Introduction to CICS Web Services for the System Programmer

Ezriel Gross
Circle Software Incorporated

August 14th, 2013 (Wed)
9:30am – 10:30pm
Session 13374
Agenda

• Background: evolution of traditional CICS application design
  • Pseudoconversational transactions
  • Program communication (COMMAREAes, channels/containers)
• Introduction to XML
• Introduction to SOAP and web services
• Support for SOAP and web services in CICS
  • CICS web services tooling
• Implementation and configuration of web services in CICS
• CICS runtime support
• Summary
Conversational transactions

• *Never* design these!

• When “conversing”, the transaction retains all the associated storage

• To prevent this, use pseudo-conversational techniques...
Pseudoconversational transactions

- Use the TRANSID= option of the Return command to nominate the next transaction to be invoked.

- Place data to be passed from one transaction to another in COMMAREA.
Pseudoconversational programming

• Pros
  • Less resource-intensive
  • User think time costs eliminated
  • Can run more simultaneous transactions

• Cons
  • Programming more difficult
  • Application design requires re-starting program
  • More difficult to debug programs
  • Programs require area to store data between tasks
Communication Area (COMMAREA)

- Terminal-based scratchpad
- Dynamic in size (0 to 32,500 bytes)
- Private and chained to the terminal

Identification division
Program-Id. ‘PAYROLL’.
Working-storage Section.
  01 task-data.
    02 data1  pic s9(8) comp.
    02 data2  pic x(4).
Linkage Section.
  01 dfhcommarea pic x(8).
Procedure Division.
  if eibcalen = 0
    add 1 to data1
    exec cics send ... first screen
    exec cics return transid (‘PAY ’) commarea (task-data) end-exec.
  end if

  move dfhcommarea to task-data.

  if data1 = 1
    add 1 to data1
    exec cics send ... second screen
    exec cics return transid (‘PAY ’) commarea (task-data) end-exec.
  end if

  if data1 = 2
    exec cics send ... third screen
    exec cics return end-exec.
COMMAREA basics

- Used to hold data between pseudo-conversations
- 32K maximum in size
- It can be passed from program to program
- It can be empty, partially full, or full
- Is automatically cleaned up when not used
- Is attached to the terminal
COMMAREA for passing data to next task

PAY

Working storage

PAY-DATA

Copy

COMMAREA

PAYROLL

PROCEDURE...

RETURN TRANSID('ACCT')

COMMAREA(PAY-DATA)

ACCT

ACCOUNT

LINKAGE...

DFHCOMMAREA
Continually using COMMAREA
CICS essential design concepts

- Pseudo-conversational programming
- Communication area: COMMAREA
- Program-to-program communication
Program-to-program communication

• Modular design and code reusability incorporated

• CALL ➔ non-CICS, can be used with CICS
• LINK ➔ CICS to CICS
• XCTL ➔ CICS to CICS
• ECI Call ➔ CICS Client to CICS
• EXCI Call ➔ z/OS to CICS

• COMMAREA supported in all call types
Contrasting various call types

**LINK**

CICS client

\[
\text{CALL 'PAYBUS'} = \text{LINK PROGB}
\]

**Distributed program link**

CICSA

CICS client

\[
\text{CALL 'PAYBUS'} = \text{LINK PAYBUS}
\]

**ECI**

non-CICS

\[
\text{CICS client CALL 'PAYBUS' = LINK}
\]

**EXCI**

\[
\text{LINK PAYBUS}
\]

**z/OS batch or TSO**

Complete your sessions evaluation online at SHARE.org/BostonEval
COMMAREA challenges

• Limited to 32K in length
• Only one can be passed to a program
• Contains different data types
• Contains unrelated data structures
• Entire COMMAREA available to all programs in UOW
Channels and containers

- No size limitation of data in containers
- Number of containers in a channel unlimited
- Can have separate containers for input and output
A COMMAREA alternative

EXEC CICS xxxx
COMMAREA(data-area)

EXEC CICS xxxx
CHANNEL(channel-name)

LINK using a channel

PAYPGM

LINK PAYBUS
CHANNEL(PAY-DATA)

PAYBUS

- 
- 
RETURN
CHANNEL(PAY-DATA)
Sample COMMAREA

********************************************************************************
* PAYROLL COMMUNICATION FIELDS *
********************************************************************************

01 ws-payroll-data.
   02 ws-request                  pic x(4).
   02 ws-key.
      05 ws-department           pic x.
      05 ws-employee-no         pic x(5).
   02 ws-name                    pic x(20).
   02 ws-addr1                   pic x(20).
   02 ws-addr2                   pic x(20).
   02 ws-addr3                   pic x(20).
   02 ws-phone-no                pic x(8).
   02 ws-timestamp               pic x(8).
   02 ws-salary                  pic x(8).
   02 ws-start-date              pic x(8).
   02 ws-remarks                 pic x(32).
   02 ws-msg                     pic x(60).

