Using RESTful web services with z/VSE

Ingo Franzki

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.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
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.

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

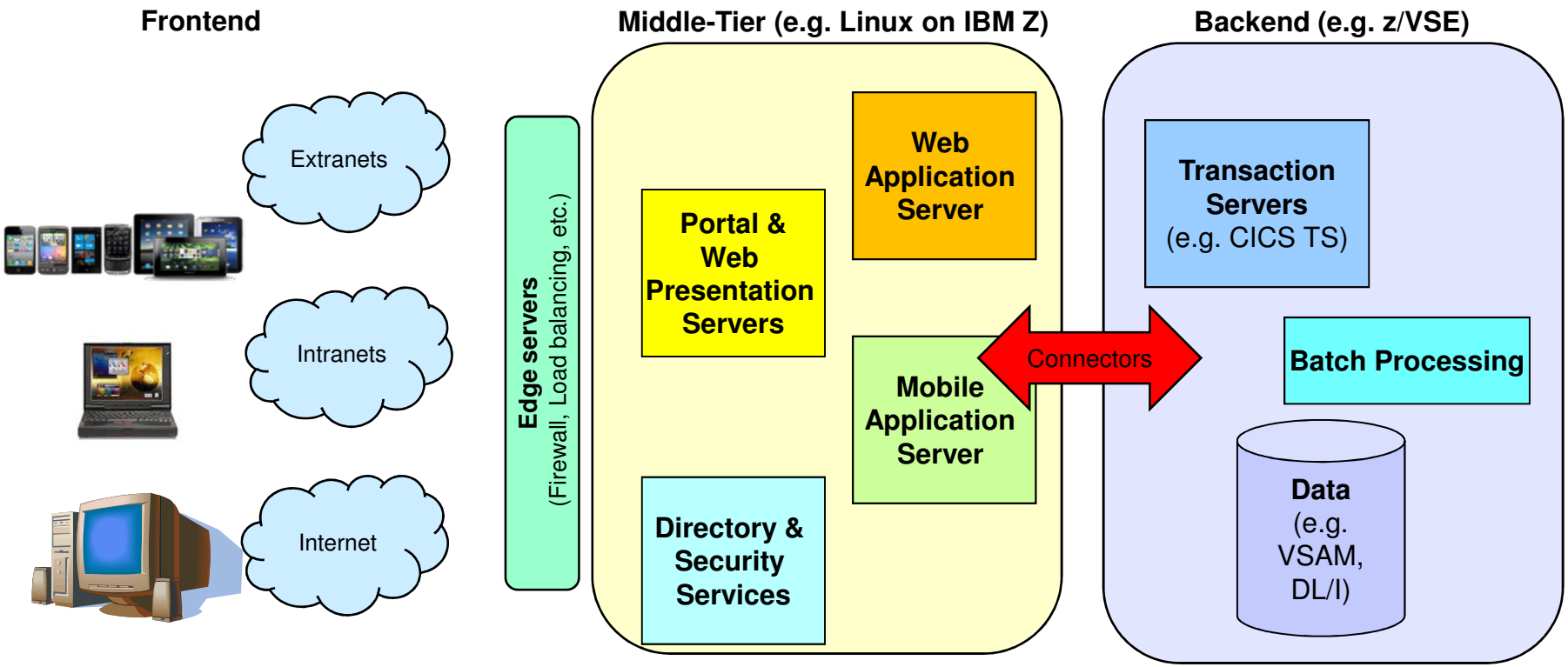
Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's 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.

Notice Regarding Specialty Engines (e.g., zIIPs, zAAPs and IFLs):

- Any information contained in this document regarding Specialty Engines ("SEs") and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at http://www.ibm.com/systems/support/machine_warranties/machine_code/aut.html (“AUT”).
- No other workload processing is authorized for execution on an SE.
- IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.

