CICS Web Services as a Provider and Requestor

Ezriel Gross
Circle Software Incorporated

March 13th, 2013 (Thu)
3:00pm – 4:00pm
Session 14827
Agenda

- Introduction to web services in general, and in CICS
- Four methods for creating a web service provider in CICS:
  1. CICS web services assistant
  2. Rational Developer for System z (RDz) with interpretive runtime XML conversion
  3. RDz, with compiled runtime XML conversion
  4. RDz Service Flow Modeler (SFM)
- Two methods for creating a web service requester in CICS:
  1. CICS web services assistant
  2. RDz
- Diagnosing web services in CICS
Terms

Web service
• A software system designed to support interoperable machine-to-machine interaction over a network
• It has an interface described in a machine-processable format (specifically WSDL)
• Other systems interact with [it...] using SOAP messages, typically conveyed using HTTP

WSDL
• [Web Service Description Language is an XML vocabulary that] describes [...] the messages that are exchanged between the requester and provider

SOAP
• [A ...] framework for packaging and exchanging XML messages

Source: Web Services Architecture
http://www.w3.org/TR/ws-arch/
Basic concept

Web service requester

SOAP

Request

Network
(Often, the Internet, or a corporate IP network)

Response

Web service provider

SOAP
Example SOAP request

```xml
  <soapenv:Body>
    <PAYBUSOperation>
      <ws_payroll_data>
        <ws_request>DISP</ws_request>
        <ws_key>
          <ws_department>1</ws_department>
          <ws_employee_no>00001</ws_employee_no>
        </ws_key>
        </ws_payroll_data>
      ...some markup omitted for brevity...
    </PAYBUS1Operation>
  </soapenv:Body>
</soapenv:Envelope>
```

XML defined by the SOAP standard

Web service-specific XML (contents of the SOAP Body) is described in a WSDL file

In plain English: Please “display” payroll data for employee number 1 in department 1
Example SOAP response

```xml
<soapenv:Envelope
   xmlns="http://www.PAYBUS.PAYCOM1.Request.com"
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <PAYBUSOperationResponse>
      <ws_payroll_data>
        <ws_request>DISP</ws_request>
        <ws_key>
          <ws_department>1</ws_department>
          <ws_employee_no>00001</ws_employee_no>
        </ws_key>
        <ws_name>CIRCLE COMPUTER 1</ws_name>
        <ws_addr1>65 WILLOWBROOK BLVD</ws_addr1>
        <ws_addr2>4TH FLOOR</ws_addr2>
        <ws_addr3>WAYNE, NJ 07470</ws_addr3>
        <ws_phone_no>890-9331</ws_phone_no>
        <ws_timestamp/>
        <ws_salary>50000.00</ws_salary>
        <ws_start_date>12312008</ws_start_date>
        <wsRemarks>CIRCLE IS MAGIC</wsRemarks>
      </ws_payroll_data>
    </PAYBUSOperationResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Response details
Web Service Description Language (WSDL) file

- WSDL 1.1 (see below) or 2.0: generated by CICS web services assistant or RDz (if you don't have one)
- Describes the request/response message XML (schema); groups messages into operations on an abstract port; binds the operations to a message transport; specifies the web service address

```xml
<definitions ... >
  <types>
    <xsd:schema ... > ... </xsd:schema>
    <xsd:schema ... > ... </xsd:schema>
  </types>
  <message name="PAYBUSOperationResponse">
    <part element="resns:PAYBUSOperationResponse" name="ResponsePart"/>
  </message>
  <message name="PAYBUSOperationRequest">
    <part element="reqns:PAYBUSOperation" name="RequestPart"/>
  </message>
</definitions>
```
WSDL 1.1 file, continued

```xml
<portType name="PAYBUSPort">
  <operation name="PAYBUSOperation">
    <input message="tns:PAYBUSOperationRequest" name="PAYBUSOperationRequest"/>
    <output message="tns:PAYBUSOperationResponse" name="PAYBUSOperationResponse"/>
  </operation>
</portType>

