CICS Web Services, Part 1 - Development

Paul Cooper (pcooper@uk.ibm.com)
Software Engineer; CICS Development, IBM Hursley
November 2021

Agenda

- Overview of CICS® Web services from CICS Transaction Server (TS) V3.1 to CICS TS V5.6
- Learn about the development, deployment, and testing of CICS Web services, for XML and JSON
- Learn about the customisation opportunities
- Things to watch out for....

Useful Resources

- IBM Web Services Redbooks®

Architecture	http://www.redbooks.ibm.com/abstracts/sg245466.html	(2012)
Implementation	http://www.redbooks.ibm.com/abstracts/sg247657.html	(2008)
Performance	http://www.redbooks.ibm.com/abstracts/sg247687.html	(2009)
Security	http://www.redbooks.ibm.com/abstracts/sg247658.html	(2008)
WLM	http://www.redbooks.ibm.com/abstracts/sg247144.html	(2008)
Development	http://www.redbooks.ibm.com/abstracts/sg247126.html	**(2015)**
JSON in CICS	http://www.redbooks.ibm.com/abstracts/sg248161.html	**(2013)**

- Examples (CA1P) https://www.ibm.com/support/pages/node/575887

Web Services, XML and JSON

- Web Services are:
  - Standards based
  - Interoperable
  - Interfaces, for accessing
  - Software Components
  - Over a network

XML: *verbose, but well specified and widely implemented.* Available from CICS TS V3.1.

```
<xml>
  <example>data</example>
</xml>
```

JSON: *efficient wire format and popular with mobile developers, but less well defined.* Available from CICS TS V4.2 (with the *CICS TS Feature Pack for Mobile Extensions*).

```
{
  "Example": "data"
}
```

Application Development 'Bottom Up'

- Start with an existing CICS application
  - COMMAREA described with language structures
    - COBOL, PL/I, C or C++
    - DFHLS2WS generates: **WSDL** and **WSBind file**
- Coverage of data types is not 100%, for example:
  - Pointers
  - Level 66
  - Limited support for PICTURE
- DFHLS2WS (WSDL)
- DFHLS2JS (JSON Schema)
- DFHLS2SC (XML Schema)

```
04    singleChar PICTURE X(1).
04    singleDouble COMP-2 SYNC.
04    singleChar2 PICTURE X(1).
04    singleDouble2 COMPUTATIONAL-2.
04    singleFloat COMP-1.
04    singleFloat2 COMPUTATIONAL-1 SYNC.
04    floatArray COMP-1 OCCURS 5.
04    structure.
07       filler PIC X(1).
07       fieldA PIC X(10).
07       substruc.
09         fieldB PIC S9(10) SIGN LEADING.
09         filedC PIC X(1).
07       fieldD PIC X(1).
04    fieldE PIC S9(10) SIGN LEADING.
04    struc2.
05      struc3.
06        fieldF PIC X(1).
05      fieldG PIC X(1).
04    fieldH PIC X(1).
```

WSDL generated by DFHLS2WS

An example WSDL fragment:

```
<?xml version="1.0" ?>
    <!--
        This document was generated using 'DFHLS2WS' at mapping level '1
    -->
<definitions targetNamespace="http://www.NULLPROG.testSup.com"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:reqns="http://www.NU
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http:/
    <types>
        <xsd:schema attributeFormDefault="qualified"
            elementFormDefault="qualified" targetNamespace="http://www.
            xmlns:tns="http://www.NULLPROG.testSup.Request.com" xmlns:x
            <xsd:annotation>
                <xsd:documentation
                    source="http://www.ibm.com/software/htp/cics/annota
            </xsd:annotation>
            <xsd:annotation>
                <xsd:appinfo source="http://www.ibm.com/software/htp/ci
com.ibm.cics.wsdl.properties.mappingLevel=2</xsd:appinfo>
            </xsd:annotation>
            <xsd:complexType abstract="false" block="#all" final="#all"
                mixed="false" name="ProgramInterface">
                <xsd:sequence>
                    <xsd:element name="singleChar" nillable="false">
                        <xsd:simpleType>
                            <xsd:annotation>
                                <xsd:appinfo source="http://www.ibm.com/
com.ibm.cics.wsdl.properties.charlength=fixed
com.ibm.cics.wsdl.properties.synchronized=false</xsd:appinfo>
                            </xsd:annotation>
                            <xsd:restriction base="xsd:string">
                                <xsd:maxLength value="1" />
                                <xsd:whiteSpace value="preserve" />
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="singleDouble" nillable="false">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:double" />
                        </xsd:simpleType>
```

An example XML schema fragment from within the WSDL:

```
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="singleFloat2" nillable="false">
        <xsd:simpleType>
            <xsd:restriction base="xsd:float" />
        </xsd:simpleType>
    </xsd:element>
    <xsd:element maxOccurs="5" minOccurs="5" name="floatArray"
```

The WSDL contains:

- A programmatic description of the service
- A transport specific binding for the service
- The location of the service (a URI)

