

Overview – AT-TLS interoperability support

- Problem Statement / Need Addressed
  - HTTPS communications require the toolkit application to point to the certificate store that SSL is to use during its handshake with the server.
  - Many applications do not want the application layer to make these security decisions and instead, would like to use the networking layer's policies and definitions to determine if the request should be running with HTTPS and if so, where to get the certificates from.
- Solution
  - Interoperate with AT-TLS
- Benefit / Value
  - The toolkit will be aware if AT-TLS auto-upgraded any connection using SSL. The toolkit will then treat this request as an HTTPS request, allowing an application program to not have to specify HTTPS constructs within their program.

Usage & Invocation – AT-TLS interoperability support

- Application Transparent – TLS is basically stack-based TLS
  - TLS process performed in TCP layer (via System SSL) without requiring any application change (transparent)
  - AT-TLS policy specifies which TCP traffic is to be TLS protected based on a variety of criteria
    - Local address, port
    - Remote address, port
    - z/OS userid, jobname
    - Time, day, week, month

Usage & Invocation – AT-TLS interoperability support

- When option HWTH_OPT_USE_SSL is set to:
  - HWTH_SSL_NONE
    - The toolkit will tolerate AT-TLS auto-upgrading the connection to an SSL connection and will be aware that this connection is secure
    - Toolkit will treat connection as HTTPS, even it did not actually initiate the SSL handshake itself
    - If connection is not auto-upgraded or AT-TLS is not present, toolkit will process this connection as straight HTTP as it does today.
  - HWTH_SSL_USE
    - The toolkit will continue to support the application specifying its own security credentials provided that the connection is not auto-upgraded by AT-TLS

Interactions & Dependencies – AT-TLS interoperability support

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - None

Migration & Coexistence Considerations – AT-TLS interoperability support

- None

Installation – AT-TLS interoperability support

- Will be rolled down to V2R1 and V2R2 via APAR OA50957 prior to V2R3 GA.

Session Summary

- z/OS Client Web Enablement Toolkit quick overview
- REXX support for the toolkit
- HTTP streaming send and receive
- Support for directing HTTP tracing to data sets or zFS files
- JSON parser performance improvements for large JSON text
- Support for the HEAD HTTP request method
- AT-TLS interoperability support

Appendix

- All information will be documented in the ***z/OS MVS Programming: Callable Services for High-Level Languages*** publication.