<binding name="PAYBUSHTTPSoapBinding" type="tns:PAYBUSPort">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="PAYBUSOperation">
    <soap:operation soapAction="" style="document"/>
    <input name="PAYBUSOperationRequest">
      <soap:body parts="RequestPart" use="literal"/>
    </input>
    <output name="PAYBUSOperationResponse">
      <soap:body parts="ResponsePart" use="literal"/>
    </output>
  </operation>
</binding>

<service name="PAYBUSService">
  <port binding="tns:PAYBUSHTTPSoapBinding" name="PAYBUSPort">
    <soap:address location="http://my-server:my-port/paybus1"/>
  </port>
</service>
</definitions>
```
Problem

Web service provider that your CICS application wants to use

SOAP

Internet or intranet (IP network)

Data area

Your CICS application

Web service requester that wants to use your CICS application as a web service provider

SOAP

Traditionally, CICS programs communicate via data areas (COMMAREA or containers); web services send SOAP messages (XML) over a network.
Solution

CICS manages IP and HTTP

A pipeline of programs unwraps data from SOAP XML into a data area, and vice versa

Your program can continue to work with data areas
CICS as a web service provider

Tasks

- Listens for requests
- CICS web support
- Sends responses

Pipeline

- Message handler(s)
- Application handler
  - Your program

Resources

- TCPIPSERVICE
- URIMAP
  - dynamic
- PIPELINE
  - dynamic
- WEBSERVICE

- Pipeline configuration
- Pickup directory
- wsbind
- WSDL

z/OS UNIX files

Soap
CICS as a web service requester

Tasks

- Your program
  - EXEC CICS
  - INVOKE
  - WEBSERVICE
- Pipeline
  - Message handler(s)
- Sends requests
  - CICS web support
  - Listens for responses

Resources

- INVOKE WEBSERVICE can optionally refer to a
  - URIMAP
  - PIPELINE
  - WSDL
- Pipeline configuration
- Pickup directory
- wsbind
- z/OS UNIX files
- SOAP

The task that invoked your program
CICS resources

- You must manually create:
  - **Provider only:**
    - **TCPIPSERVICE:** Specifies which port to listen to for requests. (This assumes HTTP message transport. For WebSphere MQ, you would create an MQCONN.)
  - **PIPELINE:** Points to a pipeline configuration file, which specifies the sequence of handler programs in the pipeline.

- CICS dynamically creates when PIPELINE is installed (or when you run the PIPELINE SCAN command):
  - **Provider only:**
    - **URIMAP:** Specifies which pipeline and web service to use for this request. (For a requester, the INVOKE (WEB)SERVICE can optionally refer to a URIMAP for the provider address.)
  - **WEBSERVICE:** Points to a WSDL file and a wsbind file.
Pipeline configuration file

• Defines the handlers that constitute the pipeline (in these examples, the single handler wraps/unwraps the contents of the SOAP message body in the SOAP envelope)

• If you do not require special processing, you can use these IBM-supplied sample files unchanged:

```xml
<provider_pipeline ...
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>