Using JCL to invoke DFHLS2WS

- Can be integrated into existing build systems
- Requires Java™ to be installed
- Can be archived for future use

```
//JAVAPROG EXEC DFHLS2WS,
//INPUT.SYSUT1 DD *
MAPPING-LEVEL=4.0
LOGFILE=/u/p9coopr/wsdl/ls2ws.log
WSDL=/u/p9coopr/wsdl/generated.wsdl
PGMNAME=NULLPROG
URI=/testing
PGMINT=COMMAREA
LANG=COBOL
WSBIND=/u/p9coopr/mybindfile.wsbind
PDSLIB=//P9COOPR.COBOL.LIBRARY
REQMEM=TMP01
RESPMEM=TMP01
```

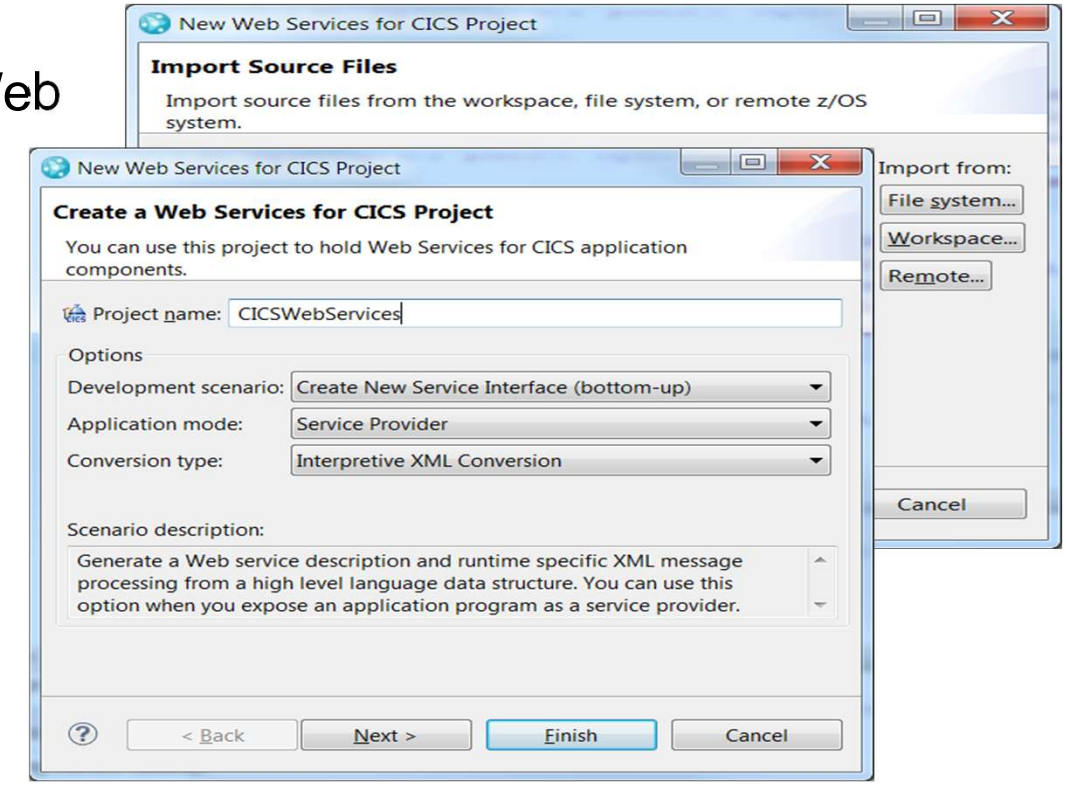
Other 'Bottom Up' Notes

- Applications are:
  - COMM area or Channel based
  - Provider mode (CICS is the server, not the client)
    - For best results use Threadsafe applications
- **IBM Developer for z/OS (IDz)** provides a rich interface with two 'bottom-up' modes:
  - Interpreted (for example, DFHLS2WS etc.)
    - Simple deployment model
  - Compiled
    - Better support for COBOL

- Both involve a WSBind file and have similar performance
- Similar mappings, but **not** identical (so can't be hot-swapped)

