Seriously? JSON parsing and HTTP services in native z/OS? Absolutely!

Steve Warren
z/OS Client Web Enablement Toolkit Technical Lead
IBM

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.
Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*, AS/400®, e business(logo)®, DBE, ESCO, eServer, FICON, IBM®, IBM (logo)®, iSeries®, MVS, OS/390®, pSeries®, RS/6000®, S/30, VM/ESA®, VSE/ESA, WebSphere®, xSeries®, z/OS®, zSeries®, z/VM®, System i, System i5, System p, System p5, System x, System z, System z9®, BladeCenter®

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
BMC Mainview AutoOPERATOR is a trademark of the BMC Software Corporation
CA Ops/MVS is a trademark of the CA Technologies corporation.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.
zCostManagement is a trademark of the zCostManagement Corporation.
zPrice Manager is a trademark of the zIT Consulting Corporation

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

- Web applications (quick overview)
  - What and why of JSON?
  - What is REST?
  - What is missing on z/OS?
- Introduction to the toolkit
- z/OS JSON Parser details
  - Usage
- z/OS HTTP/HTTPS protocol enabler details
  - Usage
  - Problem determination
- Reference material

Web applications (JSON, HTTP and REST)

Moving Beyond the Browser

Businesses Are Evolving

Web APIs are the new, fast-growing business channel

Not having an API today is like not having a Web Site in the 90s

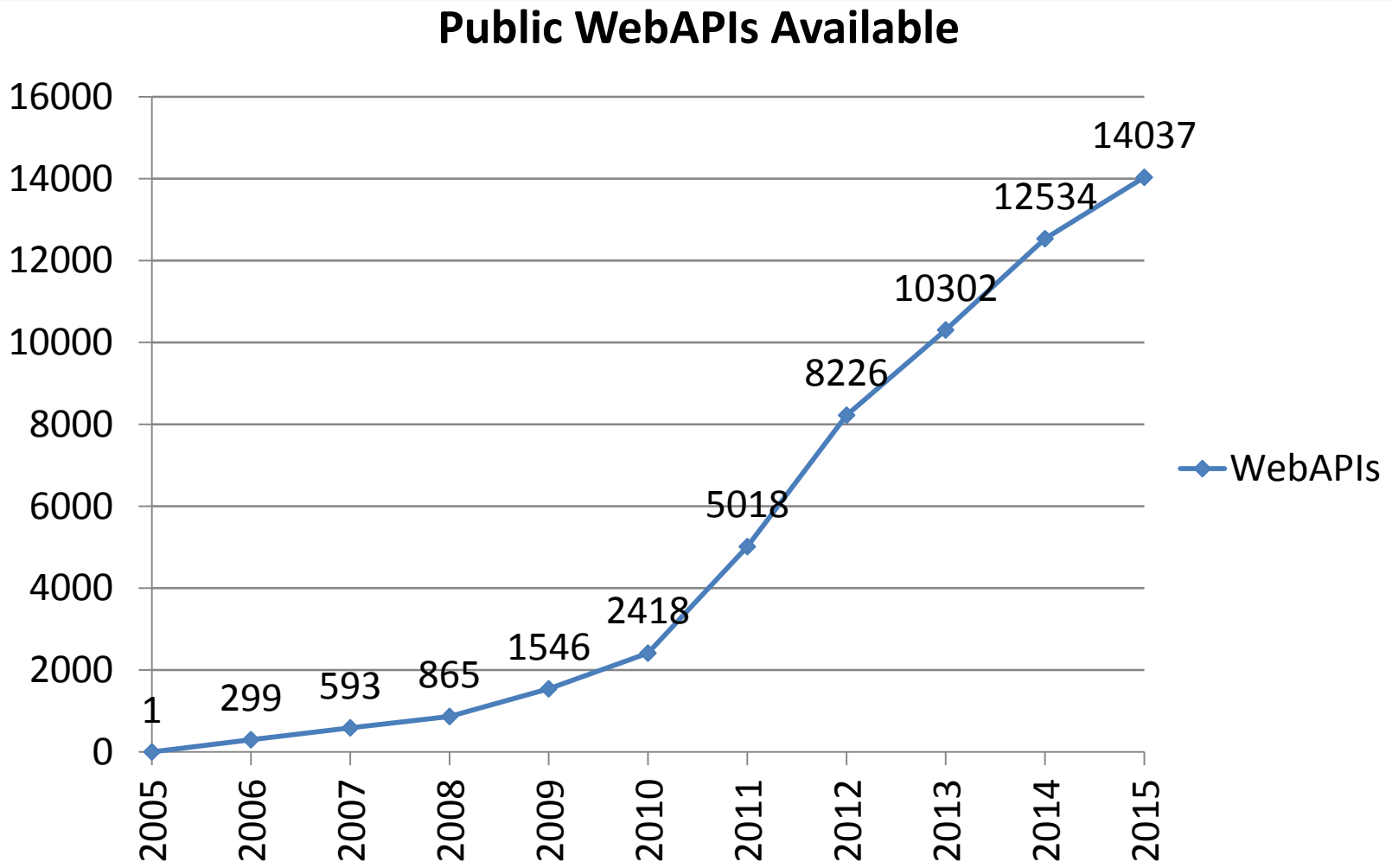
"***$7bn worth of items*** on eBay through APIs"
Mark Carges (Ebay CTO)