<requester_pipeline ...
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
    <service>
      <service_handler_list>
        <cics_soap_1.1_handler/>
      </service_handler_list>
    </service>
  </service>
</requester_pipeline>
```

Also known as a “wrapper” program. Extracts data from XML, calls your CICS application program, converts returned data back into XML.
Web service binding (wsbind) file

- Generated by CICS web services assistant or RDz
- Proprietary to CICS web services
- Contains web service-specific information, such as how to map between the fields in a COMMAREA or container and the XML in a SOAP message body
- Enables you to use the CICS-supplied application handler (DFHPITP) for different web services
wsbind file: pickup and shelf directories

- When you install the PIPELINE resource, or when you issue a PIPELINE SCAN command, CICS copies the wsbind file from the pickup directory to the shelf directory.
- At runtime, CICS refers to the copy in the shelf directory.
Creating a web service provider in CICS

Web service requester → Network → SOAP request → Web service provider → SOAP response → Your CICS program
Methods for creating a web service provider in CICS

1. **CICS web services assistant** (batch utilities supplied with CICS) from a copybook, using the DFHLS2WS batch utility (generates a WSDL file and a wsbind file)

2. **Rational Developer for System z (RDz)** from a copybook (using a wizard), with *interpretive* runtime XML conversion (as per DFHLS2WS, above)

3. **RDz** as above, but with *compiled* runtime XML conversion (in addition to WSDL and wsbind files, also generates a bespoke COBOL program to convert XML)

4. **RDz Service Flow Modeler** from a recording of an interactive CICS terminal user interface (and using a wizard)
Creating a provider using the CICS web services assistant

• **Use this method for:** an existing CICS application that is fully functional and has a COMMAREA or channel interface

• **You will need:** a COBOL copybook (or PL/I, C/C++ equivalent)

```
Language structure (COBOL, PL/I, C/C++)

DFHLS2WS batch utility

wsbind Web service binding file

WSDL Web Service Description Language file
```
Creating the CICS infrastructure for a provider

• These steps apply to any method for creating a provider.

1. Create a **TCPIPSERVICE** resource.
2. Create a **pipeline configuration file**.
3. Create a **PIPELINE** resource.
4. Unless you use autoinstalled PROGRAM definitions, create a **PROGRAM** resource for each program in the pipeline.
Creating a provider using the CICS web services assistant

1. Run the **DFHLS2WS** batch utility (for example, specifying a COBOL copybook as the input file).

2. Copy the generated **wsbind** file to the pickup directory (the z/OS UNIX path specified by the WSDIR attribute of the PIPELINE resource).
   Optionally, copy the generated **WSDL** file to the same path (if you want to validate the SOAP messages).

3. Install the **PIPELINE** (dynamically creates the WEBSERVICE and URIMAP resources).
   The provider is ready for testing.
//SYSEGXLS JOB (39248C,A,T), 'LS2WS',
// MSGCLASS=A, NOTIFY=&SYSUID, REGION=0M
// SET QT=''''
// WHERESMA JCLLIB ORDER=CIRCLE.CICSWS.PROCLIB
// JAVAPROG EXEC DFHLS2WS,
// JAVADIR='Java601_64/J6.0.1_64', PATHPREF='/u', TMPDIR='/u/tmp',