• Full COMMAREA length is 1036 bytes
Problems with raw data communication

- Data must be accessed using its relative position
- Modifications to fields require application changes
- Addition of new fields affect all existing users
- Data requires a separate document to describe fields
- Data is not portable therefore platform dependant
Introduction to XML

• Extensible Markup Language (XML), is owned by the World Wide Web Consortium (W3C).

• XML is an open standard protocol that provides a mechanism to create and define a meta-language that can be used to structure information.

• XML is extensible in that there are no pre-defined tags, as in HTML. Each set of users defines tags that have meaning to them and their peers, such as within a business, across an industry, or across multiple industries.

• XML is used to create new Internet languages.
Introduction to XML

- XML tagging means that the context of the data is retained along with the information. Therefore, a name is not just a string of characters but can be described as a customer name, last name, first name, and so forth.

- XML is portable and self-defining, down to the code page used, so you can ship XML to any platform and to any application worldwide. If that application is able to process XML, you know that the information can be retrieved and processed.

- As the creator of XML, you do not need to understand where the data will eventually be used.

- As the receiver of XML, you do not need to know where it originated.
XML front-end application

Content Creation

XML document content

XML | FOP

- PDF file
- Fax back
- Web content
- Handhelds
- VoiceML
- eBook content

Transcoding server
XML syntax

- XML documents use a self-describing, simple syntax.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<ws-payroll-data>
    <ws-request>DISP</ws-request>
    <ws-department>1</ws-department>
    <ws-employee>00001</ws-employee>
</ws-payroll-data>
```

- The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1) represents the Latin-1/West European character set.
- The `<ws-payroll-data>` tag describes the root element of the document.
- The `<ws-request>`, `<ws-department>` and `<ws-employee>` tags describe three child elements of the root.
- The `</ws-payroll-data>` tag defines the end of the root element.
XML glossary

• Extensible Stylesheet Language Transformation (XSLT) is the recommended style sheet language of XML.

• **XML namespaces:** In XML, element names are defined by a developer.

• **XML schema** is an XML document that describes the structure and constrains the contents of other XML documents.

• **XML parser** is a program that is invoked by an application to process an XML document, ensure that it meets all the rules of XML as well as the syntax of the DTD or schema, making the data available to the calling application.
XML technologies

• **Parsers:**
  - Document Object Model (DOM) builds a tree structure in memory as a result of parsing an input document.
  - Simple API for XML (SAX) triggers an event when a tag is encountered, causing execution of an event handler.
  - Most parsers validate an input document against a document type definition (DTD).

• **Generators:**
  - DOM based generators walk the tree structure and produce a valid XML document as a result.
  - Template based generators insert data values into a pre-built XML document template.

• **Transformers:**
  - Transform a document with one DTD into a document with a different DTD, using corresponding tags.
Parse XML input: alternatives

- COBOL XML PARSE command
- COBOL program logic
- COBOL UNSTRING command
- EXEC CICS TRANSFORM XMLTODATA (CICS V4)
  - Requires running DFHSC2LS in advance
COBOL XML PARSE statement:
Format

XML PARSE identifier-1
  ENCODING codepage
    RETURNING NATIONAL
  WITH

VALIDATING
  WITH
    identifier-2
      WITH
        FILE xml-schema-name-1

PROCESSING PROCEDURE
  IS
    procedure-name-1
      THROUGH
        procedure-name-2
        THRU

EXCEPTION imperative-statement-1
  ON

NOT
  EXCEPTION imperative-statement-2
  END-XML