The API which has easily ***10 times more traffic*** then the website, has been really very important to us."
Biz Stone (Co-founder, Twitter)

"The adoption of Amazon's Web services is currently driving more network activity then everything Amazon does through their traditional web sites."
Jeff Bar (Amazon evangelist) / Dion Hinchcliffe (Journalist)

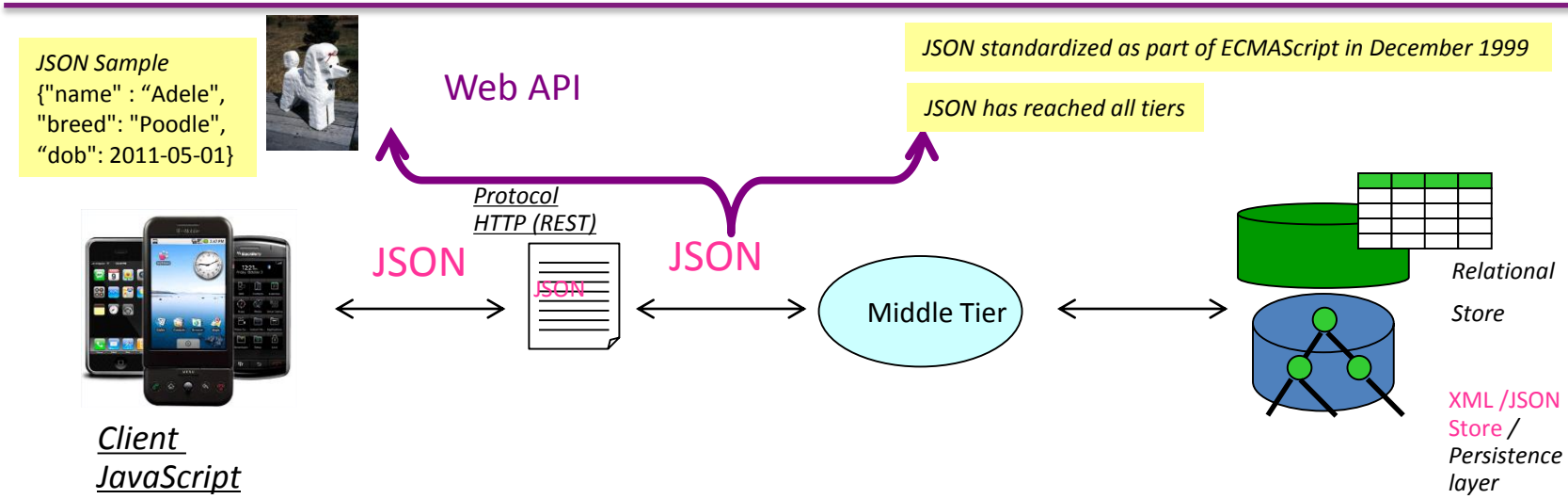
Web 1994 was the "get me a domain and a page" era.
Web 2000 was the "make my page(s) interactive and put people on it" era.
Web 2010 is the "get rid of pages and glue APIs and people together" era.
Robert Scoble (Author of tech blog Scobleizer)

