IBM Education Assistance for z/OS V2R2

Item: z/OS Client Web Enablement Toolkit
Element/Component: z/OS Client Web Enablement Toolkit

Material current as of May 2015

Agenda

- Trademarks
- Presentation Objectives
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Installation
- Presentation Summary
- Appendix

Trademarks

Presentation Objectives

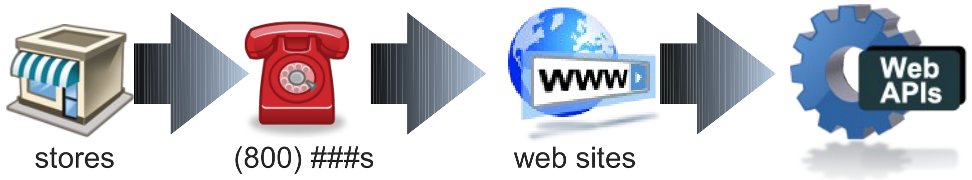
- Introduction of the z/OS Client Web Enablement Toolkit
- Provide a business case why the toolkit is important on the z/OS platform
- Provide usage particulars for both parts of the toolkit
  - z/OS JSON parser
  - z/OS HTTP enabler
- Reference materials

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
  - Any application running on z/OS can easily play the role of a client in a client/server web application.

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

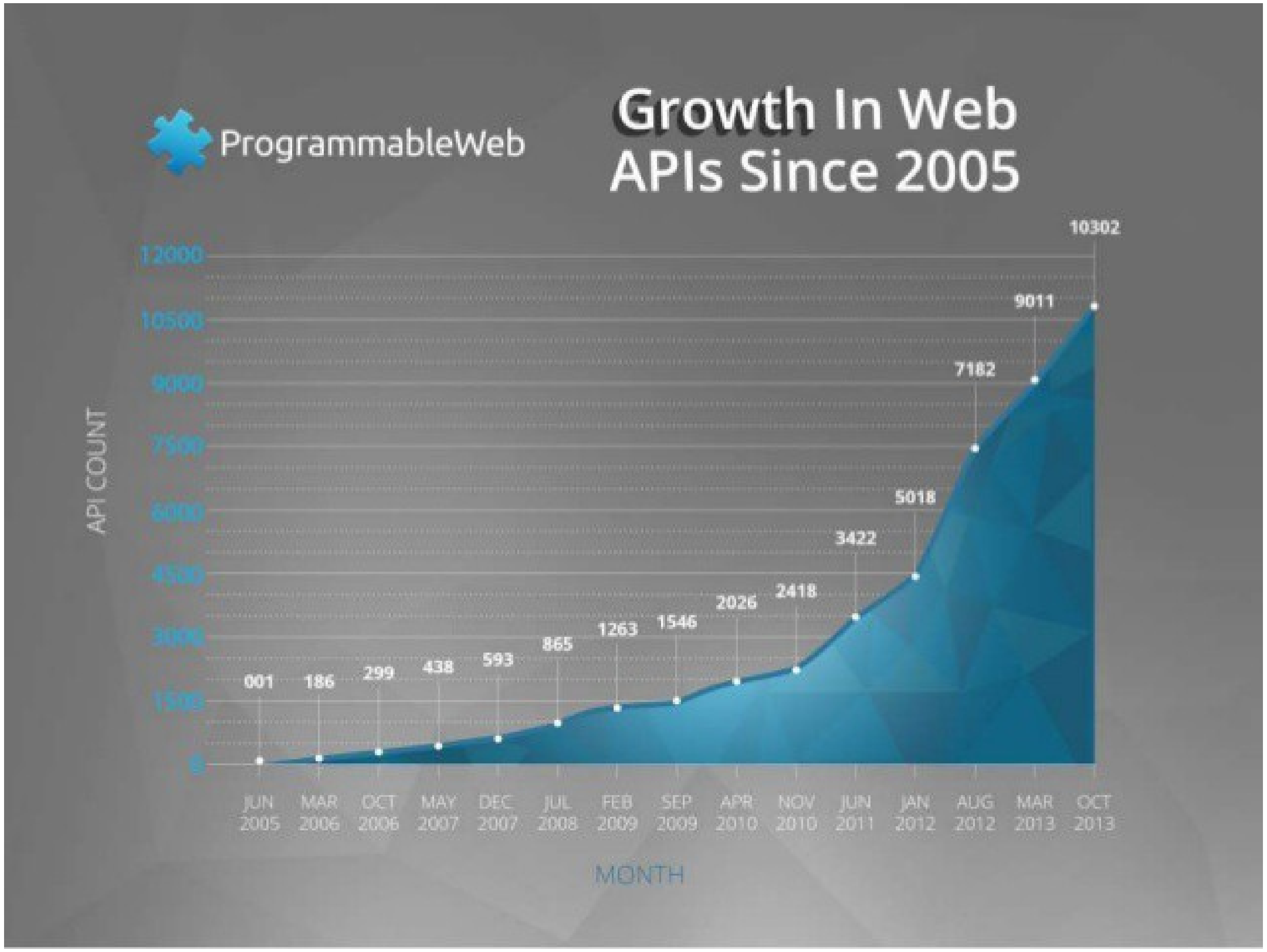
"***$7bn worth of items*** on eBay through APIs"
Mark Carges (Ebay CTO)