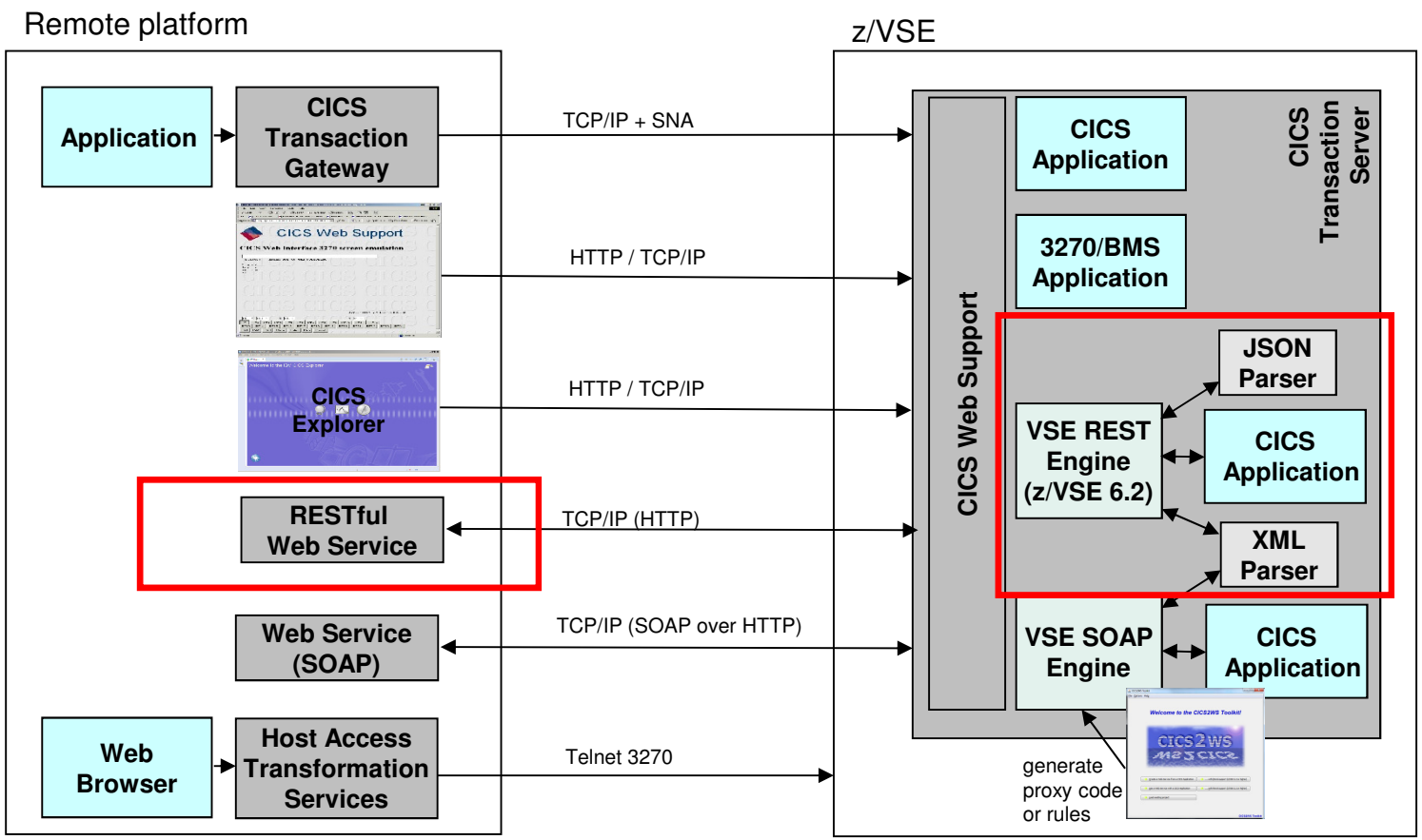
Web- / Mobile-enabling of Applications

→Web-enable z/VSE Applications
→Mobile-enable z/VSE Applications
→Modernize User Interface for applications