Growth in Publically Published Web APIs



Source: Programmable Web (some intermediate numbers extrapolated)

JSON : The Exchange Notation For Mobile Devices



- With the increased popularity of Web APIs (literally thousands of Web APIs) and the use of Mobile Devices
  - User Interfaces usually have a JavaScript component
    - JSON is the data structure for JavaScript
  - JSON is integrated with JavaScript and Java and other languages (through libraries)
  - The JSON trend is developer driven and is reaching all tiers (UI, Middle Tier, Data Tier)

Aspects of JSON:

No namespaces

No schemas

No mixed content support Mixed content example: hello Adelehow are you

JSON Penetration: Web API Trend Towards JSON

- "As more and more Web and mobile applications utilize APIs to drive their respective front ends, performance becomes an emerging concern. XML, long used as a method for exchanging data, is giving way to JSON, now considered the gold standard. " - Programmable Web 2013
- "JSON's simplicity has made it a favored data exchange format" – Mashery 2014
- "In general, JSON wins the battle on brevity which is why many web applications are using JSON for RESTful data transfer." – GCN 2014

JSON example

```
{
"firstName": "Steve",
"lastName": "Jones",
"age": 46,
"address": {
        "streetAddress": "123 Anywhere Ave",
        "city": "Poughkeepsie",
        "state": "NY",
        "postalCode": "12601",
        "country": "USA"
        },
"phone": [
        {
          "type": "mobile",
          "number": "914 555 5555"
        },
        {
          "type": "home",
          "number": "845 555 1234"
        }
    ]
}
```

Name – value pairs

- Name in quotes
- Value one of the types below

Objects – { }

Arrays – [ ]

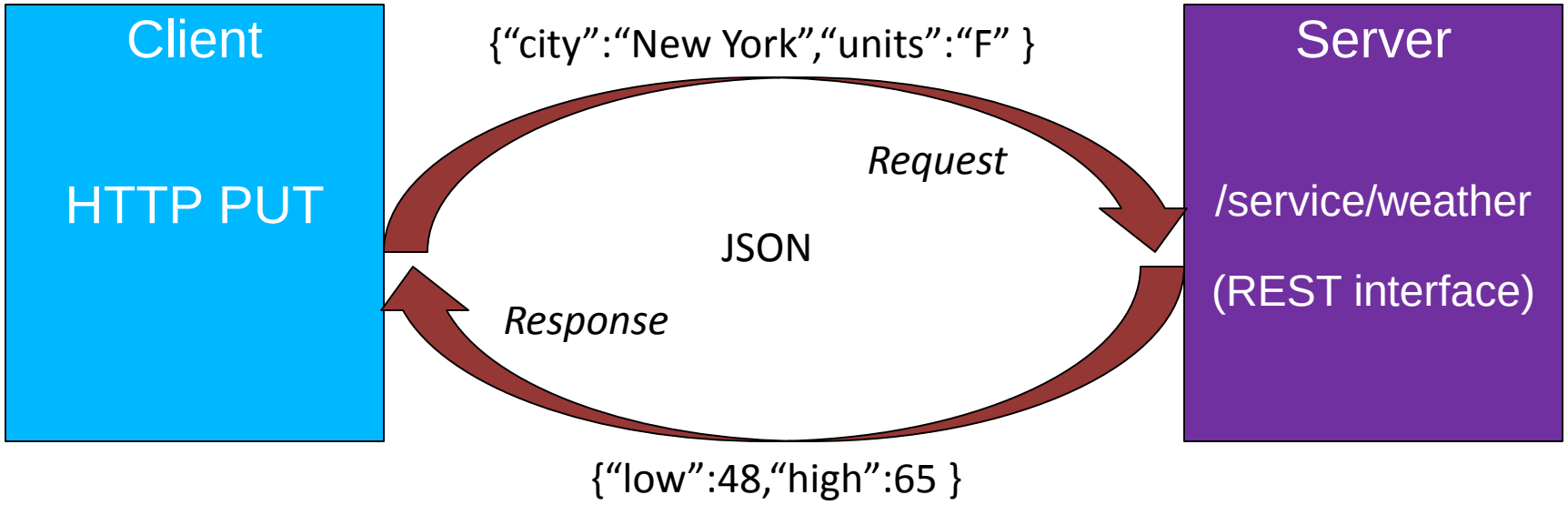
Object entries

- Other objects or arrays
- String – “ ”
- Numbers
- Boolean
- Null

What are REST APIs?