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 Web APIs

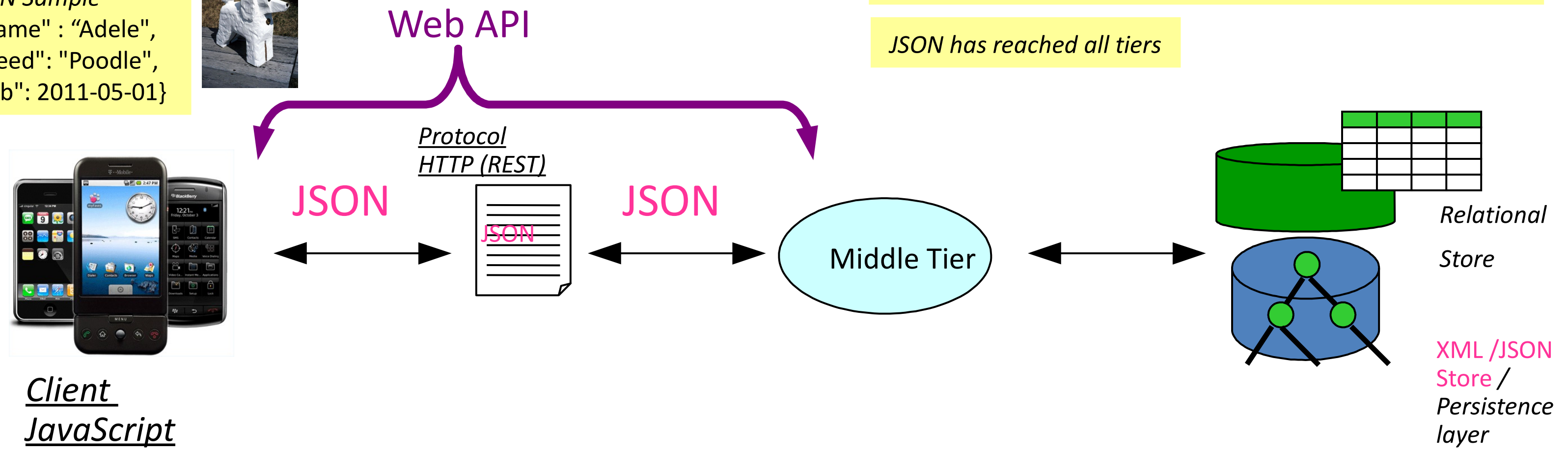


JSON : The Exchange Notation For Mobile Devices

JSON Sample
{"name" : “Adele",
"breed": "Poodle",
“dob": 2011-05-01}

JSON standardized as part of ECMAScript in December 1999

JSON has reached all tiers



- 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:

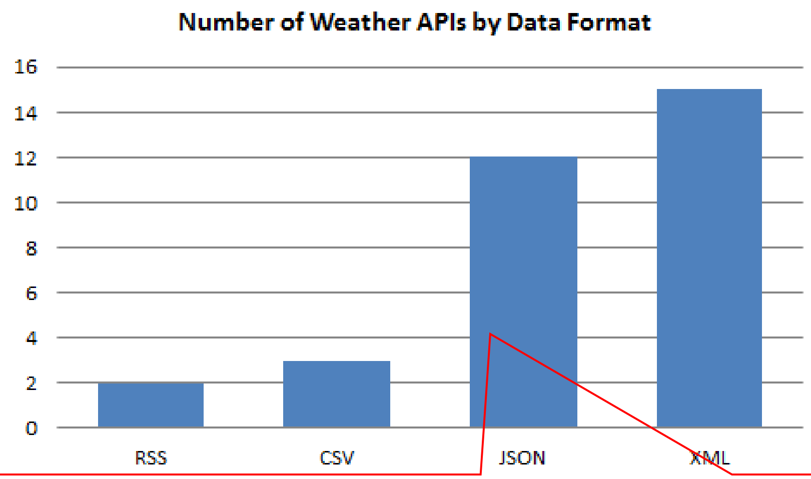
No namespaces

No schemas

No mixed content support Mixed content example: <p>hello <b>Adele</b>how are you</p>