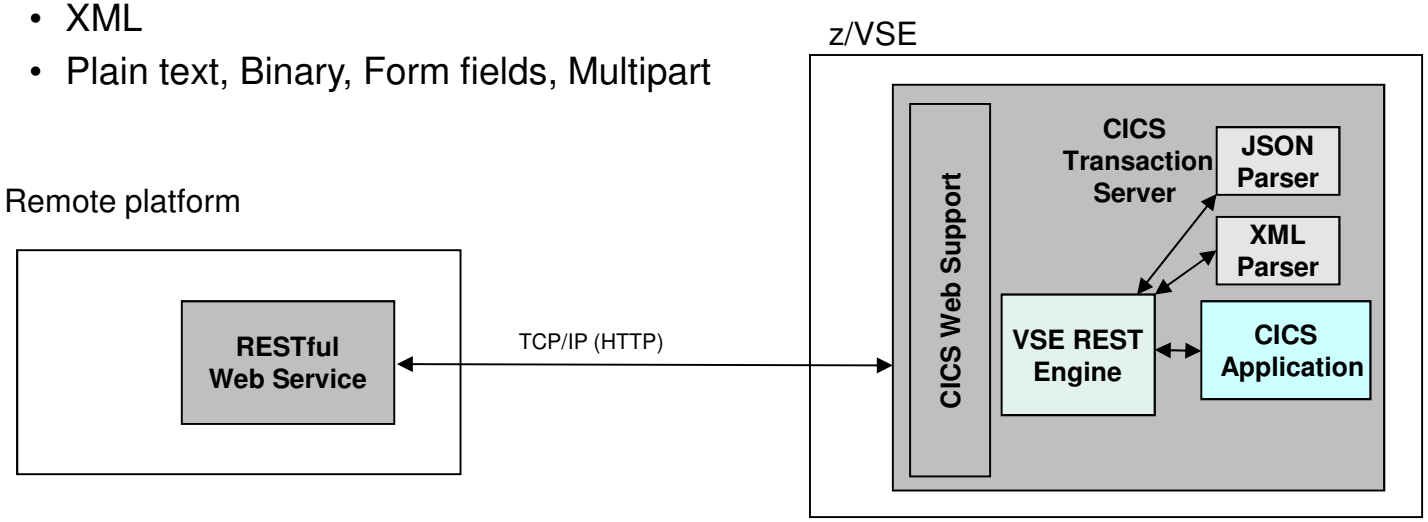
CICS Connectivity

CICS Web Support is the base of CICS connectivity



z/VSE 6.2: RESTful Web Services support

- **Use REST (Representational State Transfer) with CICS applications**
  - **Provide existing CICS applications as RESTful Web Service to the outside world**
    - z/VSE as the REST server
    - Provide an easy to use RESTful API to services for z/VSE services
  - **Use/call external RESTFul Web Services from within z/VSE CICS applications**
    - z/VSE as the REST client
    - Use external RESTful APIs within z/VSE applications
  - **Payload can be:**
    - JSON (JavaScript Object Notation)
    - XML
    - Plain text, Binary, Form fields, Multipart



What is REST (Representational State Transfer)?

- Representational State Transfer (REST) is a **software architecture style consisting of guidelines and best practices** for creating web services
- REST has gained widespread acceptance across the web as a **simpler alternative to SOAP** and WSDL-based web services
- RESTful systems typically communicate over the **Hypertext Transfer Protocol** (**HTTP**)
  - with the same HTTP verbs (GET, POST, PUT, DELETE, and so on) used by web browsers
- The **payload** (message) transported by RESTful web services can be of various types (content types)
  - Commonly used is **JSON** as well as **XML**, but it can also be plain text, or even binary data

What is REST (Representational State Transfer)?

- A RESTful web service typically operates on a certain **'object**' on a server
  - The object is typically addressed through the URI (part of the URL)
    - http://host:port/resource-uri

- Actions on such resources are typically denoted by the HTTP request types:
  - **GET** would typically **read** the resource
  - **PUT** would typically **update/replace** the resource
  - **POST** would typically **create** the resource
  - **DELETE** would typically **delete** the resource

- Additional parameters can be supplied via the URL query string
  - http://host:port/resource-uri?query-string

What is REST (Representational State Transfer)?

- RESTful web services are typically **stateless**
  - Each request from any client contains all the information necessary to service the request
  - The session state is therefore held in the client

- RESTful web services may use **HTTP specific features**
  - **HTTP headers** – to transport additional attributes
  - **Cookies** – to manage state information between requests

→ As denoted by the term 'typically' in above descriptions, there is no hard requirement for any of the described properties

HTTP basics

Request:

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

Request Message Header

A blank line separates header & body

Request Message Body

Response:

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

Response Message Header

A blank line separates header & body

Response Message Body

Taken from: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Example: A RESTful web service

Request a list of books:

Request:
```
GET /api/v1/books
```

Response:
```
{
  meta: {  },
  data: [{
    id: 24,
    title: 'Behavior-Driven Development',
    author: 'Viktor Farcic'
  }, {
    id: 25,
    title: 'Continuous Integration',
    author: 'Viktor Farcic'
  }]
}
```

Create a book:

Request:
```
POST /api/v1/books/id/24
{
  id: 24,
  title: 'Behavior-Driven Development',
  author: 'Viktor Farcic'
}
```

Response:
```
Status: 201 Created
{
  meta: {  },
  data: {
    uri: /api/v1/books/id/24
  }
}
```

Request a single book:

Request:
```
GET /api/v1/books/id/24
```

Response:
```
{
  meta: {  },
  data: {
    id: 24,
    title: 'Behavior-Driven Development',
    author: 'Viktor Farcic'
  }
}
```