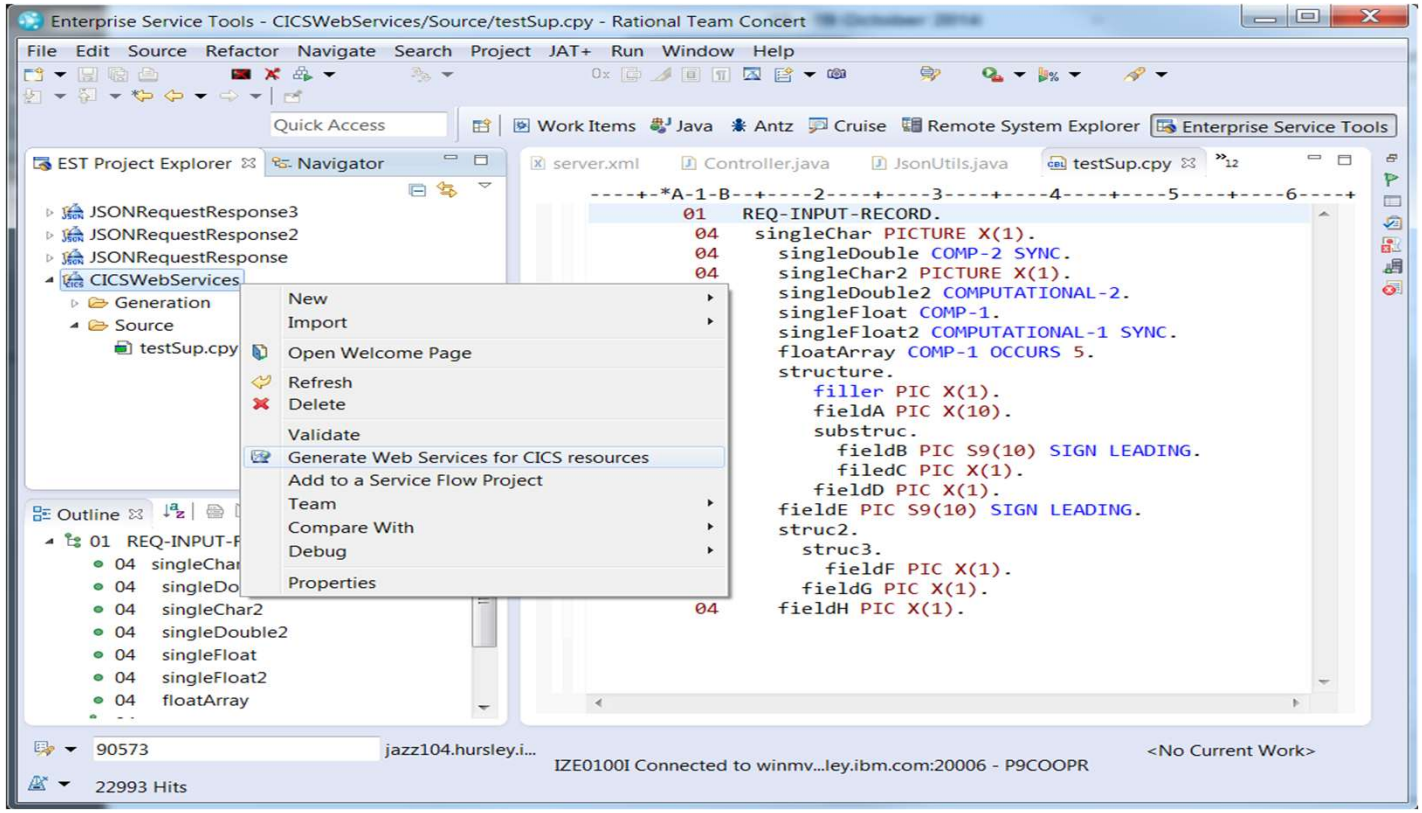
Using IDz 14.2 to expose a Web Service

- Use the wizard to create a new Web Services for CICS Project
- The supported scenarios are:
  - bottom-up
  - top-down
  - meet-in-middle
- Input files can be local or remote
- Generation for SOAP or JSON



Using IDz 14.2 to expose a Web Service (2)

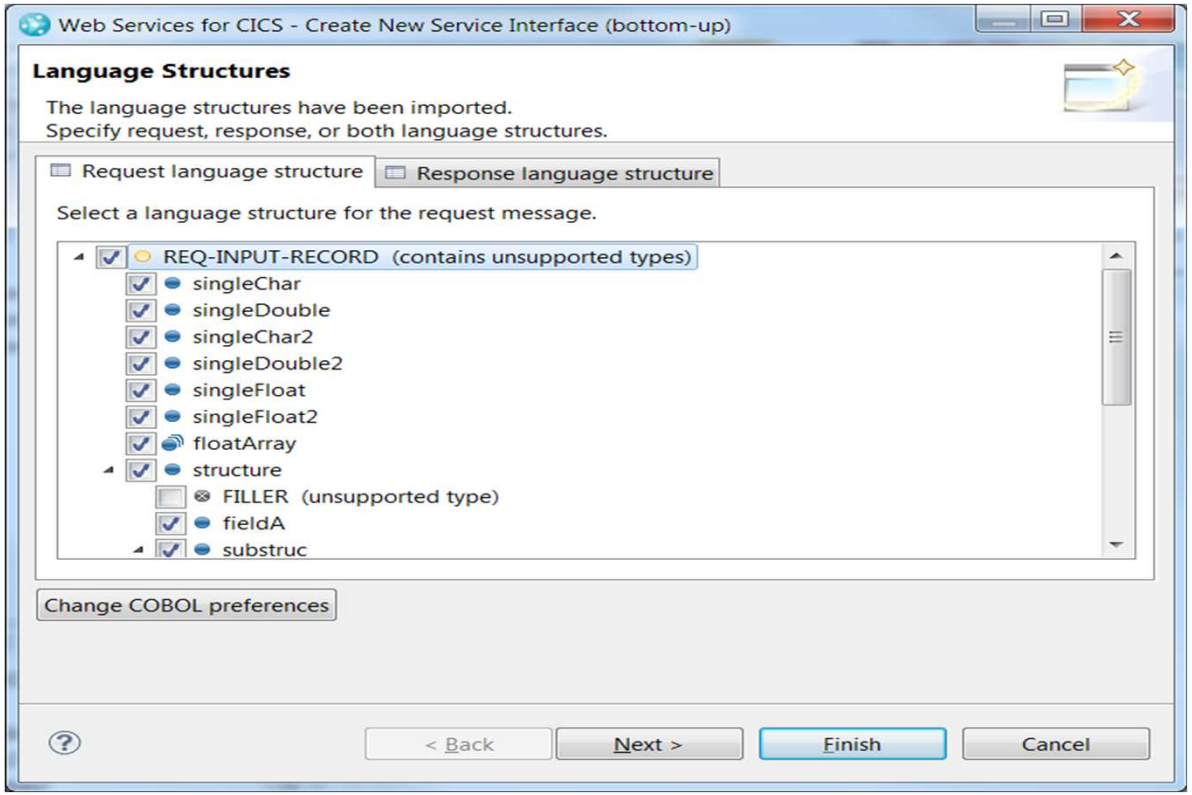
- Generate the Web Services for CICS resources