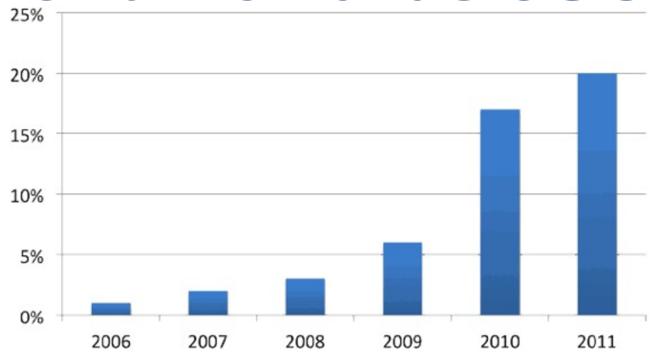
JSON Penetration: Web API Trend Towards JSON

Programmable Web, Jan 2012



26 Weather APIs, 12 Support JSON

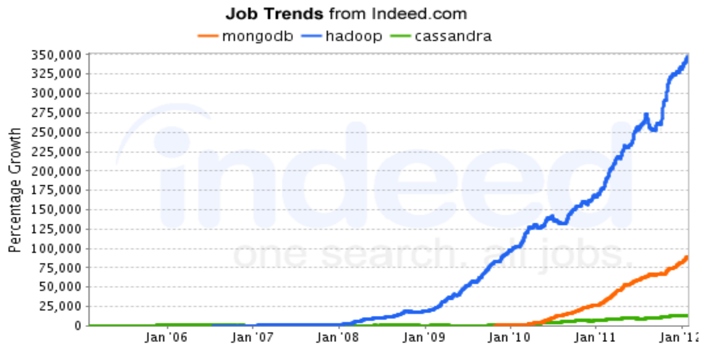
"Weather has always been a popular category with 26 Weather APIs listed in our directory. While XML is still the leading data format used, the trend of JSON becoming the Developer's Choice over the last few years is also reflected in the Weather category"



*Percentage of new APIs with **only** JSON support*

Programmable Web, December 2012

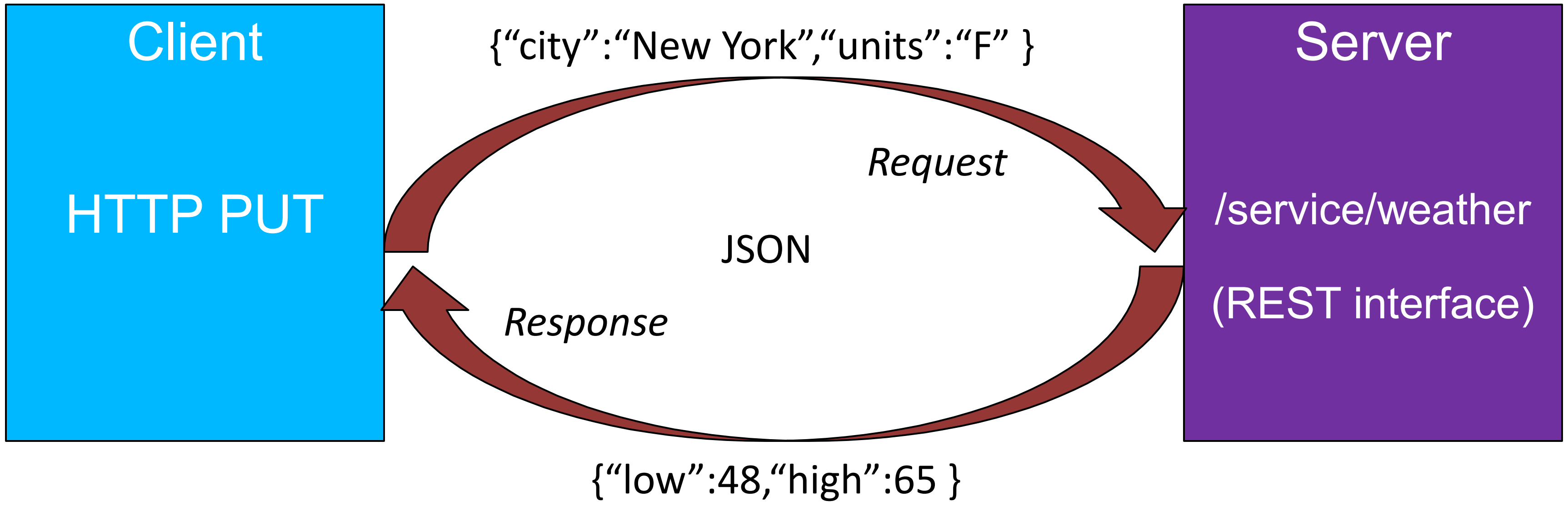
1 in 5 APIs said "Bye XML" - JSON is popular, at least when it comes to API data formats. Of the new APIs we added to our directory, one in four supports *only* JSON (Dec 2012). YouTube API, for example, switched to JSON-only