COBOL XML PARSE statement: control flow

Program

XML PARSE

Processing procedure

Process event

End of parse

Parser

Detect event

Document end?

Yes

No
COBOL XML PARSE statement:
Special registers

- To receive and pass information:
  - **XML-CODE** to determine the status of XML parsing
  - **XML-EVENT** to receive the name of each XML event
  - **XML-TEXT** to receive XML document fragments from an alphanumeric document
  - **XML-NTEXT** to receive XML document fragments from a national document
## COBOL XML PARSE statement: XML-EVENT event parameters

<table>
<thead>
<tr>
<th>Attribute/Character Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE-CHARACTER</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE-CHARACTERS</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE-NAME</td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE-NATIONAL-CHARACTER</td>
<td></td>
</tr>
<tr>
<td>COMMENT</td>
<td></td>
</tr>
<tr>
<td>CONTENT-CHARACTER</td>
<td></td>
</tr>
<tr>
<td>CONTENT-CHARACTERS</td>
<td></td>
</tr>
<tr>
<td>CONTENT-NATIONAL-CHARACTER</td>
<td></td>
</tr>
<tr>
<td>DOCUMENT-TYPE-DECLARATION</td>
<td></td>
</tr>
<tr>
<td>ENCODING-DECLARATION</td>
<td></td>
</tr>
<tr>
<td>END-OF-CDATA-SECTION</td>
<td></td>
</tr>
<tr>
<td>END-OF-DOCUMENT</td>
<td></td>
</tr>
<tr>
<td>END-OF-ELEMENT</td>
<td></td>
</tr>
<tr>
<td>EXCEPTION</td>
<td></td>
</tr>
<tr>
<td>PROCESSING-INSTRUCTION-DATA</td>
<td></td>
</tr>
<tr>
<td>PROCESSING-INSTRUCTION-TARGET</td>
<td></td>
</tr>
<tr>
<td>STANDALONE-DECLARATION</td>
<td></td>
</tr>
<tr>
<td>START-OF-CDATA-SECTION</td>
<td></td>
</tr>
<tr>
<td>START-OF-DOCUMENT</td>
<td></td>
</tr>
<tr>
<td>START-OF-ELEMENT</td>
<td></td>
</tr>
<tr>
<td>UNKNOWN-REFERENCE-IN-ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td>UNKNOWN-REFERENCE-IN-CONTENT</td>
<td></td>
</tr>
<tr>
<td>VERSION-INFORMATION</td>
<td></td>
</tr>
</tbody>
</table>
COBOL XML PARSE statement: XML-EVENT event parameters

<!--This document is just an example--> WHEN 'COMMENT'
******************************************************************************
* 'COMMENT' = "This document is just an example"
******************************************************************************

<bread type="baker's best" />

WHEN 'START-OF-ELEMENT'
******************************************************************************
* 'START-OF-ELEMENT' = "bread"
******************************************************************************

WHEN 'ATTRIBUTE-NAME'
******************************************************************************
* 'ATTRIBUTE-NAME' = "type"
******************************************************************************

WHEN 'ATTRIBUTE-CHARACTERS'
******************************************************************************
* 'ATTRIBUTE-CHARACTERS' = "baker"
******************************************************************************

WHEN 'ATTRIBUTE-CHARACTER'
******************************************************************************
* 'ATTRIBUTE-CHARACTER' = "' (apos;)
******************************************************************************

WHEN 'ATTRIBUTE-CHARACTERS'
******************************************************************************
* 'ATTRIBUTE-CHARACTERS' = "s best"
******************************************************************************

WHEN 'END-OF-ELEMENT'
******************************************************************************
* 'END-OF-ELEMENT' = "bread"
******************************************************************************
COBOL XML PARSE statement: XML-EVENT event parameters

UNSTRING LS-BUFFER
   DELIMITED BY
     '<WS-Request>' OR
     '<ws-request>' OR
     '<WS-REQUEST>' OR
     '</WS-Request>' OR
     '</ws-request>' OR
     '</WS-REQUEST>'
   INTO WS-WS WS-REQUEST
END-UNSTRING.
Generate XML output: alternatives

- COBOL XML  GENERATE statement
- COBOL MOVE  CORRESPONDING statement
- Roll-your-own COBOL code
- EXEC CICS TRANSFORM DATATOXML (CICS V4)
  - Requires running DFHLS2SC in advance
COBOL XML GENERATE statement: Format