Using IDz 14.2 to expose a Web Service (3)

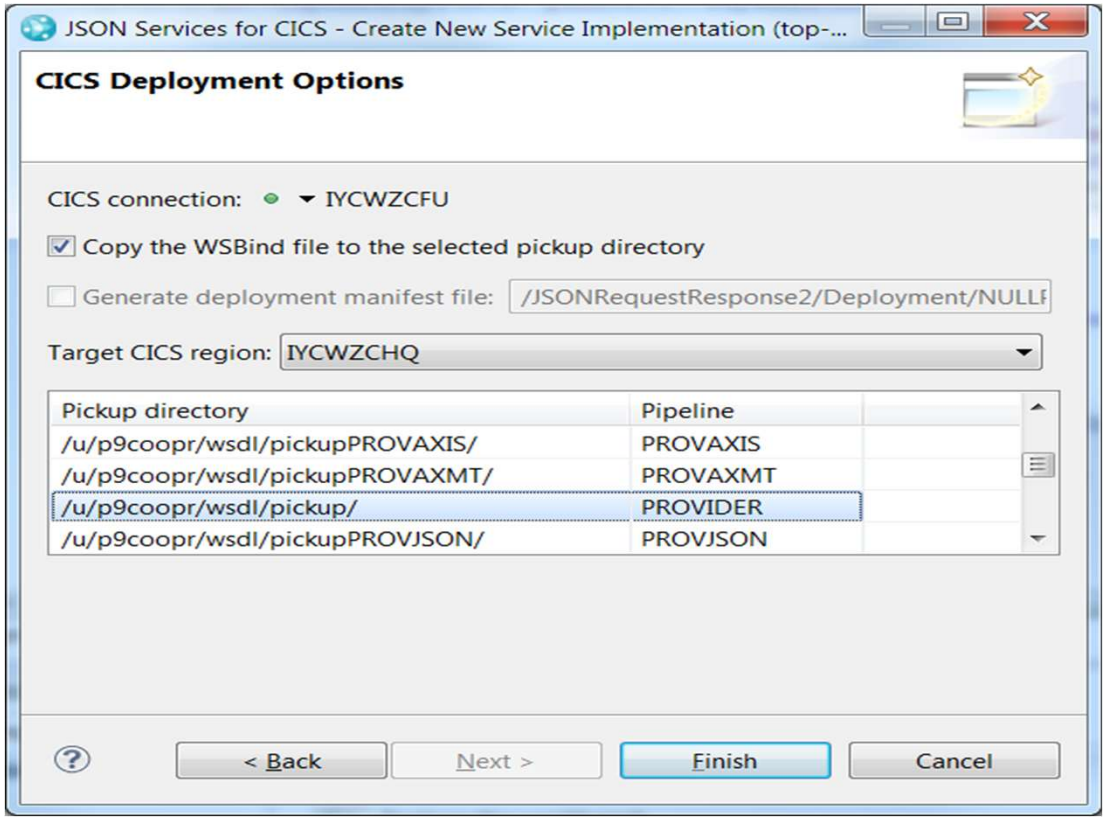
- Select which structures and fields to include in the Service.
- Some fields may be request only whilst others are response only.

Note: using DFHLS2WS from JCL doesn't give you option of omitting fields.