- Web applications are a client/server programming model using a request/response protocol.
- RESTful applications are web applications that follow a simple set of architectural constraints such as:
  - using standard HTTP methods
  - stateless servers
  - using URIs (URLs) strictly to identify the server resource being modified or interrogated
  - sending data back and forth in human-readable form
- REST APIs are defined by a server
  - What URI should be interrogated
  - Which HTTP service should be used against that URI
  - The format of the data to be sent back and forth

30,000 foot view of REST / HTTP / JSON



No Generic z/OS Client Web Services

- A few z/OS implementations here and there
  - Rexx & Curl in z/OS UNIX (USS)
  - Socket from COBOL
  - Apache http client from Java
  - DB2 REST UDF
  - CICS Sockets
  - WOLA & Liberty Profile
- No generic web services available to all z/OS clients
- No general usage JSON parser available in all z/OS environments

Intro to the toolkit

Overview

- Problem Statement / Need Addressed
  - The need for generic JSON parsing and HTTP services on the z/OS platform
- Solution
  - A set of APIs that allow any z/OS application in just about any execution environment to avail themselves of these services
- Benefit / Value
  - Almost any application running on z/OS can easily play the role of a client in a client/server web application.

Introducing 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.

- 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
  - Multi-language samples provided

z/OS JSON parser details

Elements of the z/OS JSON Parser

Types of services:

- Initialize / Terminate
  - Create and free the memory space (parser instance) required by the parser
- Parse
  - Assigns a particular JSON text stream to a parser instance
  - Check input JSON text for syntax errors
  - Creates an internal representation of the JSON
- Traverse
  - Data format is not well known
  - Program learns what data was passed using several traversal services
- Search
  - Quickly locate a particular name in a name/value pair
- Create / Serialize
  - Create JSON text from scratch or insert new JSON text into the existing text.

z/OS JSON parser environment

- z/OS Client Toolkit execution environment:
  - Supports both authorized and un-authorized callers
    - Allow supervisor or problem state callers running in any PKM
  - Supports task and SRB mode invokers
  - Supports cross-memory mode invokers
  - Recovery needed by caller

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 - Traverse

Traversal methods return an object or element handle

These handles are used to address a particular object or object entry

- An object handle is returned when the data value type of an name value pair is an object. This object handle can then be used to find out the number of entries in the object and to traverse all the elements in the object.
- An entry handle is returned when addressing a particular entry. It is used to drill down into the contents of the entry.

z/OS JSON parsing services

- 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

z/OS Client Toolkit JSON Parser Syntax (Initialize Parser Instance)

- Call HWTJINIT (returnCode, maxParserWorkAreaSize, parserHandle, diagArea)
  - returnCode (output)
  - maxParserWorkAreaSize (input) represents the maximum size of storage the parser can consume during parser functions. This maximum size is not necessarily obtained when the JSON instance is initialized, but allows the parser to consume UP to this value. Defaults to unlimited if the value is specified as zero.
  - parserHandle (output) is a 12-byte character value generated by the parser which contains a handle to be used on all subsequent JSON parser services for this parser instance. This instance contains all of the data structures and storage areas required for the parser to run and to run efficiently.
  - diagArea (output) 4-byte address pointing to a 132-byte storage area mapped in the provided include files. It is comprised of a 4-byte reason code field and a 128-byte error text field.

z/OS Client Toolkit JSON Parser Syntax (Parse existing JSON text)

- Call HWTJPARS (returnCode, parserHandle, JSONTextAddr, JSONTextLen, objectHandle, diagArea);
  - returnCode (output)
  - parserHandle (input) 12-byte character value representing the JSON parser instance to be used for the new JSON text to be parsed.
  - JSONTextAddr (input) 4-byte address representing the actual storage location of the JSONText to be parsed.
  - JSONTextLen (input) 4-byte value representing the length of the JSONText storage area supplied in the above *JSONTextAddr* input parameter.
  - diagArea (output) 4-byte address pointing to a 132-byte storage area mapped in the provided include files. It is comprised of a 4-byte reason code field and a 128-byte error text field.

z/OS Client Toolkit JSON Parser Syntax (Traversal example)

- Call HWTJGJST (returnCode, parserHandle, objOrEntryValueHandle, JSONType, diagArea)
  - **objOrEntryValueHandle** (input) 4-byte value representing a previously returned object or entry handle.
  - **JSONType** (output) 4-byte returned value specifying the JSON type.