// TMPFILE=&QT.&SYSUID.&QT, USSDIR='cicsts42'
// INPUT.SYSUT1 DD *
// PDSLIB=CIRCLE.CICSWS.COPYLIB
// REQMEM=PAYCOM1
// RESPMEM=PAYCOM1
// PGMINT=COMMAREA
// MAPPING-LEVEL=3.0
// MINIMUM-BASELEVEL=3.0
// MAPPING-LEVEL=3.0
// MINIMUM-BASELEVEL=3.0
// LANG=COBOL
// PGMNAME=PAYBUS
// URI=/paybus1
// WSBind=/u/usr/lpp/cicsts/cicsts42/samples/webservices/wsbind/provider/paybus1.wsbind
// WSDL=/u/usr/lpp/cicsts/cicsts42/samples/webservices/wsdl/paybus1.wsdl
// LOGFILE=/u/syssegx0/paybus
/*

JCL to run DFHLS2WS

Input COBOL copybook PDS members: one for the request, another for the response (same in this case)

Your existing CICS program

Output wsbind and WSDL files

Complete your session evaluations online at www.SHARE.org/AnaheimEval
DFHLS2WS log

DFHPI9609I Parameter "LOGFILE" has value "/u/sysegx0/paybus".

... 

DFHPI9609I Parameter "PDSLIB" has value "/CIRCLE.CICSWS.COPYLIB".  
DFHPI9609I Parameter "PGMINT" has value "COMMAREA".  
DFHPI9609I Parameter "PGMNAME" has value "PAYBUS".  
DFHPI9609I Parameter "REQMEM" has value "PAYCOM1".

...  

DFHPI9609I Parameter "RESPMEM" has value "PAYCOM1".

...  

DFHPI9609I Parameter "URI" has value "/paybus1".

... 

DFHPI9629I The minimum runtime level required for this Web service is "3.0".  

DFHPI9640I This Web service should be installed into a PIPELINE that uses SOAP version "1.1".  

DFHPI9587I Program "DFHLS2WS" has completed SUCCESSFULLY.
Testing the provider using RDz Web Services Tester

- The following slides demonstrate using the RDz Web Services Tester to test the provider:
  1. Create a CICS web service project in RDz
  2. Import the WSDL file
  3. Run the Web Services Tester
  4. Use the GUI to create and send a request to the provider
Testing the provider using RDz (1 of 8)
Testing the provider using RDz (2 of 8)

Create a Web Services for CICS Project

You can use this project to hold Web Services for CICS application components. You can also use this project as part of a service flow project.

Project name: DFHLS2WSTest

Options
Development scenario: Create New Service Implementation (top-down)
Application mode: Service Requestor
Conversion type: Interpretive XML Conversion

Scenario description:
Generate high level language data structures and runtime specific XML message processing from a Web service description. You can use this option to (1) Create a new service provider application program (2) Expose an existing application program as a service provider or (3) Construct a new service requester application program.
Testing the provider using RDz (3 of 8)
Testing the provider using RDz (4 of 8)
Testing the provider using RDz (5 of 8)
Testing the provider using RDz (6 of 8)
Testing the provider using RDz (7 of 8)
Testing the provider using RDz (8 of 8)

Invoke a WSDL Operation

Enter the parameters for the WSDL operation "PAYBUSOperation" and click Go to invoke.

Endpoints

http://192.86.32.129:6000/paybus

Body

PAYBUSOperation

ws_payroll_data

ws_request string

DISP

ws_key

ws_department string

1

ws_employee_no string

00001
Creating a provider using Rational Developer for System z (RDz)

- Step-by-step wizard, with two options for runtime XML conversion:
- **Interpretive** uses a standard wrapper program, as per the CICS assistant
- **Compiled** generates a bespoke COBOL application handler (wrapper program)

Diagram:

```
Language structure
(COBOL, PL/I, C/C++)

RDz
interactive wizard in Eclipse GUI

wsbind
WSDL
Application handler
COBOL program

Compiled option only
```
Creating a provider using RDz: interpretive (1 of 9)
Creating a provider using RDz: interpretive (2 of 9)
Creating a provider using RDz: interpretive (3 of 9)
Creating a provider using RDz: interpretive (4 of 9)
Creating a provider using RDz: interpretive (5 of 9)
Creating a provider using RDz: interpretive (6 of 9)
Creating a provider using RDz: interpretive (7 of 9)
Creating a provider using RDz: interpretive (8 of 9)
Creating a provider using RDz: interpretive (9 of 9)
Creating a provider using RDz: compiled (1 of 6)
Creating a provider using RDz: compiled (2 of 6)
Creating a provider using RDz: compiled (3 of 6)
Creating a provider using RDz: compiled (4 of 6)
Creating a provider using RDz: compiled (5 of 6)
Creating a provider using RDz: compiled (6 of 6)
Creating a provider using RDz: after running the RDz wizard

1. Transfer the wsbind file to the z/OS UNIX pickup directory. Optionally, transfer the WSDL file to the same directory.

2. Compiled option only (generated wrapper program):
   - Compile and link the COBOL source program
   - Create a PROGRAM resource

3. Issue a PIPELINE SCAN command.
Creating a provider using RDz Service Flow Modeler

1. In RDz, create a Service Flow Project. This starts a wizard that directs you to:
2. Define a host connection (to the z/OS system mainframe that hosts your CICS application).
3. Navigate to the “start” screen (signon to CICS, start the transaction, clear the screen).
4. Start recording the “flow” (your input, and the transaction output).
5. For each input field (request data), specify a variable name.
6. For each output field (response data), highlight the item on the screen, and specify a variable name.
7. Stop recording. This generates a .seqflow file.
9. Click Generate Runtime code. (This wizard can submit the compile JCL on z/OS for you.)
10. The generated code includes a web service provider COBOL program that drives your original CICS application.
Creating a web service requester in CICS

Web service requester

Your CICS program

SOAP request

Network

SOAP response

Web service provider
Methods for creating a web service requester in CICS

1. **CICS web services assistant** from a WSDL, using the DFHWS2LS batch utility

2. **RDz** from a WSDL (using a wizard), with interpretive runtime XML conversion, as per DFHWS2LS, above (no compiled option for a requester)

- Both methods generate copybooks and a wsbind file. However, the RDz also generates COBOL source for a requester program, demonstrating how to use the EXEC CICS INVOKE WEBSERVICE command.
Creating a requester using the CICS web services assistant

- **You will need:** the WSDL for the web service that you want to use

---

**WSDL**
- Web Service Description Language file

**DFHWS2LS**
- batch utility

**wsbind**
- Web service binding file

**Language structure**
- (COBOL, PL/I, C/C++)
Creating the CICS infrastructure for a requester

- Identical to the steps for a provider, except that a requester does not require a TCPIPSERVICE or a URIMAP resource

1. Create a **pipeline configuration file**.
2. Create a **PIPELINE** resource.
3. Unless you use autoinstalled PROGRAM definitions, create a **PROGRAM** resource for each program in the pipeline.
Creating a requester using the CICS web services assistant

1. Run the **DFHWS2LS** batch utility (for example, specifying a COBOL copybook as the input file).

2. Copy the generated **wsbind** file to the pickup directory (the z/OS UNIX path specified by the WSDIR attribute of the PIPELINE resource). Optionally, copy the generated **WSDL** file to the same path.

3. Install the **PIPELINE** (dynamically creates the WEBSERVICE resource).

4. Add an **EXEC CICS INVOKE WEBSERVICE** command to your COBOL program to send the request, and additional code to process the response.

The requester is ready for testing.
JCL to run DFHWS2LS

//SYSEGXLS JOB (39248C,A,T),'LS2WS',
// MSGCLASS=A,NOTIFY=&SYSUID,REGION=0M
// SET QT=''''
//WHERESMA JCLLIB ORDER=CIRCLE.CICSWS.PROCLIB
//JAVA PROG EXEC DFHWS2LS,
// JAVADIR='Java601_64/J6.0.1_64',PATHPREF='/u',TMPDIR='/u/tmp',
// TMPFILE=&QT.&SYSUID.&QT,USSDIR='cicsts42'
//INPUT.SYSUT1 DD *
PDSLIB=CIRCLE.CICSWS.COPYLIB
REQMEM=REQCOM
RESPMEM=RESPCOM
MAPPING-LEVEL=3.0
MINIMUM-RUNTIME-LEVEL=CURRENT
LANG=COBOL
WSBIND=/u/usr/lpp/cicsts/cicsts42/samples/webservices/wsbind/requester/*
paybus6.wsbind
WSDL=/u/usr/lpp/cicsts/cicsts42/samples/webservices/wsd1/paybus.wsdl
LOGFILE=/u/sysegx0/paybus6
/*

Output COBOL copybook PDS members:
one for the request, another for the response

Output wsbind file

Input WSDL file
Corresponding XML snippet