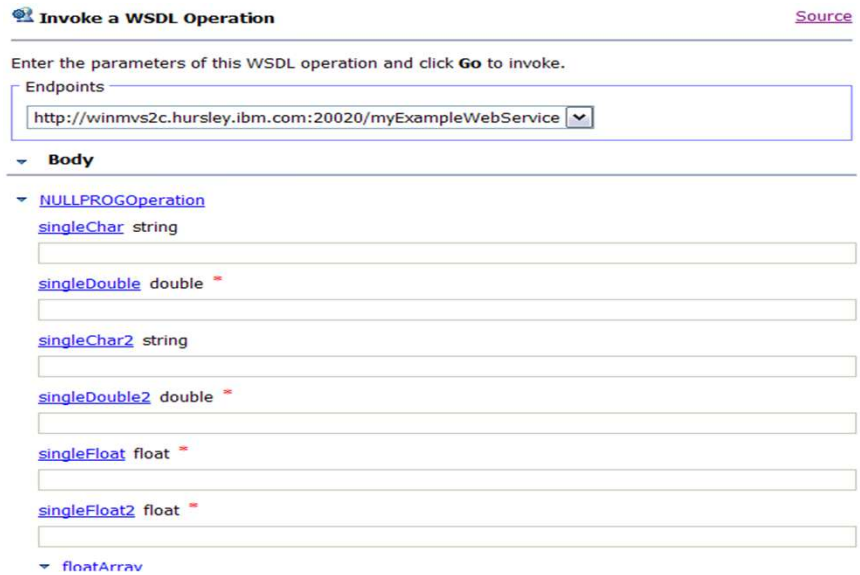
Using IDz 14.2 to expose a Web Service (4)

- And deploy the generated artefacts to CICS.
- You can select a specific CICS region to deploy to, and a specific PIPELINE resource within that region.
- Uses a CICS CMCI connection to update a live CICS region.



Testing the Web Service

- Many tools exist:
  - IDz
  - SoapUI
  - Other vendors

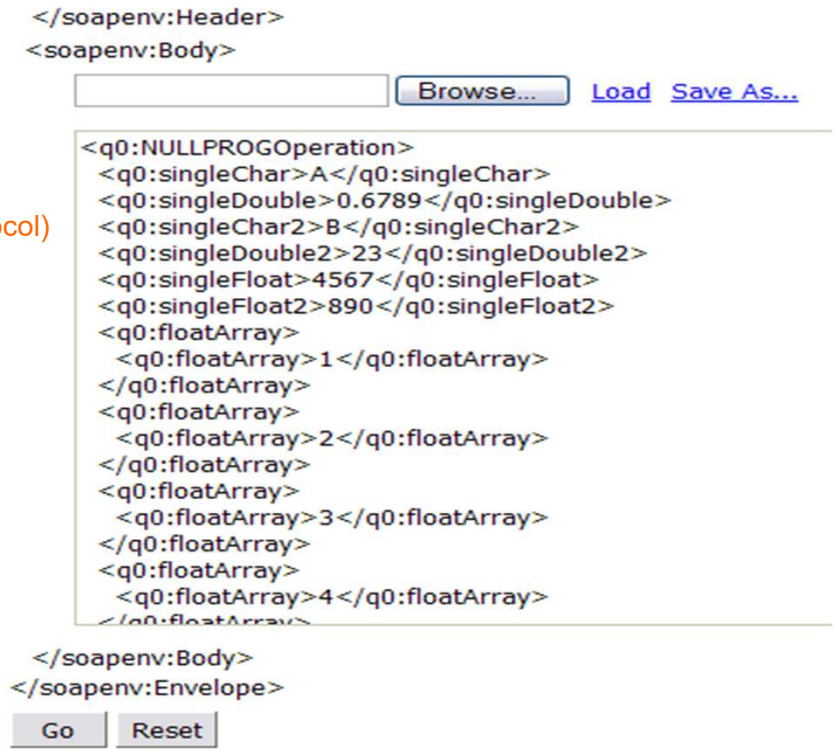


I generated a test program for the Web service in Eclipse with only a few clicks of my mouse! The form opposite allows the values for the input fields to be set. For more details see IBM Docs:

https://www.ibm.com/docs/en/cics-ts/5.6?topic=services-testing-cics-soap-web

Testing the Web Service (2)

- A SOAP message is sent to CICS
- CICS:
  - Receives the SOAP (SOAP is the XML messaging protocol)
  - Converts it to application data
  - Links to the application
  - Converts the response data back into SOAP
  - Sends the SOAP back to the client
- Single day Web service enablement
  .... Hopefully....



JSON Bottom-Up

- Generate JSON Schema
  (IETF draft v4 level)
  - IDz has equivalent JSON Wizards as for SOAP
- JSON WSBind files can be deployed to
  CICS V4.2+
  CTG V9.1+
- z/OS Connect Enterprise Edition
  - More comprehensive support
  - RESTful APIs
- Test with a JSON Client
  - Such as the 'Postman' Chrome App

```
{
    "$schema":"http:\/\/json-schema.org\/draft-04\/schema#",
    "description":"Request schema for the TESTSUP JSON interface",
    "type":"object",
    "properties":{
       "TESTSUPOperation":{
          "type":"object",
          "properties":{
             "req_input_record":{
                "type":"object",
                "properties":{
                   "singleChar":{
                      "type":"string",
                      "maxLength":1
                   },
                   "singleDouble":{
                      "type":"number",
                      "format":"double"
                   },
                   "singleChar2":{
```

Note: *JSON Schema is not widely supported by tooling, so many JSON API designers document their APIs using examples, and developers adopt trial-and-error interactive approaches to client side development*

Should generated WSDL be published?