```cobol
XML GENERATE identifier-1 FROM identifier-2
  COUNT identifier-3
  ENCODING codepage
  WITH XML-DECLARATION
  WITH ATTRIBUTES
  NAMESPACE identifier-4
    IS literal-4
  NAMESPACE-PREFIX
    IS identifier-5
    IS literal-5
  EXCEPTION imperative-statement-1
    ON
  NOT ON
  EXCEPTION imperative-statement-2
  END-XML
```
XML GENERATE statement: example (1 of 3)

01 FILLER PIC X(16) VALUE 'RESPONSE HEADERS'.
01 RESPONSE-HEADERS.
   05 CONTENT-TYPE-VX PIC X(8) VALUE 'text/xml'.
   05 CONTENT-TYPE-VX-L PIC S9(8) COMP VALUE +8.
01 FILLER PIC X(8) VALUE 'customer'.
01 ws-payroll-data.
   02 ws-request            pic x(4).
   02 ws-key.
      05 ws-department     pic x.
      05 ws-employee-no    pic x(5).
   02 ws-name            pic x(20).
   02 ws-addr1           pic x(20).
   02 ws-addr2           pic x(20).
   02 ws-addr3           pic x(20).
   02 ws-phone-no        pic x(8).
   02 ws-timestamp       pic x(8).
   02 ws-salary          pic x(8).
   02 ws-start-date      pic x(8).
   02 ws-remarks         pic x(32).
   02 ws-msg             pic x(60).

01 FILLER PIC X(8) VALUE 'XML-AREA'.
01 XML-AREA.
   05 FILLER PIC X(43) VALUE '<?xml version="1.0" encoding="ISO-8859-1"?>'.
   05 XMLOUT PIC X(512) VALUE SPACES.
XML GENERATE statement: example (2 of 3)

WHEN '2'
   EXEC CICS
       LINK
       PROGRAM('WBCUSB99')
       COMMAREA(customer)
   END-EXEC

   XML GENERATE
       XMLOUT FROM
           ws-payroll-data

   PERFORM 0700-SEND-CUST
   PERFORM 0900-RETURN

   EXEC CICS
       DOCUMENT CREATE
           DOCTOKEN(DOCUMENT-TOKEN)
           NOHANDLE
       END-EXEC.

   EXEC CICS
       DOCUMENT INSERT
           DOCTOKEN(DOCUMENT-TOKEN)
           TEXT(XML-AREA)
           LENGTH(LENGTH OF XML-AREA)
           NOHANDLE
       END-EXEC

   PERFORM 0800-FILL-CONTENT-TYPE-XML
   PERFORM 0800-FILL-PRAGMA-HEADER.
   EXEC CICS
       WEB SEND
           DOCTOKEN(DOCUMENT-TOKEN)
           CLNTCODEPAGE('819')
       END-EXEC.
XML GENERATE statement: example (3 of 3)

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ws-payroll-data>
<ws-request><ws-request>DISP</ws-request>
<ws-key>
<ws-department>1</ws-department>
<ws-employee-no>00001</ws-employee-no>
</ws-key>
<ws-name>EZRIEL</ws-name>
<ws-addr1>123 WILLOWBROOK BLVD</ws-addr1>
<ws-addr2>WAYNE, NJ</ws-addr2>
<ws-addr3>07470</ws-addr3>

</ws-payroll-data>
```
MOVE CORRESPONDING statement:
message text

01 XML-MESSAGE-TEXT.
   05 FILLER PIC X(12) VALUE '<LOANSTATUS>'.
   05 FILLER PIC X(06) VALUE '<Loan>'.
   05 FILLER PIC X(09) VALUE '<LoanNum>'.
   05 LS-LOAN-NUM PIC X(10) VALUE SPACE.
   05 FILLER PIC X(10) VALUE '</LoanNum>'.
   05 FILLER PIC X(09) VALUE '<SalesAU>'.
   05 LS-SALES-AU PIC X(06) VALUE SPACE.
   05 FILLER PIC X(10) VALUE '</SalesAU>'.
   05 FILLER PIC X(15) VALUE '<FulfillmentAU>'.
   05 LS-FULL-AU PIC X(06) VALUE SPACE.
   05 FILLER PIC X(16) VALUE '</FulfillmentAU>'.
   05 FILLER PIC X(18) VALUE '<BorrowerLastName>'.
   05 LS-BORR-LNAME PIC X(30) VALUE SPACE.
   05 FILLER PIC X(19) VALUE '</BorrowerLastName>'.
   ..........................................................
   05 FILLER PIC X(09) VALUE '<HMCLast>'.
   05 LS-HMC-LNAME PIC X(30) VALUE SPACE.
   05 FILLER PIC X(10) VALUE '</HMCLast>'.
   05 FILLER PIC X(16) VALUE '<HMCOfficePhone>'.
   05 LS-HMC-OFF-PHONE PIC X(10) VALUE SPACE.
   05 FILLER PIC X(17) VALUE '</HMCOfficePhone>'.
   05 FILLER PIC X(11) VALUE '</Employee>'.
   05 FILLER PIC X(13) VALUE '</LOANSTATUS>'.
MOVE CORRESPONDING statement: message text

01 WS-COMMAREA.
   05 LINK-LOAN-NUM PIC x(10).
   05 LINK-SYS-ORIG PIC x(03).

05 HTML-LOAN-STATUS.
   10 LS-LOAN-NUM PIC X(10) VALUE SPACE.
   10 LS-SALES-AU PIC X(06) VALUE SPACE.
   10 LS-FULL-AU PIC X(06) VALUE SPACE.
   10 LS-BORR-LNAME PIC X(30) VALUE SPACE.