```xml
<wsXpayrollXdata>
  <wsXrequest>DISP</wsXrequest>
  <wsXkey>
    <wsXdepartment>1</wsXdepartment>
    <wsXemployeeXno>00001</wsXemployeeXno>
  </wsXkey>
  <wsXname>CIRCLE COMPUTER 1</wsXname>
  ...
</wsXpayrollXdata>
```

XML allows hyphens in element names, but some applications and programming languages interpret such hyphens as minus signs (mathematical operators), with undesirable results.
Sending a request to a web service from a CICS COBOL program

EXEC CICS INVOKE
  WEBSERVICE(CV-WEBSERVICE)
  CHANNEL(CV-CHANNEL-NAME)
  OPERATION(CV-OPERATION)
  URI(CV-URI)
  RESP(WS-EIB-RESP)
END-EXEC.

The RDz wizard generates a sample CICS COBOL program that does this.
Creating a requester using RDz

- **WSDL:** Web Service Description Language file
- **RDz:** Interactive wizard in Eclipse GUI
- **wsbind:** Web service binding file
- **Language structure:** (COBOL, PL/I, C/C++)
- **Example requester program**

The equivalent CICS web services assistant batch utility (DFHWS2LS) does not create this.
Creating a requester using RDz (1 of 8)
Creating a requester using RDz (2 of 8)

Project name: WS2LSInterpretive

Options
- Development scenario: Create New Service Implementation (top-down)
- Application mode: Service Requestor
- Conversion type: Interpretive XML Conversion

Scenario description:
Generate high level language data structures and runtime specific XML message processing from a Web service description. You can use this option to (1) Create a new service provider application program (2) Expose an existing application program as a service provider or (3) Construct a new service requester application program.
Creating a requester using RDz (3 of 8)
Creating a requester using RDz (4 of 8)

**DFHWS2LS: Application and Service Properties**

Generate high level language structures and a Web service binding file from a Web service description.

- **Application properties**
- **Service properties**

**Application language:** COBOL

**Program name:** PAYBUS

Change WSBind preferences
Creating a requester using RDz (5 of 8)

DFHWS2LS: Application and Service Properties

Generate high level language structures and a Web service binding file from a Web service description.

- **Application properties**
  - Local URI: `/cics/services/paybus`
  - WSDL service: 
  - Binding element: `PAYBUSHTTPSoapBinding`
  - Available operations: `PAYBUSOOperation`

- **Service properties**

**Change WSBind preferences**
Creating a requester using RDz (6 of 8)
Creating a requester using RDz (7 of 8)
Creating a requester using RDz (8 of 8)

```
*+++++*A-1-B+++++2+++++3+++++4+++++5+++++6+++++7

PROCESS_CICS,NODYNAM,NSYMBOL(NATIONAL),TRUNC(STD)

* New CICS TS Web Service Requester
* +++++++++++++++++++++++++++++++++++++++++++++++++++++

IDENTIFICATION DIVISION.
* Begin Identification Division
  PROGRAM-ID. 'PAYBUS'.
  AUTHOR. RDZ.
  INSTALLATION. 10.0.0.V20130529_1611.
  DATE-WRITTEN. 2/18/14 1:12 PM.
* End Identification Division

DATA DIVISION.
* Begin Data Division
  WORKING-STORAGE SECTION.
* Begin Working-Storage Section
* +++++++++++++++++++++++++++++++++++++++++++++++++++++

* Operations Available on the Remote Web Service
* +++++++++++++++++++++++++++++++++++++++++++++++++++++

1 OPERATION-NAME-1.
2 PIC X(15) USAGE DISPLAY
   VALUE 'PAYBUSOperation'.
```
Structure of the pipeline definition for a service provider

- provider_pipeline
  - cics_mtom_handler
  - transport
    - dfhmtom_configuration
      - default_transport_handler_list
        - handler
      - default_http_transport_handler_list
        - handler
      - default_mq_transport_handler_list
        - handler
      - named_transport_entry
        - transport_handler_list
  - service
    - service_handler_list
      - handler
      - cics_soap_1.1_handler
      - cics_soap_1.2_handler
      - wsse_handler
    - terminal_handler
    - apphandler
    - service_parameter_list
      - handler
        - cics_soap_1.1_handler
        - cics_soap_1.2_handler
        - cics_soap_1.1_handler_java
        - cics_soap_1.2_handler_java
Diagnosing web services in CICS: sniffing containers in the pipeline

- The IBM Redbook *Implementing CICS Web Services*, SG24-7206, presents a simple “sniffer” program that displays (in tdqueue CESE) the contents of the containers available in the pipeline.
- To use the sniffer, you add it to the pipeline (configuration file) as a message handler.