Perhaps not (*but you can if you want to*)

- Customise it first in order to:
  - Have meaningful field names or alternative data types
  - Your choice of namespaces, operation names, service names, etc.
  - Remove unwanted content (such as annotations or unused fields)
  - Combine multiple generated WSDLs into one composite Service
  - Plan for interface evolution (add version numbers or similar)
  - Generally, to have the external interface you want it to have
- Then reprocess the customised generated WSDL using the top-down tooling
  - This may require a 'wrapper' program to be written (aka meet-in-the-middle)
  - The WSDL becomes the source of the interface (not the original copybooks)

– This approach takes a lot more EFFORT, but you get much better RESULTS!

Application Development 'Top Down'

- Start with WSDL
- Either Provider or Requester mode can be enabled
- DFHWS2LS generates:
  - language structures
  - WSBind file
- Or: DFHJS2LS (JSON Schema)
- Or: DFHSC2LS (XML Schema)

```
<?xml version="1.0"?>
<definitions name="lengthTests"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:s0="http://test.org/"
             targetNamespace="http://test.org/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <s:schema targetNamespace="http://test.org/"
        xmlns:s0="http://test.org/"
        xmlns:s="http://www.w3.org/2001/XMLSchema">
      <s:complexType name="customerType">
        <s:sequence>
          <s:element name="accountNumber"    type="s:int" />
          <s:element name="name"             type="s:stringy"/>
          <s:any/>
        </s:sequence>
      </s:complexType>
    </s:schema>
  </types>
```

Good editors can validate your WSDL. I validated my WSDL in Eclipse and it highlighted a problem that must be fixed. It's well worth validating your WSDL prior to processing it with DFHWS2LS.

Mapping Levels

- The Mapping Level is the version of the programmatic interface shared between the application and CICS.
  - A form of Version Control
  - Allows old WSBind files to be regenerated without requiring application changes
- New capabilities are implemented at new mapping levels. For new applications, use the most recent mapping level available.

Mapping Level 1.0 – Original CICS TS V3.1 capabilities
Mapping Levels 1.1/1.2 – Enhancements added by APARs PK15904 and PK23547
Mapping Level 2.0 – Original CICS TS V3.2 capabilities
Mapping Levels 2.1/2.2 – Enhancements added by APARs PK59794 and PK69738
Mapping Level 3.0 – CICS TS V4.1 capabilities
Mapping Level 4.0 – CICS TS V5.2 capabilities (UTF-16, COBOL 'DEPENDING ON')
Mapping Levels 4.1/4.2/4.3 - Enhancements added by APARs PI67641, PI86039 and PI88519

Application Development Challenges

- Complex WSDL leads to very complex COBOL, so take care!
  - Just because you have a WSDL description of a service doesn't mean that you can easily call or implement it in COBOL. Keep it simple!
- An application program must be written to call or implement the service
  - Using the generated language structures
    - Comments are generated into the language structures
  - IDz will generate some template code to get you started
    - Interpreted mode only (there is no compiled mode 'top down')
  - Many options exist in the tools to tweak the data mapping

Performance Considerations

- Complexity will cost you!
  - Sending large volumes of data is expensive
    (CICS is optimised for 32K even though you can send much more)
  - CPU costs increase with the number of XML tags used
    (regardless of the length of data sent)

Also, see Redpaper 'SOAP Message Size Performance Considerations' which discusses a real CICS Web services application with real performance characteristics
http://www.redbooks.ibm.com/abstracts/redp4344.html

Requester Mode (top down, SOAP only)

- EXEC CICS INVOKE SERVICE
  - SOAP only
- Select the WSDL Operation(s) to enable
  - Avoid generating meta-data and language structures for unused operations
- Time-out:
  - Set on the PIPELINE (from TS 3.2)
    - Or per request using the DFHWS-RESPWAIT container
- URI comes from a client mode URIMAP
  - TS V4.1 or above
  - Can be supplied in a handler program in the Pipeline, or from the application

Requester Mode (top down) (2)

- Cipher configuration for TLS
  - Set in the URIMAP (TS V4.1+)
  - CICS looks for a 'client' mode URIMAP that matches the URI
    - If found, that URIMAP is used (including SSL parameters)
    - Otherwise the default certificate for the region is used
- JSON Alternative
  - Use z/OS Connect Enterprise Edition
  - Use EXEC CICS TRANSFORM for programmatic conversions
    - And the EXEC CICS WEB api

Provider Mode (top down)

- Similar to the bottom-up approach, except that a new application has to be written to implement the service
- Takes as input a Channel with Containers
  - Various CICS generated Containers can be used by the application
    - For example, DFHWS-OPERATION (holds name of WSDL Operation invoked)
- EXEC CICS SOAPFAULT api may be used to create application specific FAULT responses
  - For SOAP aware applications
  - ABENDs are turned into SOAP Fault messages by CICS
    - (Or a JSON equivalent)