   10 LS-HMC-LNAME PIC X(30) VALUE SPACE.
   10 LS-HMC-OFF-PHONE PIC X(10) VALUE SPACE.
   10 LS-ERROR-MSG PIC X(50) VALUE SPACE.

*****************************************************
* BUILD THE XML RESPONSE.
*****************************************************
0300-BUILD-MESSAGE.

MOVE CORRESPONDING HTML-LOAN-STATUS TO XML-MESSAGE-TEXT.
Character entities

- Five special pieces of code for characters that are recognized as part of the XML language.
- Must be used if these characters are used as part of an elements content.
- Character entities use a special syntax to represent the character in a way that avoids the XML parser interpreting the character as code.
- Standard character entities are:
  - `&lt;` represents the ‘<’ character
  - `&gt;` represents the ‘>’ character
  - `&amp;` represents the ‘&’ character
  - `&apos;` represents the “” character
  - `&quot;` represents the “” character
- To add characters that are not generally available XML supports character references using its Unicode character code: `&#nnn;` where `nnn` is the code
Generic XML Tester

Written by Circle Software Inc

XML Function: PAYXML

Loop Count: 1

XML Envelope:

```xml
<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>
```

XML Response:

```xml
<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 EDUCATION LTD</ws_name><ws_addr1>QUEENSBUY HSE</ws_addr1><ws_addr2>BRIGHTON</ws_addr2><ws_addr3>SUSSEX</ws_addr3><ws_phone_no>75529900</ws_phone_no><ws_timestamp><ws_salary>1234.56</ws_salary><ws_start_date>28101984</ws_start_date><ws_remarks>CIRCLE IS MAGIC</ws_remarks><ws_upd_ind><ws_item_no>Y</ws_item_no></ws_upd_ind><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_line_out><ws_payroll_data>
```
CICS web services overview

- Web services terms and concepts available in CICS TS
- Web services architecture:
  - Service provider
  - Service requestor
  - Service registry
- WSDL
- SOAP messages
- Message handlers and pipelines
- PIPELINE and WEBSERVICE resource definitions
Introduction to web services

• A web service is a collection of operations that are network accessible through standardized XML messaging.
• The web services architecture is based upon interactions between three components:
  • Service provider is the platform that hosts access to the service.
  • Service requester is the application that is looking for and invoking or initiating an interaction with a service.
  • Service registry is a place where service providers publish their service descriptions, and where service requesters find them.
• Universal Description, Discovery and Integration (UDDI) is a specification for distributed web-based information registries of web services.
Web services architecture

- **Architecture**
  - **Service provider**
    - “Owns” the service
    - Creates the WSDL
    - Publishes the WSDL
    - Processes the requests
  - **Service requester**
    - “Finds” the service
    - Binds to the service
      - *Invokes the service using the WSDL*
  - **Service registry**
    - Hosts the service description
    - Optional for statically bound requesters

- **Find**
  - WSDL, UDDI

- **Publish**
  - WSDL, UDDI

- **Bind**
  - SOAP
Web Service Description Language (WSDL)

- Web Service Description Language (WSDL) is an XML application for describing web services.
- WSDL is comprised of:
  - **Types**: the data types in the form of XML schemas
  - **Message**: an abstract definition of the data in the form of a message
  - **PortType**: an abstract set of operations mapped to one or more end points
  - **Binding**: the concrete protocol and data formats for the operations
  - **Service**: a collection of related end points
WSDL generation: Input

```java
//SYSEGXLS JOB (39248C,A,T),'LS2WS',
// MSGCLASS=A,NOTIFY=&SYSUID,REGION=0M
// SET QT='''
//WHERESTM JCLLIB ORDER=CIRCLE.CICSWS.PROCLIB
//JAVAPROG EXEC DFHLS2WS,
// JAVADIR='Java/J6.0.1_64',PATHPREF='/u',TMPDIR='/u/tmp',
// TMPFILE=&QT.&SYSUID.&QT,USSDIR='cicsts42'
//INPUT.SYSUT1 DD *
PDSLIB=CIRCLE.CICSWS.COPYLIB
REQMEM=PAYCOM
RESPMEM=PAYCOM
PGMINT=COMMAREA
MAPPING-LEVEL=3.0
LANG=COBOL
PGMNAME=PAYBUS
URI=/paybus
WSBIND=/u/usr/lpp/cicsts/cicsts42/samples/webservices/paybus.wsbind
WSDL=/u/usr/lpp/cicsts/cicsts42/samples/webservices/paybus.wsdl
LOGFILE=/u/sysegx0/paybus
/*
```