```xml
<provider_pipeline>
  <service>
    <service_handler_list>
      <handler>
        <program>SNIFFER</program>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```
Sniffer output (1 of 5)

CPIH 20120314113934 SNIFFER : *** Start ***
CPIH 20120314113934 SNIFFER : >================================<
CPIH 20120314113934 SNIFFER : Container Name   : DFHFUNCTION
CPIH 20120314113934 SNIFFER : Content length   : 00000016
CPIH 20120314113934 SNIFFER : Container content: RECEIVE-REQUEST
CPIH 20120314113934 SNIFFER : Containers on channel: List starts.
CPIH 20120314113934 SNIFFER : >================================<
CPIH 20120314113934 SNIFFER : Container Name   : DFHFUNCTION
CPIH 20120314113934 SNIFFER : Content length   : 00000016
CPIH 20120314113934 SNIFFER : Container content: RECEIVE-REQUEST
CPIH 20120314113934 SNIFFER : >================================<
CPIH 20120314113934 SNIFFER : Container Name   : DFHWS-URI
CPIH 20120314113934 SNIFFER : Content length   : 00000008
CPIH 20120314113934 SNIFFER : Container content: /paybus1
CPIH 20120314113934 SNIFFER : >================================<
CPIH 20120314113934 SNIFFER : Container Name   : DFHREQUEST
CPIH 20120314113934 SNIFFER : Content length   : 00002928
CPIH 20120314113934 SNIFFER : Container content:

<SOAP-ENV:Envelope ... >
<SOAP-ENV:Body ... >
<PAYBUSOOperationRequest>
 <ws_payroll_data>
  <ws_request>DISP</ws_request>
  <ws_key>
   <ws_department>1</ws_department>
   <ws_employee_no>00001</ws_employee_no>
  </ws_key>
...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Sniffer output (2 of 5)

CPIH 20120314113934  SNIFFER :  >================================<
CPIH 20120314113934  SNIFFER :  Container Name   :  DFHWS-PIPELINE
CPIH 20120314113934  SNIFFER :  Content length   :  00000008
CPIH 20120314113934  SNIFFER :  Container content:  CICSWSS
CPIH 20120314113934  SNIFFER :  >================================<
CPIH 20120314113934  SNIFFER :  Container Name   :  DFHWS-USERID
CPIH 20120314113934  SNIFFER :  Content length   :  00000008
CPIH 20120314113934  SNIFFER :  Container content:  CICST541
CPIH 20120314113934  SNIFFER :  >================================<
CPIH 20120314113934  SNIFFER :  Container Name   :  DFHWS-TRANID
CPIH 20120314113934  SNIFFER :  Content length   :  00000004
CPIH 20120314113934  SNIFFER :  Container content:  CPIH
CPIH 20120314113934  SNIFFER :  >================================<
CPIH 20120314113934  SNIFFER :  Container Name   :  DFHWS-WEBSERVICE
CPIH 20120314113934  SNIFFER :  Content length   :  00000032
CPIH 20120314113934  SNIFFER :  Container content:  paybus1
CPIH 20120314113934  SNIFFER :  >================================<
CPIH 20120314113934  SNIFFER :  Container Name   :  DFHWS-APPHANDLER
CPIH 20120314113934  SNIFFER :  Content length   :  00000008
CPIH 20120314113934  SNIFFER :  Container content:  DFHPITP
CPIH 20120314113934  SNIFFER :  Containers on channel: List ends
CPIH 20120314113934  SNIFFER :  DFHRESPONSE       container deleted
CPIH 20120314113934  SNIFFER :  **** End ****
Sniffer output (3 of 5)