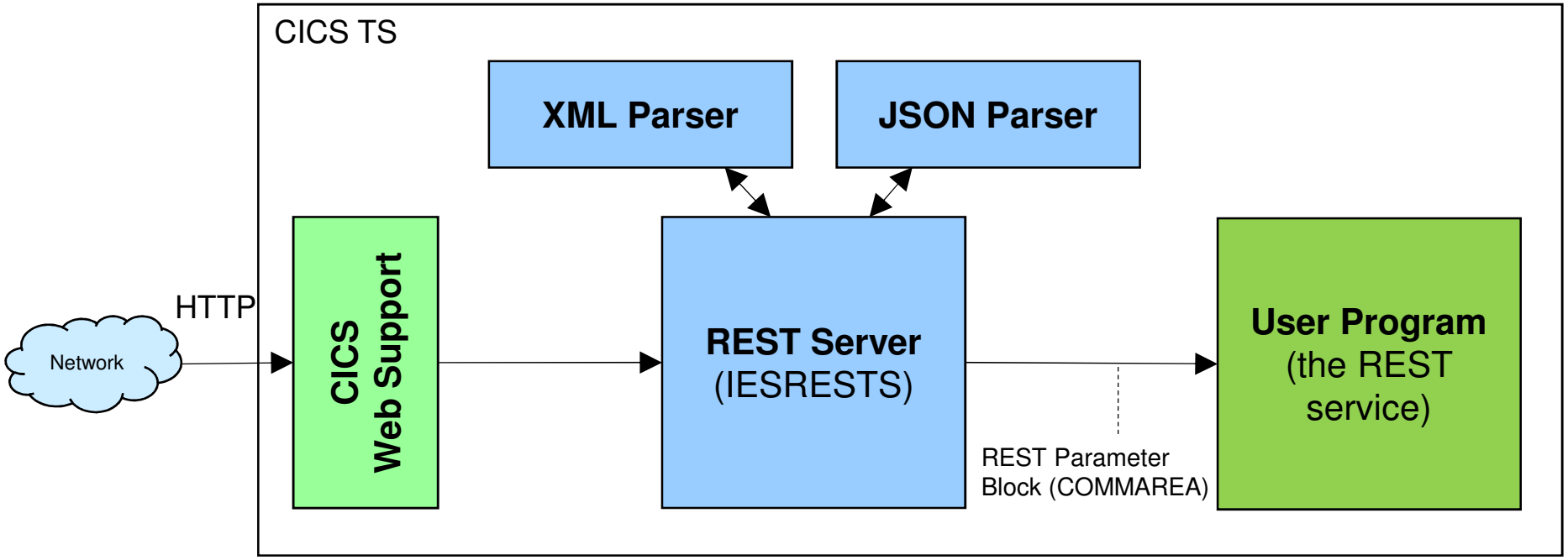
Delete a book:

Request:
```
DELETE /api/v1/books
{
  id: 24,
}
```

Response:
```
Status: 202 Accepted
{
  meta: {  },
  data: {  }
}
```

Example taken from: https://technologyconversations.com/2014/08/12/rest-api-with-json/