  –Example: JSONType returned is HWTJ_OBJECT_TYPE
- Call HWTJGNUE (returnCode, parserHandle, objectHandle, numOfEntries, diagArea)
  - **objectHandle** (input) 4-byte value representing a previously returned handle.
  - **numofEntries** (output) 4-byte returned value with the number of entries in the referenced object.
- Now loop thru all the entries, invoking HWTJGOEN (Get Object Entry) for each entry in the object.

z/OS Client Toolkit JSON Language Support

Include files and sample programs provided in:

- C/C++
- COBOL
- PL/I
- Assembler (Include file only)

Installation – z/OS JSON Parsing Services

- V2R2 – None (in the base)
- V2R1 – Install APAR OA46575 and re-IPL
- External message to know the toolkit is installed and ready to go:
  - HWT001I message will appear in the syslog stating the toolkit is enabled

z/OS HTTP/HTTPS protocol enabler details

Usage & Invocation – z/OS HTTP Services

- Provides similar functionality to existing open-source libcurl HTTP/HTTPS interface
  - Interface is very similar
  - Underlying code is z/OS-specific and not ported in any way

HTTP features supported by toolkit

- HTTPS connections
- HTTP cookies management
- Proxies
- URI redirection
- Basic client authentication
- Chunked encoding

HTTP Services execution environment

- Same as the JSON part of the toolkit except:
  - Task mode callers only
  - Key zero callers not allowed
  - OMVS segment required for address space using HTTP enabler
- Recovery recommended by calle

Two aspects of HTTP toolkit app

- Connection
  - The socket created between the client and server for the purpose of exchanging information
  - Can be established as SSL/TLS, thru a particular local IP addresses, thru a proxy, using a particular IP stack
- Request
  - An HTTP request sent over a previously established connection
  - Requests not tightly coupled to a particular connection

Elements of the z/OS HTTP enabler

Types of services:

- Initialize / Reset / Terminate
  - Create, reset or free the memory space (connection or request instance) required by the HTTP enabler
- Set options
  - Prepare the connection or request instance with the desired configuration options
  - Options set one at a time
- Connect / Disconnect
  - Establish or disconnect a connection instance with a server using sockets
  - If SSL/TLS has been selected, the connect will handle all SSL interactions
- Send request
  - Couple a request with previously connected connection instance.
- Set Linked List service
  - Utility service that creates a linked list of data objects of the same type. For example: create a list of HTTP headers and set the HWTH_OPT_HTTPHEADERS option to the list created by this service.

Usage of the z/OS HTTP enabler services

- Simple services allow the user to easily build an HTTP/HTTPS request, step-by-step.
  - Init a connection.
  - Set the desired HTTP connect options.
  - Issue the HTTP connect.
  - Init a request.
  - Set the desired HTTP request options.
  - Issue the HTTP request.
  - Process the response.
  - Term the request or re-use.
  - Term the connection

HTTP Services – Connection Options

- Various settable options for preparing an HTTP connect for the set command include:
  - HWTH_OPT_HTTP_VERSION – values are 4-byte integers representing the HTTP version desired. The following constants are provided:
    - HTTP_VERSION_NONE
    - HTTP_VERSION_1_0
    - HTTP_VERSION_1_1
  - HWTH_OPT_URI – valid values are either a v4 or v6 IP address, or hostname
    - Example http://192.168.0.1 or http://[2001:1890:1112:1::20]/ or http://www.example.com
  - HWTH_OPT_PORT – Value specifying which remote port number to connect to, instead of the one specified in the URL or the default HTTP or HTTPS port.
  - HWTH_OPT_IPSTACK – Optional value 1 to 8 character z/OS TCP/IP stack to be used by the connection
  - HWTH_OPT_LOCALIPADDR – Optional outgoing local IP address
  - HWTH_OPT_LOCALPORT – Optional outgoing local port
  - HWTH_OPT_SNDTIMEOUTVAL – Sending timeout value
  - HWTH_OPT_RCVTIMEOUTVAL – Receiving timeout value