COBOL data structure

```plaintext
01 ws-payroll-data.
   02 ws-request pic x(4).
   02 ws-key.
      05 ws-department pic x.
      05 ws-employee-no pic x(5).
   02 ws-name pic x(20).
   02 ws-addr1 pic x(20).
   02 ws-addr2 pic x(20).
   02 ws-addr3 pic x(20).
   02 ws-phone-no pic x(8).
   02 ws-timestamp pic x(8).
   02 ws-salary pic x(8).
   02 ws-start-date pic x(8).
   02 ws-remarks pic x(32).
   02 ws-msg pic x(60).
```
WSDL generation: Output (1 of 7)

<?xml version="1.0" ?>
<!--This document was generated using 'DFHLS2WS' at mapping level '3.0'. -->
<definitions targetNamespace="http://www.PAYBUS.PAYCOM.com"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:reqns="http://www.PAYBUS.PAYCOM.Request.com"
 xmlns:resns="http://www.PAYBUS.PAYCOM.Response.com"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://www.PAYBUS.PAYCOM.com">
  <types> …
  <message name="PAYBUSOperationResponse"> …
  <message name="PAYBUSOperationRequest"> …
  <portType name="PAYBUSPort"> …
  <binding name="PAYBUSHTTPSoapBinding" type="tns:PAYBUSPort"> …
  <service name="PAYBUSService"> …
</definitions>
WSDL generation: Output (2 of 7)

- Types identifies the data types in the form of XML schemas

  `<types>
  
  <xsd:schema attributeFormDefault="qualified"
    elementFormDefault="qualified"
    targetNamespace="http://www.PAYBUS.PAYCOM.Request.com"
    xmlns:tns="http://www.PAYBUS.PAYCOM.Request.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"> …
  
  <xsd:schema attributeFormDefault="qualified"
    elementFormDefault="qualified"
    targetNamespace="http://www.PAYBUS.PAYCOM.Response.com"
    xmlns:tns="http://www.PAYBUS.PAYCOM.Response.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"> …
  
  </types>`
WSDL generation: Output (3 of 7)

```xml
<xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
  targetNamespace="http://www.PAYBUS.PAYCOM.Request.com"
  xmlns:tns="http://www.PAYBUS.PAYCOM.Request.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation source="http://www.ibm.com/software/htp/cics/annotations">This
    schema was generated by the CICS Web services assistant.</xsd:documentation>
  </xsd:annotation>
  <xsd:annotation>
      com.ibm.cics.wsd1.properties.mappingLevel=3.0
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType abstract="false" block="#all" final="#all" mixed="false"
    name="ProgramInterface"> …
  <xsd:element name="PAYBUSOperation" nillable="false" type="tns:ProgramInterface"/>
</xsd:schema>
```
WSDL generation: Output (4 of 7)

```xml
<xsd:complexType abstract="false" block="#all" final="#all" mixed="false"
name="ProgramInterface">
  <xsd:sequence>
    <xsd:element name="ws_payroll_data" nillable="false">
      <xsd:complexType mixed="false">
        <xsd:sequence>
          <xsd:element name="ws_request" nillable="false">
            <xsd:simpleType>
              <xsd:annotation>
                  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="4"/>
                <xsd:whiteSpace value="collapse"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```
WSDL generation: Output (5 of 7)

• Message describes an abstract definition of the data in the form of a message

  <message name="PAYBUSOperationResponse">
    <part element="resns:PAYBUSOperationResponse" name="ResponsePart"/>
  </message>

  <message name="PAYBUSOperationRequest">
    <part element="reqns:PAYBUSOperation" name="RequestPart"/>
  </message>

• PortType describes an abstract set of operations mapped to one or more end points

  <portType name="PAYBUSPort">
    <operation name="PAYBUSOperation">
      <input message="tns:PAYBUSOperationRequest" name="PAYBUSOperationRequest"/>
      <output message="tns:PAYBUSOperationResponse" name="PAYBUSOperationResponse"/>
    </operation>
  </portType>
• Binding is the concrete protocol and data formats for the operations

```xml
<wSDL>
  <binding name="PAYBUSHTTPSoapBinding" type="tns:PAYBUSPort">
    <!-- This soap:binding indicates the use of SOAP 1.1 -->
    <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>
</wSDL>
```
WSDL generation: Output (7 of 7)

• Service is a collection of related end points