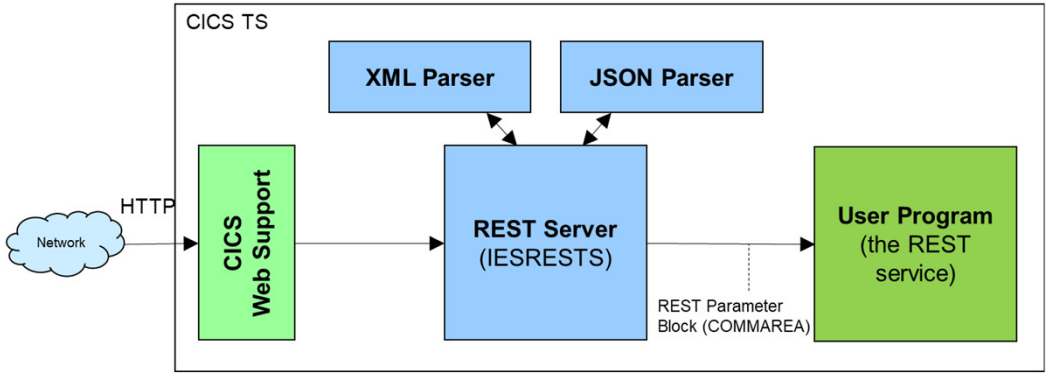
z/VSE 6.2: z/VSE as a REST Server



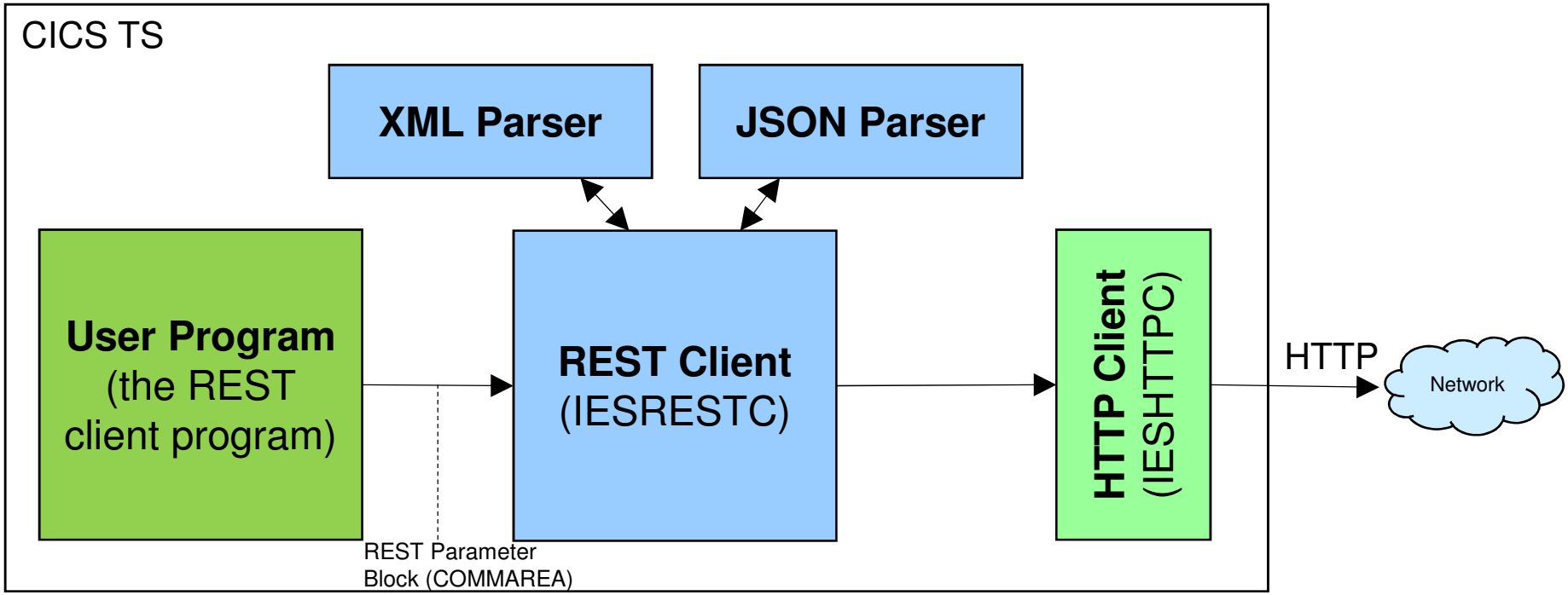
Description of the REST Parameter Block:
https://www.ibm.com/support/knowledgecenter/SSB27H_6.2.0/fa2ws_how_rest_control_blocks_are_used.html

z/VSE 6.2: z/VSE as a REST Server

- **The z/VSE REST-Engine…**
  - Receives the request (via CICS Web Support)
  - Extracts information from the request:
    - User program to call from the URL:
      http://host:port/cics/CWBA/IESRESTS/user-program/resource-uri[?query-string]
    - URL parameters from the query string (if any)
    - HTTP headers
    - Cookies (if any)
    - Request data (if any)
  - Calls the user program
  - Constructs the response:
    - HTTP status code
    - HTTP headers
    - Set-Cookies requests (if any)
    - Response data (if any)
  - Sends the response back to the client



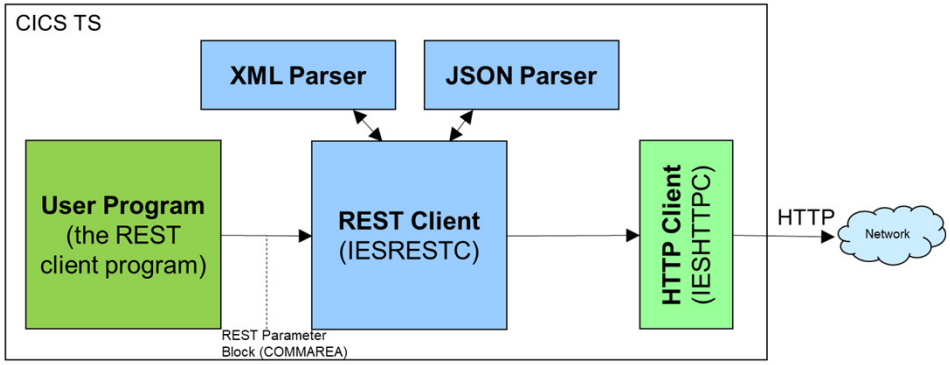
z/VSE 6.2: z/VSE as a REST Client



Description of the REST Parameter Block:
https://www.ibm.com/support/knowledgecenter/SSB27H_6.2.0/fa2ws_how_rest_control_blocks_are_used.html