Request/Response & RESTful (top down)

- Request-Response
  - A typical CICS program
  - Receives a request, sends a Response
  - SOAP or JSON

- RESTful APIs
  - JSON only
  - URIs address state instances (for example: /accounts/0001, /accounts/0002, etc.)
  - Instances can be created (POST), queried (GET), changed (PUT), or deleted (DELETE)
  - Use z/OS Connect Enterprise Edition

XML aware applications

- You can write CICS applications that work directly with the XML
  - Custom application handler program (provider mode)
  - EXEC linkable pipeline program – DFHPIRT (requester mode)
  - XML-ONLY in DFHWS2LS
    - generate a WSBind file that tells CICS not to do any conversions
    - shared deployment model, support for validation, monitoring, INVOKE WEBSERVICE, etc..
    - the application populates/parses the DFHWS-BODY container.
  - XML parsing / generation can be done using Enterprise COBOL
    - Or using converter programs generated in IDz
    - Or using EXEC CICS TRANSFORM
    - Or in Java
    - Or using vendor products, etc.

WSDL unsupported by DFHWS2LS

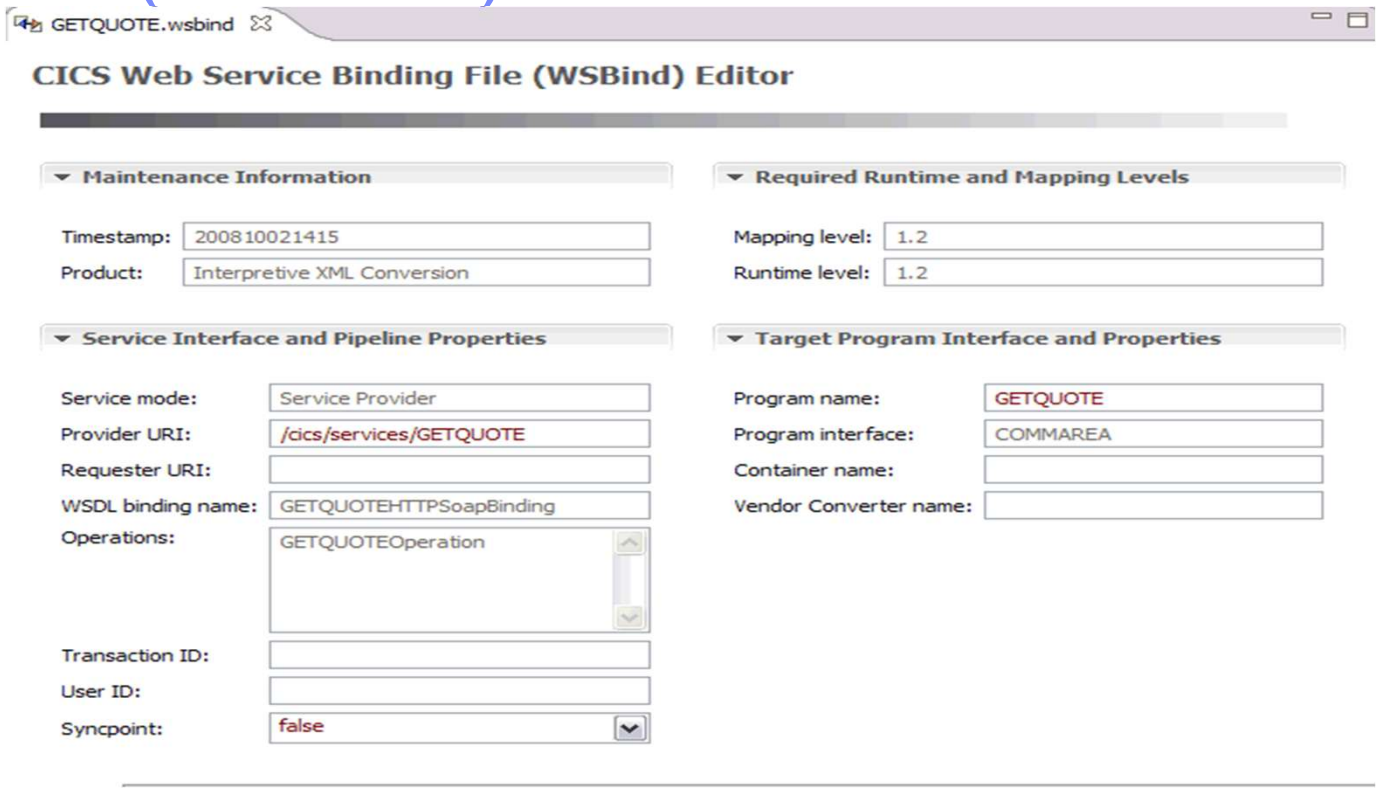
- Some WSDL isn't supported by DFHWS2LS
  - Validate the WSDL and try again with the best mapping level available
  - Unsupported constructs include:
    - 'SOAP encoding'; 'minOccurs' and 'maxOccurs' on xsd:sequences
    - Recursion
- Other options:
  - Work directly with the XML
  - Write JAX-WS Java applications, and host them in WebSphere Liberty Profile
    https://www.ibm.com/docs/en/was-liberty/base?topic=liberty-deploying-jax-ws-applications
  - Use a different transformation technology such as an IBM WebSphere DataPower appliance