```xml
<service name="PAYBUSService">
  <port binding="tns:PAYBUSHTTPSoapBinding" name="PAYBUSPort">
    <!-- This soap:address indicates the location of the Web service over HTTP. Please replace "my-server" with the TCPIP host name of your CICS region. Please replace "my-port" with the port number of your CICS TCPIPSERVICE. -->
    <soap:address location="http://my-server:my-port/paybus"/>
    <!-- This soap:address indicates the location of the Web service over HTTPS. -->
    <!-- <soap:address location="https://my-server:my-port/paybus"/> -->
    <!-- This soap:address indicates the location of the Web service over WebSphere MQSeries. Please replace "my-queue" with the appropriate queue name. -->
    <!-- <soap:address location="jms:/queue?destination=my-queue&amp;connectionFactory=()&amp;targetService=/paybus&amp;initialContextFactory=com.ibm.mq.jms.Nojndi" /> -->
  </port>
</service>
```
Introduction to SOAP

- Simple Object Access Protocol (SOAP) is an XML based protocol for the exchange of information in a distributed environment.
- A SOAP message is encoded as an XML document, consisting of an `<Envelope>` element, which contains:
  - The SOAP `<Header>` is an optional element within the SOAP message, and is used to pass information in SOAP messages that is not application payload. The SOAP header allows features to be added to a SOAP message. SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory.
  - The SOAP `<body>`, a mandatory element, contains information intended for the ultimate recipient of the message. In CICS terms, this will become the COMMAREA for the application program.
  - The SOAP `<fault>`, contained within the `<body>`, is used for reporting errors.
SOAP example (1 of 2)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:pay="http://www.PAYBUS.PAYCOM.Request.com"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <wsa:Action>PAYBUS</wsa:Action>
        <wsa:MessageID>uuid:31f43668-eddf-40fe-9453-7777bb95df14</wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsa:To>http://ctrek.dal-ebis.ihost.com:6000/paybus</wsa:To>
        <wsse:Security>
            <wsu:Timestamp wsu:Id="Timestamp-9f1d63d6-b276-44d1-8c9f-68e24828d30a">
                <wsu:Created>2012-02-29T16:41:50Z</wsu:Created>
                <wsu:Expires>2012-02-29T16:46:50Z</wsu:Expires>
            </wsu:Timestamp>
        </wsse:Security>
    </soap:Header>
...
SOAP example (2 of 2)

```xml
<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <PAYBUSOperation xmlns="http://www.PAYBUS.PAYCOM.Request.com">
    <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>
  </PAYBUSOperation>
</soap:Body>
</soap:Envelope>
```
SOAP fault example

<soap:Fault>
  <faultcode>soap:Server</faultcode>
  <faultstring>Server was unable to process request. --&gt; Something bad happened</faultstring>
  <detail />
</soap:Fault>
Approach to web services development (1 of 2)

Provider – means that CICS is the host of the web service which is being invoked from elsewhere.

Top down – Start with a WSDL document and convert it into a form usable as a CICS web service.

Web service requester

CICS as provider

CICS

CICS as requester

Web service provider

Bottom up – Start with a language structure and convert it into a form usable as a CICS web service

Requester – means that CICS is the user of a web service provided elsewhere

WSDL

Bottom up

Language structures
Approach to web services development
(2 of 2)

Bottom-up

Existing business application (COBOL, C, C++, PLI)

New service: WSDL

Generate

Top-down

Existing service description WSDL

Generate

Meet in the middle

Existing service description WSDL

Map and Generate

New business application (COBOL, C, C++, PLI)

Existing business application (COBOL, C, C++, PLI)
CICS web services assistant

• Web services assistant is a set of batch utilities which can help you to transform existing CICS applications into web services and to enable CICS applications to use web services provided by external providers.

• The CICS web services assistant comprises two utility programs:
  • DFHLS2WS generates a web service binding file from a language structure. This utility also generates a web service description.
  • DFHWS2LS generates a web service binding file from a web service description. This utility also generates a language structure that you can use in your application programs.
Rational Developer for System z

- Web Services and XML development
  - Offers the ability to create, view, edit and validate WSDL, document-type definitions (DTD) and XML schemas, transforms XML documents into text, HTML, or other XML document types.
  - Integrates relational databases and XML.
  - Generates COBOL adapters and CICS TS V3 WSBind and COBOL artifacts for converting between Web Service Definition Language.
- The CICS service flow feature provides components that extend CICS Transaction Server by providing adapters that exploit CICS interfaces to invoke the CICS terminal-oriented transactions and COMMAREA programs required by the web service generated from the service flow project.
CICS Service Flow Modeler

• At build time, a developer uses the Service Flow Modeler for the following tasks:
  • Model a newly composed business service or flow using processes or services and their interfaces.
  • Capture existing enterprise information system (EIS) interfaces, such as screens or communication areas.
  • Generate adapters to and from interfaces, supporting both the accumulation of information that is used in request and response processing, as well as data transformation activities between the flow and the interface.
  •Expose business flows as a service.
• At run time, a service requester invokes the deployed adapter service to perform a business function as modeled in the CICS Service Flow Modeler.
Web services: Pipeline

• What is a CICS pipeline?
  • Responsible for dealing with SOAP headers
  • Implemented as a series of programs
  • Can be configured by end user by using message handlers

• A message handler is a program in which you can perform your own processing of web service requests and responses.

• A pipeline is a set of message handlers that are executed in sequence.
• **CONFIGFILE**(name) specifies the name of a z/OS UNIX file that contains information about the processing nodes that will act on a service request, and on the response.

• **SHELF({/var/cicsts/|directory})** specifies the 1–255 character fully-qualified name of a directory (a *shelf*, primarily for web service binding files) on z/OS UNIX.

• **WSDIR(directory)** specifies the 1–255 character fully-qualified name of the *web service binding directory* (also known as the *pickup directory*) on z/OS UNIX.
Pipeline configuration file

• The pipeline configuration file, named in a PIPELINE resource definition, is used to describe the series of message handlers to process the request.

• The configuration file is an XML document, stored in HFS and can be edited with any XML editor.

• The configuration file will contain mandatory `<service>` and optional `<transport>` elements along with application handler `<apphandler>` and a service parameter list `<service_parameter_list>`.
Pipeline XML configuration for a service provider