HTTP Services – SSL Options

- SSL support options include:
  - HWTH_OPT_SSLVERSION – sets the SSL versions to be supported by this HTTP request. More than one version may be selected. (e.g. TLS1.2, TLS1.1, TLS1.0, SSLv3)
  - HWTH_OPT_SSLKEYTYPE – Specifies the manner the key will be supplied to this HTTPS request. The following constants are provided:

    SSLKEYTYPE_KEYDBFILE

    SSLKEYTYPE_KEYRINGLABEL

    SSLKEYTYPE_KEYRINGNAME

  - HWTH_OPT_SSLKEY – Specifies the value of the key. The value specified depends on the value set by SSLKEYTYPE.

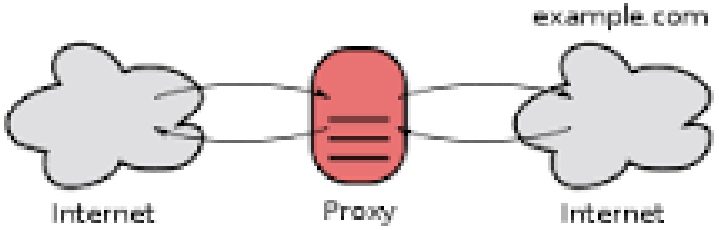
    For SSLKEYTYPE_KEYDBFILE - represents path and name of the key database file name

    For SSLKEYTYPE_KEYRINGNAME – represents the RACF key ring name

  - HWTH_OPT_SSLKEYSTASHFILE – specifies the stash file of the key database file. Only valid if SSLKEYTYPE_KEYDBFILE is specified. Ignored in all other cases.
  - HWTH_OPT_SSLCLIENTAUTHLABEL – optional label that represents a client certificate if SSL client authentication is requested by the server.

HTTP Services – Proxy Options

- Proxy support options include:
  - HWTH_OPT_PROXY – set the HTTP proxy to user. Specified the exact same as HWTH_OPT_URI above.
  - HWTH_OPT_PROXYPORT – specify the proxy port to connect to. Specified the exact same as HWTH_OPT_PORT above



HTTP Services – Redirect Options

- Redirect options include:
  - HWTH_OPT_MAX_REDIRECTS – maximum number of redirects to follow for a given request.
  - HWTH_OPT_XDOMAIN_REDIRECTS – are cross-domain redirects allowed?
  - HWT_OPT_REDIRECT_PROTOCOLS – do you allow the HTTP/HTTPS protocol to be upgraded and/or downgraded on a redirect?

HTTP Services – Cookie Options

- Cookie support options include:
  - HWTH_OPT_COOKIETYPE – sets cookie handling type

    COOKIETYPE_NONE – cookie engine not activated

    COOKIETYPE_SESSION – cookie engine enabled – cookies automatically sent, but end when connection ends

    COOKIETYPE_PERSIST - cookie engine enabled – cookies automatically sent, cookies saved to output buffer when connection endsin
  - HWTH_OPT_COOKIE_INPUT_BUFFER – specifies input cookie data store
  - HWTH_OPT_COOKIE_OUTPUT_BUFFER – specifies output cookie location for a cookietype of COOKIETYPE_PERSIST

HTTP Services – Connect/Disconnect

- Connect to HTTP Server
  - Attempts to connect using all of the attributes set by previous set functions using this connection handle.
  - If successful, this connection is eligible to be used for HTTP/HTTPS requests.
- Disconnect from HTTP Server
  - Attempts to disconnect the connection created by the Connect service.
  - If successful, the connection element will be disconnected but all attributes associated with the connection still be intact. If a subsequent Connect is issued, the attributes specified on the prior Connect will be used.