Programmable Web, January 2012

Library of Congress, Prints and Photos API goes JSON - The Library of Congress, with their Prints and Photographs API has finally taken the REST API plunge. If this government institution can do it, maybe you can convince the head of your IT department that APIs are a good idea too.

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

z/OS Client Web Enablement Toolkit Objectives

- Externalize http and https client functions in an easy-to-use generic fashion for user's in almost any z/OS environment
- Implement a native z/OS JSON parser
- Make sure the web services and the payload processing of the HTTP request body and response body are independent entities
  - HTTP/HTTPS functions separate from JSON support
- Make the interface intuitive for people familiar with programming in this area
- Make the solution z/OS-responsible

Usage & Invocation – z/OS JSON parsing services

- Ideal world:
  - Model JSON parser after its native environment and where the parsing is most intuitive: Java
- Reality:
  - Other non-object-oriented programming languages are less suitable for a direct correlation with a Java programming model
    - For example: no function overloading available in most non-OO languages

Usage & Invocation – z/OS JSON parsing services

- z/OS Client Toolkit execution environment:
  - Any address space
  - Supports both authorized and un-authorized callers
    - Allow supervisor or problem state callers running in any PKM
  - Supports task and SRB mode invokers
  - Supports 31-bit callers now
    - Support 64-bit callers in immediate follow-on and rollback
  - Multi-language support
    - Include files supplied for C/C++, COBOL, PL/I, Assembler

Usage & Invocation – z/OS JSON parsing services

- z/OS Client Toolkit execution environment (continued):
  - Code runs in user's address space
  - Recovery, if any, needs to be provided by caller of service
  - Optimized for best performance
  - Will not offload its parsing processing to a specialty engine

Usage & Invocation – z/OS JSON parsing services (Design points)

- No security controls required
- No throttle control required
- No schema validation

Usage & Invocation – z/OS JSON parsing services

- z/OS Client Toolkit JSON types of services:
  - Init/Term services
  - Parse a JSON String
  - Auto-discovery parse methods
  - Search a JSON string for a specific “name” in a name/value pair
  - JSON text creation methods
    - Serialize text data into JSON syntax

Usage & Invocation – 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

Usage & Invocation – z/OS JSON parsing services

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)

Usage & Invocation – z/OS JSON parsing services

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.

Usage & Invocation – z/OS JSON parsing services

- 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
- Delete methods not provided for the first release

Usage & Invocation – 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 handle of the entry containing the found "name" is returned.
  - Two search types:
    - HWTJ_SEARCHTYPE_GLOBAL
    - HWTJ_SEARCHTYPE_OBJECT

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)

Interactions & Dependencies (JSON parser)

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - COBOL compiler recommended

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

cURL interface

- cURL was ported to z/OS as part of “z/OS Ported Tools” and is available to z/OS UNIX applications
- Libcurl (the API implementation of cURL) is not available in z/OS Ported Tools
- z/OS Ported Tools is open-source and is shipped separately from z/OS
- Libcurl is huge and much bigger than just HTTP/HTTPS communications

Usage & Invocation – z/OS HTTP Services

- Provide similar functionality to existing open-source libcurl interface
  - Interface would be very similar
  - Underlying code to implement this would be z/OS-specific and not ported

Usage & Invocation – HTTP Services

- Target environment:
  - Same as the JSON part of the toolkit
  - Task mode callers
  - Any address space running could avail themselves of these services (non-POSIX environment the focal point of these services)

Miscellaneous z/OS Client Toolkit HTTP client design points

- Protected by standard TCPIP security controls (NetAccess)
- No additional security checks to be performed
- No additional throttle checks provided.