Web Service Binding Files

- The WSBind file contains:
  - meta-data to facilitate the runtime data conversions
  - deployment information
    - The URI of the service
    - And in provider mode:
      - The name of the PROGRAM to link to
      - (Optionally) the TRANSID to run under
      - (Optionally) the default Userid to run under
- Can be viewed, edited and deployed from within ID/z

WSBind file Editor (IDz 14.2)

- Useful if you want to change the deployment characteristics without regenerating the entire WSBind file:
  - URI
  - Transaction ID
  - User ID
  - PROGRAM Name
  - SYNC-ON-RETURN
- See also SupportPac CS04



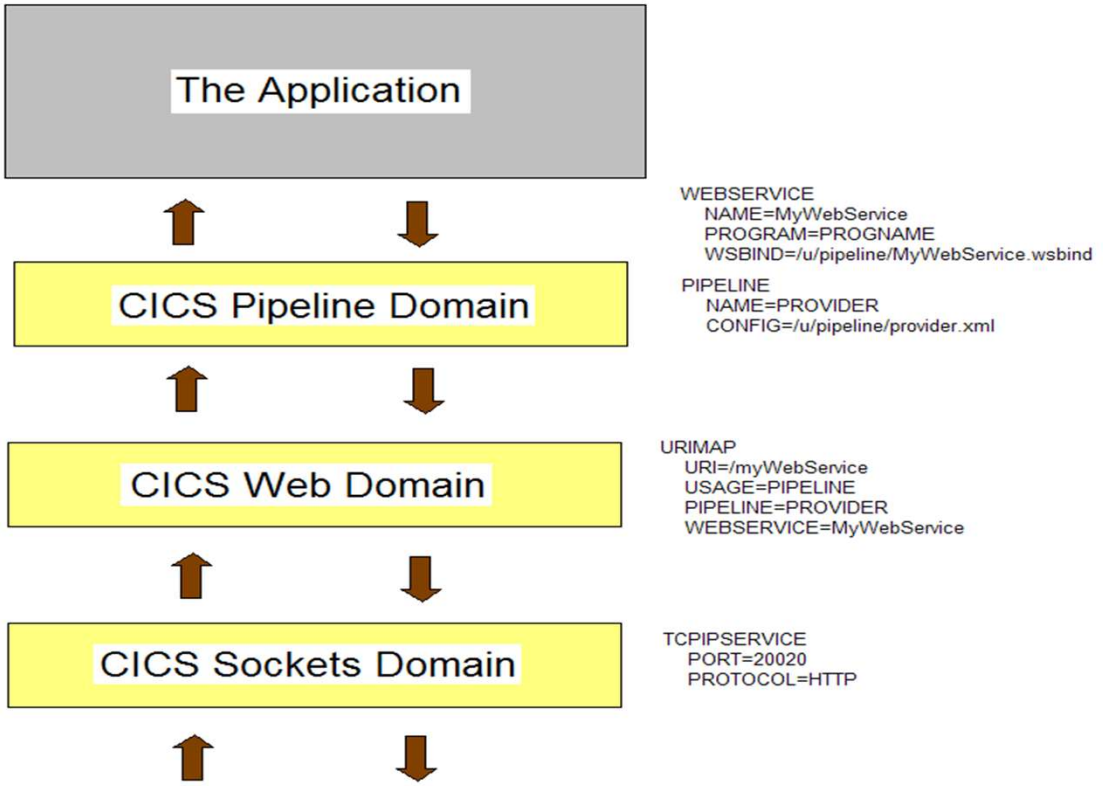
CICS resources involved (provider mode)

WEBSERVICE
Identifies application specific processing

PIPELINE
Identifies shared qualities of service

URIMAP
Identifies the type of processing required

TCPIPSERVICE
The listener process (if HTTP is used)



Web Service Deployment

- Each Web Service will normally need a WEBSERVICE resource and a URIMAP resource
  - PIPELINE 'SCAN' will install a set of WEBSERVICE resources from WSBind files
    - Place the WSBind files in the WSDIR directory of the Pipeline in z/FS
    - A WEBSERVICE resource is created in CICS for each WSBind file
    - If URIs are specified in the WSBind files then URIMAP resources are installed too (provider mode only)
  - Traditional definitions through RDO, BAS are also supported
  - Can be deployed from a CICS Bundle resource (TS V5.2+)

Summary

- CICS is a first class Web services end-point with a highly customisable technology stack.
- Web services enable interoperability with products and services from many different vendors.
- You can use industry standard and best-of-breed tools to interact with CICS & there's an ecosystem of associated products that add value.

Questions and Answers