HTTP Services – Request Options

- Setting Request Options
  - Use same Set service as for HTTP connect options
    - Various settable options for preparing an HTTP request include:

      HWTH_OPT_REQUEST – values are 4-byte integers representing the desired HTTP CRUD request methods:

      HTTP_REQUEST_GET

      HTTP_REQUEST_PUT

      HTTP_REQUEST_POST

      HTTP_REQUEST_DELETE

HTTP Services – Request Options

- Various settable options for preparing an HTTP request (continued):
  - HWTH_OPT_HTTPHEADERS – 4-byte pointer to a linked list of HTTP request headers. These request headers were chained via the HWTHSLST link list append service.
  - HWTH_OPT_COOKIE – specific cookies to send apart from normal cookie handling
  - HWTH_OPT_REQUESTBODY – 4-byte pointer to a request body data (useful on HTTP POST and PUT operations).
  - HWTH_OPT_RESPONSEHEADER_EXIT – 4-byte address of the program to receive control once for each response header received by the application.
  - HWTH_OPT_RESPONSEHEADER_USERDATA – 4-byte address of the optional user data buffer to be passed into the response header exit when it receives control.
  - HWTH_OPT_RESPONSEBODY_EXIT – 4-byte address of the program to receive control when the response body is received.
  - HWTH_OPT_RESPONSEBODY_USERDATA – 4-byte address of the optional user data buffer to be passed into the response body exit when it receives control.

HTTP Services – Request Options

- HTTP authorization options:
  - HWTH_OPT_HTTPAUTH – Do I want HTTP basic client authentication?
  - HWTH_OPT_USERNAME and HWTH_OPT_PASSWORD must be set if basic client authentication is selected.
- HTTP request body and response body translate functions
  - HWTH_OPT_TRANSLATE_REQBODY – translate request body from EBCDIC to ASCII automatically.
  - HWTH_OPT_TRANSLATE_RESPBODY – translate response body from ASCII to EBCDIC automatically.

HTTP Services – Send Request

- Send Request to HTTP server
  - Attempts to send the request represented by the request handle using the connection represented by the connection handle.
  - Receives the appropriate response.

HTTP Services – Processing responses

- Response Header and Response Body exits
  - Callback exits receive control for each response header received and once for the response body.
  - The response header exit is called in series (meaning serially) so that only one response header is presented to the application at any one time. If the user's exit does not return control to the toolkit, the next response header will not be delivered.
  - Response header can reject the rest of the request at any time by setting the return code back to the toolkit to "abort"

HTTP Services – Chunked encoding

- Responses with chunked encoding present
  - Toolkit supports the chunked encoding data transfer method (Transfer-encoding: chunked).
  - Automatically de-chunks data sent from the server using the chunked encoding method.
    - The response body exit does not need to handle the various chunks; rather, the data is delivered to the exit already decoded.
    - If the chunked data contains trailer headers, the header exit will be invoked (once for each trailer header) prior to this routine receiving control.
  - Note: The toolkit ignores chunk extensions

z/OS Client Toolkit HTTP Language Support

Include files and sample programs provided in:

- C
- COBOL – sample delivered soon after V2R2 GA via APAR OA49002
- PL/I – sample delivered soon after V2R2 GA via APAR OA49002
- Assembler (Include file only)

Installation – z/OS HTTP Enabler Services

- V2R2 – None (in the base)
- V2R1 – Install APAR OA46622 after GA of V2R2 and re-IPL
- External message to know the toolkit is installed and ready to go:
  - HWT001I message will appear in the syslog stating the toolkit is enabled

Problem determination – HTTP Enabler

- **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.
- **SOCKAPI CTRACE option**
- **System SSL tracing**

HTTP Enabler C code snippet

```
handleType = HWTH_HANDLETYPE_CONNECTION;
hwthinit(pm_rc, handleType, handle, diagArea);

option = HWTH_OPT_URI;
charOptionVal = "http://services.faa.gov";
optionValLen  = strlen(charOptionVal);
hwthset(rc, handle, option, *charOptionVal, optionValLen, diagArea);

hwthconn(rc, handle, diagArea);
```