CPIH 20120314113934 SNIFFER :  *** Start ***
CPIH 20120314113934 SNIFFER :  >================================<
CPIH 20120314113934 SNIFFER :  Container Name   : DFHFUNCTION
CPIH 20120314113934 SNIFFER :  Content length   : 00000016
CPIH 20120314113934 SNIFFER :  Container content: SEND-RESPONSE
CPIH 20120314113934 SNIFFER :  Containers on channel: List starts.
CPIH 20120314113934 SNIFFER :  >================================<
CPIH 20120314113934 SNIFFER :  Container Name   : DFHWS-OUTACTION
CPIH 20120314113934 SNIFFER :  Content length   : 00000067
CPIH 20120314113934 SNIFFER :  Container content: C"http://www.PAYBUS.PAYCOM1.com/PAYBUSPort/PAYBUSOperationResponse"
CPIH 20120314113934 SNIFFER :  >================================<
CPIH 20120314113934 SNIFFER :  Container Name   : DFHWS-WSDL-CTX
CPIH 20120314113934 SNIFFER :  Content length   : 00000116
CPIH 20120314113934 SNIFFER :  Container content: http://www.PAYBUS.PAYCOM1.com PAYBUSOperation
http://www.PAYBUS.PAYCOM1.com
http://www.PAYBUS.PAYCOM1.com PAYBUSPort
CPIH 20120314113934 SNIFFER :  >================================<
CPIH 20120314113934 SNIFFER :  Container Name   : DFHWS-OPERATION
CPIH 20120314113934 SNIFFER :  Content length   : 00000015
CPIH 20120314113934 SNIFFER :  Container content: PAYBUSOperation
Sniffer output (4 of 5)

CPIH 20120314113934 SNIFFER : >================================<
CPIH 20120314113934 SNIFFER : Container Name : DFHRESPONSE
CPIH 20120314113934 SNIFFER : Content length : 00002446
CPIH 20120314113934 SNIFFER : Container content:

<SOAP-ENV:Envelope ... >
<SOAP-ENV:Body>
<PAYBUSOperationResponse ... >
<ws_payroll_data>
<ws_request>DISP</ws_request>
<ws_key>
<ws_department>1</ws_department>
<ws_employee_no>00001</ws_employee_no>
</ws_key>
<ws_name>SHARE</ws_name>
<ws_addr1>QUEENSBURY HSE</ws_addr1>
<ws_addr2>BRIGHTON</ws_addr2>
<ws_addr3>SUSSEX</ws_addr3>
<ws_phone_no>75529900</ws_phone_no>
<ws_timestamp></ws_timestamp>
<ws_salary>1234.56</ws_salary>
<ws_start_date>28101984</ws_start_date>
<wsRemarks>CIRCLE IS MAGIC</wsRemarks>
<ws_msg></ws_msg>
<ws_upd_inds>
<ws_upd_name></ws_upd_name>
...
<table>
<thead>
<tr>
<th>Container Name</th>
<th>Content length</th>
<th>Container content</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFHFUNCTION</td>
<td>0000016</td>
<td>SEND-RESPONSE</td>
</tr>
<tr>
<td>DFHWS-WEBSERVICE</td>
<td>0000032</td>
<td>paybus1</td>
</tr>
<tr>
<td>DFHWS-APPHANDLER</td>
<td>00000008</td>
<td></td>
</tr>
</tbody>
</table>

Containers on channel: List ends

**** End ****
Summary

• To create a service provider or requester in CICS:
  • Create a PIPELINE resource and pipeline configuration file.
  • Provider only: create a TCPIPSERVICE resource.
  • Use CICS web service assistant or RDz to create wsbind (and WSDL) files. You will need a COBOL copybook (or other language structure) or a WSDL file.
  • Install the PIPELINE (or issue a PIPELINE SCAN command if already installed).
• Consider Service Flow Modeler for applications that do not have separate presentation and business logic structures.
• Add a sniffer program to the pipeline to diagnose problems.