z/VSE 6.2: z/VSE as a REST Client

- **The z/VSE REST-Engine…**
  - Gets called from the user program
  - Constructs the request
    - Splits the URL into host and port and resource-uri
      http://host:port/resource-uri[?query-string]
    - Adds URL parameters to the query string (if any)
    - HTTP headers
    - Cookies (if any)
    - Request data (if any)
  - Sends the request to the server
  - Receives the response from the server
  - Extracts information from the response:
    - HTTP status code
    - HTTP headers
    - Set-Cookies requests (if any)
    - Response data (if any)
  - Returns back to the user program



The REST parameter block

- **Contains information about the request**
  - Request type (GET, PUT, POST, …)
  - URL
  - Content-Type
  - Data-type (XML, JSON, plain text, binary)
  - URL parameters from query string (http://…..?a=b&c=d)
  - Form fields
  - HTTP headers
  - Cookies
  - Authentication information
  - Response status code

- **Copybooks in PRD1.BASE**:
  - IESRESTH.H LE/C
  - IESRESTL.C COBOL
  - IESJSONP.P PL/1
  - IESRESTA.A HLASM

```
* COMMAREA layout used by the REST Engine to call the user program
* (VSE as REST server) or to get called by the user program
* (VSE as REST client):
*
 01 REST-COMMAREA.
   02 REST-VERSION                PIC 9(9) BINARY.
   02 REST-EBCDIC-CODEPAGE        PIC X(16).
   02 REST-FLAGS                  PIC 9(9) BINARY.
   02 REST-RETCODE                PIC 9(9) BINARY.
   02 REST-PRIVATE                USAGE IS POINTER.
*  Request specific fields:
   02 REST-REQ-ACTION             PIC 9(9) BINARY.
   02 REST-REQ-URL                PIC X(2048).
   02 REST-REQ-CONTENT-TYPE       PIC X(128).
   02 REST-REQ-DATA-TYPE          PIC 9(9) BINARY.
   02 REST-REQ-DATA-PTR           USAGE IS POINTER.
   02 REST-REQ-DATA-LENGTH        PIC 9(9) BINARY.
   02 REST-REQ-URL-PARAMS-TSQ     PIC X(8).
   02 REST-REQ-FORM-FIELDS-TSQ    PIC X(8).
   02 REST-REQ-HTTP-HEADERS-TSQ   PIC X(8).
   02 REST-REQ-COOKIES-TSQ        PIC X(8).
   02 REST-REQ-AUTH-TYPE          PIC 9(9) BINARY.
   02 REST-REQ-AUTH-USER          PIC X(64).
   02 REST-REQ-AUTH-PASSWORD      PIC X(64).
   02 REST-REQ-ACCEPT             PIC X(128).
*  Response specific fields:
   02 REST-RESP-HTTP-STATUS-CODE PIC 9(9) BINARY.
   02 REST-RESP-HTTP-STATUS-TEXT PIC X(128).
   02 REST-RESP-CONTENT-TYPE      PIC X(128).
   02 REST-RESP-DATA-TYPE         PIC 9(9) BINARY.
   02 REST-RESP-DATA-PTR          USAGE IS POINTER.
   02 REST-RESP-DATA-LENGTH       PIC 9(9) BINARY.
   02 REST-RESP-FORM-FIELDS-TSQ   PIC X(8).
   02 REST-RESP-HTTP-HEADERS-TSQ PIC X(8).
   02 REST-RESP-COOKIES-TSQ       PIC X(8).
   02 REST-RESP-LOCATION          PIC X(2048).
```