HTTP Enabler C code snippet (continued)

```
handleType = HWTH_HANDLETYPE_HTTPREQUEST;
hwthinit(pm_rc, handleType, reqHhandle, diagArea);

option = HWTH_OPT_REQUESTMETHOD;
intOptionVal = HWTH_HTTP_REQUEST_GET;
optionValLen  = sizeof(intOptionVal);
hwthset(rc, reqHandle, option, *intOptionVal, optionValLen, diagArea);

loption = HWTH_OPT_URI;
const char airportPrefix??(20??) = "/airport/status/";
memset(charOptionVal,0,30);
sprintf(charOptionVal,"%s%s\0",airportPrefix,JCLparm);
charOptionValPtr = (char *)&charOptionVal;
optionValLen  = strlen(charOptionVal);
hwthset(rc, reqHandle, option, *charOptionVal, optionValLen, diagArea);

hwthrqst(rc, handle, reqHandle, diagArea);
```


Reference Materials

Toolkit Reference Materials

- **z/OS 2.2 MVS Programming: Callable Services for High-Level Languages**
  - Complete toolkit documentation
- **z/OS 2.2 MVS System Messages, Volume 6 (GOS**
  - Toolkit message documentation
- **z/OS 2.2 MVS System Codes**
  - Toolkit abend ’04D’x documentation

REST easy on z/OS

Introducing the z/OS Client Web Enablement Toolkit

BY STEVE WARREN

The number of web service applications on the internet has increased significantly in recent years. RESTful applications that use HTTP or HTTPS as a means of communication and send JSON or XML data is as common as it gets in the mobile, client/server world.

Wouldn't it be cool if your existing z/OS applications running in a traditional environment could easily ramp up to play in the game as well as through a set of base z/OS services available to most programs on z/OS?

If you're excited about these options, welcome to the new z/OS Client Web Enablement Toolkit! Built into the base of the z/OS operating system, the toolkit provides a lightweight solution to enable these applications to more easily participate in this client/server space by providing the following built-in features:

- A z/OS JSON parser that can be used to parse JSON text that comes from any source and create new JSON text or add to existing JSON text.
- A z/OS HTTP/HTTPS protocol enabler which uses interfaces similar to other industry-standard APIs.

Just about all environments on z/OS can avail themselves of these new services. Traditional z/OS programs that run in native z/OS have little or no options that they can easily use to participate in web services applications. Programs running as a batch job, as a started procedure or in almost any address space on a z/OS system now have APIs that can be used in a similar manner to any standard z/OS APIs provided by the operating system.

Furthermore, programs can use these APIs in the programming language of their choice. You can use C/C++, COBOL, PL/I, and Assembler languages, and samples are provided for C/C++, COBOL, and PL/I.

Would you like to hear more about the parts of the toolkit and get a small taste of what you can do?

z/OS JSON parser

Suppose that you would like to be able to make sense of a large JSON text file that was sent to you from a web server that you are communicating with. The new z/OS JSON parser can do the heavy lifting for you.

The following questions can help you decide which style of parsing is best for you:

Welcome to the new z/OS Client Web Enablement Toolkit!

- Do you know the format of the data that is being returned?
- Are you looking for specific fields in a particular format?
- Do you need to learn about all the data that is returned?

Based on your answer, you can choose the "search" style, the "traversal" style, or a combination of both. The "search" style looks for specific key values in the text stream and then finds the values that are associated with those key values. The "traversal" style starts the traversal parser services to recursively move through the text stream until it learns what was sent.

In either case, a program that uses the JSON parsing services follows this format:

1. Start the JSON parser initialize service (HWTJINIT) to create a parsing instance.
   Tip: You can go wild here and create as many parser instances as your application requires. Each parser instance allows the z/OS JSON parser the ability to separately manage the parsing of a JSON text stream. The more instances you have, the more concurrent JSON text streams you can parse.
2. Call the JSON Parse service (HWTJPARS) to have the parser validate the syntax of the text stream and create an internal representation of the JSON text data. Once the data is parsed, it allows all subsequent services to run faster

- **z/OS Hot Topics magazine (August 2015)**
  - **REST easy on z/OS – Introducing the z/OS Client Web Enablement Toolkit** (pg. 26-27)

Questions?

Contact Info

Steve Warren
IBM
email: swarren@us.ibm.com
: @StevieWarr2