Miscellaneous z/OS Client Toolkit HTTP client design points

Need to support in first release:

- HTTPS connections
- Cookies
- Proxies
- Re-directs

Usage & Invocation – z/OS Client Toolkit HTTP Services

- Simple services will be created to 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

Usage & Invocation – HTTP Services Connections

- Initialize a Connection
  - Allocates storage necessary for connection
  - Returns a connection handle
- Reset a Connection
  - Resets the connection handle to be reused to connect to another server without the need to obtain or release the connection storage
- Terminate a Connection
  - Frees connection storage
  - Invalidates the connection handle

Usage & Invocation – HTTP Services options

- Set HTTP Options
  - Prepares options prior to connecting to the targeted HTTP server
  - Specify the proper connectionHandle
  - Specify one option at a time, with the address and length of the value

Usage & Invocation – HTTP Services – Connection Options

- Various settable options for preparing an HTTP connect for the set command include:
  - 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
  - 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
  - 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.
  - OPT_IPSTACK – Optional value 1 to 8 character z/OS TCP/IP stack to be used by the connection
  - OPT_LOCALIPADDR – Optional outgoing local IP addres
  - OPT_LOCALPORT – Optional outgoing local port
  - OPT_SNDTIMEOUTVAL – Sending timeout value
  - OPT_RCVTIMEOUTVAL – Receiving timeout value
    -

Usage & Invocation – HTTP Services – SSL Options

- SSL support options include:
  - OPT_SSLVERSION – sets the SSL versions to be supported by this HTTP request. More than one version may be selected. The following constants are provided:
  - 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
  - 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_KEYRINGLABEL – represents the RACF key ring label

    For SSLKEYTYPE_KEYRINGNAME – represents the RACF key ring name
  - OPT_SSLKEYPASSWD – optional to set the password of the key database file. Only valid if SSLKEYTYPE_KEYDBFILE is specified. Ignored in all other cases.

Usage & Invocation – HTTP Services – Proxy Options

- Proxy support options include:
  - OPT_PROXY – set the HTTP proxy to user. Specifed the exact same as OPT_URI above.
  - OPT_PROXYPORT – specify the proxy port to connect to. Specified the exact same as OPT_PORT above

Usage & Invocation – HTTP Services – Cookie Options

- Cookie support options include:
  - 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
  - OPT_COOKIE_INPUT_BUFFER – specifies input cookie data store
  - OPT_COOKIE_OUTPUT_BUFFER – specifies output cookie location for a cookietype of COOKIETYPE_PERSIST

Usage & Invocation – 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 issue 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 intact. If a subsequent Connect is issued, the attributes specified on the prior Connect will be used.

Usage & Invocation – 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:

    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

Usage & Invocation – HTTP Services – Request Options

- Various settable options for preparing an HTTP request (continued):
  - OPT_HTTPHEADERS – 4-byte pointer to a linked list of HTTP request headers. These request headers were chained via the HWTIHSLST link list append service.
  - OPT_COOKIE – specific cookies to send apart from normal cookie handling
  - OPT_REQUESTBODY – 4-byte pointer to a request body data (useful on an HTTP POST operation only).
  - OPT_RESPONSEHEADER_EXIT – 4-byte address of the program to receive control once for each response header received by the application.
  - OPT_RESPONSEHEADER_USERDATA – 4-byte address of the optional user data to be passed into the response header exit when it receives control.
  - OPT_RESPONSEBODY_EXIT – 4-byte address of the program to receive control when the response body is received.
  - OPT_RESPONSEBODY_USERDATA – 4-byte address of the optional user data to be passed into the response body exit when it receives control.

Usage & Invocation – HTTP Services

- 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.

Usage & Invocation – 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.

z/OS Client Toolkit HTTP Language Support

- Include files and sample programs provided in:
  - C/C++
  - COBOL
  - PL/I
  - Assembler (Include file only)

Installation – z/OS HTTP Services

- V2R2 – None (in the base)
- V2R1 – Install APAR OA46622 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

Presentation Summary

- Learned what the z/OS Client Web Enablement Toolkit is all about
- Learned the basics about how the APIs are used for both parts of the toolkit
  - z/OS JSON parser
  - z/OS HTTP enabler
- Learned where to find more information

Appendix

Documentation

- V2R2 - *z/OS MVS Callable Services for High-level Languages* (V2R2) – Both JSON parser and HTTP
- V2R1 - Website link to online documentation found in:
  - *z/OS Introduction and Release Guide* (GA32-0887-01) (JSON Parser only)

- In both releases, *z/OS MVS System Codes* – new abend code 04D