Handling XML and JSON data

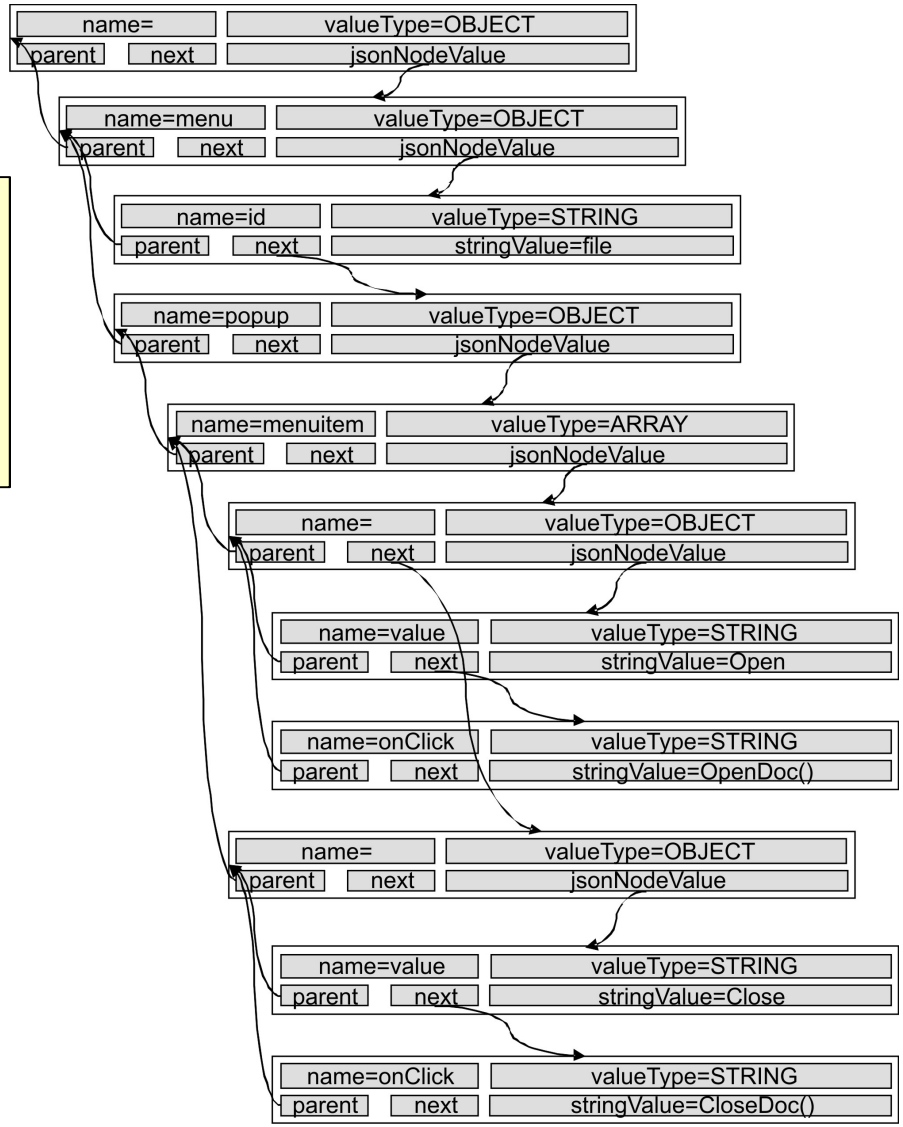
The z/VSE REST Engine automatically translates request and response data

- **XML:** Content-Type: text/xml or application/xml
  - XML data is parsed by the XML parser
  - An XML tree in memory is passed to the user program
- **JSON:** Content-Type: text/json or application/json
  - JSON data is parsed by the JSON parser
  - A JSON tree in memory is passed to the user program
- **URL encoded:** Content-Type: application/x-www-form-urlencoded
  - Form field data is parsed and passed via TS queue entries
- **Plain text:** Content-Type: text/* (other than xml or json)
  - ASCII-EBCDIC converted
- **Binary data:** anything else
- **Multipart data:**
  - Each part is converted individually based on its content type

JSON data

Example:

```
{"menu": {
  "id": "file",
  "popup": {
    "menuitem": [
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```



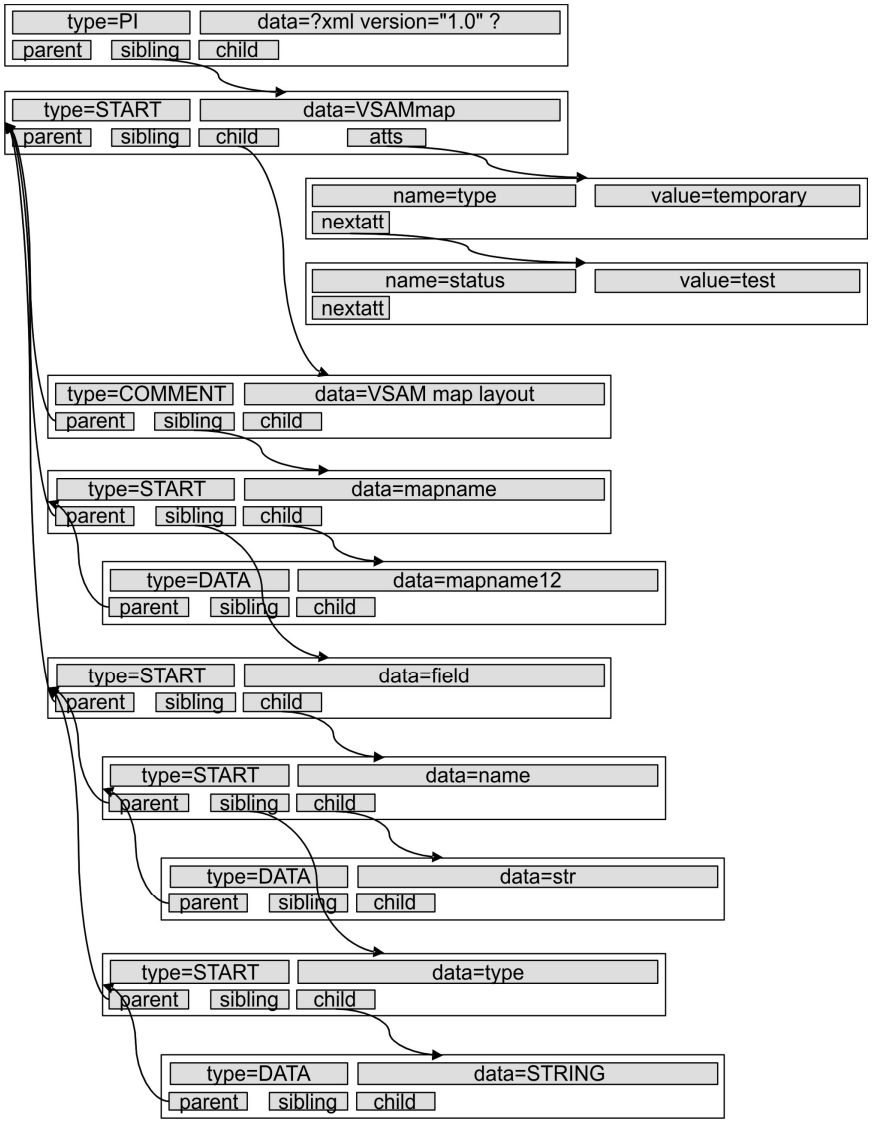
The JSON control blocks are defined in copybooks in PRD1.BASE:

- IESJSONH.H LE/C
- IESJSONC.C COBOL
- IESJSONP.P PL/1
- IESJSONA.A HLASM

XML data

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<VSAMmap type="temporary" status="test">
  <!-- VSAM map layout -->
  <mapname>mapname12</mapname>
  <field>
    <name>str</name>
    <type>STRING</type>
  </field>
  <field>
    <name>sign</name>
    <type>SIGNED</type>
  </field>
</VSAMmap>
```



The XML control blocks are defined in copybooks in PRD1.BASE:

- IESXMLAH.H LE/C
- IESXMLCB.C COBOL
- IESXMLPL.P PL/1
- IESXMLAS.A HLASM

Live Demo

How to provide a RESTful web service with z/VSE

- Based on a COBOL example provided with the z/VSE Connector Client: <conn-client>/samples/rest/COBRESTS.c
- Use the REST testing tool Postman
  - https://www.getpostman.com/

```
* Example COBOL application to illustrate how to use the
* z/VSE REST Engine - VSE as REST server.
*
* This REST service accesses a VSAM file. The name of the
* file is specified in the URL:
*   http://host:port/cics/CWBA/IESRESTS/COBRESTS/<file-name>
*
* Operations:
* - GET:  append URL parameter 'key=<key-value>' to get a
*         particular record:
*                 ../COBRESTS/<file-name>?key=<key-value>
*         Without an URL parameter it starts reading at the
*         first record. Any subsequent GET without an URL
*         parameter will read the following record. The
*         last record read is stored in a session cookie.
*         The record content is returned as JSON data:
*         {
*          "key": "00000002",
*          "data": "Record 2"
*         }
* - POST: Create a new VSAM record. The record content is
*         expected to be in JSON format. The key of the new
*         record is contained in the JSON data as well.
* - PUT:  Updates an existing VSAM record. The record content
*         is expected to be in JSON format. The key of the
*         record to be updated is contained in the JSON data
*         as well.
* - DELETE: Deletes an VSAM record. The record key is specified
*         as URL parameter:
*                  ../COBRESTS/<file-name>?key=<key-value>
```

Questions ?