```xml
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```
WEBSERVICE resource definition

- The WEBSERVICE resource defines aspects of the run time environment for a CICS application program deployed in a web services setting.

- The aspects of the run time environment that are defined by the WEBSERVICE resource definition are:
  - Pipeline resource definition: Defines the set of message handlers that operate on Web Service requests and responses.
  - Web Service binding file: A file which contains information that CICS uses to map data between input and output messages, and application data structures.
  - Web Service description: Used only when runtime validation of SOAP messages is required.
Web services: URIMAP resource definition

- The URIMAP definition is used to match a URI to a WEBSERVICE definition and a PIPELINE definition.
- You should have a unique URI for each web service that you want to use in CICS.
- The parameters that apply to a web service form of the URIMAP are:
  - USAGE (PIPELINE) parameter indicates that the URIMAP definition is applicable to a web service and that the PIPELINE and WEBSERVICE parameters must be specified.
  - The PIPELINE parameter names an installed PIPELINE resource which will be used to determine the processing nodes or message handlers that will be invoked for this web service request.
  - The WEBSERVICE parameter names an installed WEBSERVICE resource that defines the execution environment that lets a CICS application program operate as a web service provider or requester.

<table>
<thead>
<tr>
<th>URIMAP</th>
<th>Description</th>
<th>Status</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group</td>
<td></td>
<td>Pipeline</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Usage</td>
<td>Pipeline</td>
<td></td>
</tr>
</tbody>
</table>

**UNIVERSAL RESOURCE IDENTIFIER**

- Scheme: HTTP
- Port: No
- Host: (Mixed Case)
- Path: (Mixed Case)

**ASSOCIATED CICS RESOURCES**

- Tcipservice
- Analyzer: No
- Converter
- Transaction
- Program
- Pipeline
- Webservice
Web services: WEBSERVICE resource definition

```plaintext
>>>WEBSERVICE (name)--GROUP (groupname)-----------------+-
    '-DESCRIPTION(text)-'

>---PIPELINE (pipelinename)--WSBIND (hfsfile)------------------>
   .-VALIDATION (NO) --.

>-----------------------------------+------------------>
   '-VALIDATION (YES)-'  '-WSDLFILE (hfsfile)-'
```

- **PIPELINE(pipelinename)** specifies the 1-8 character name of the PIPELINE with which this WEBSERVICE is associated.
- **VALIDATION(NO|YES)** specifies whether full validation of SOAP messages against the corresponding schema in the web service description should be performed at run time.
- **WSBIND(hfsfile)** specifies the 1-255 character fully-qualified file name of the Web Service binding file on z/OS UNIX.
- **WSDLFILE(hfsfile)** specifies the 1-255 character fully-qualified file name of the Web Services Description Language (WSDL) file on z/OS UNIX. This file is used when full runtime validation is active.
Web service resource interrelationships

- HFS
  - config
-Batch Tools
  - BINDING=
  - URI=
  - PGMNAME=
  - PGMINT=
- WSBind
  - WSDL
- URIMAP
  - USAGE(PIPELINE)
  - HOST
  - PATH
- PIPELINE
  - CONFIGFILE
  - SHELF
  - WSDIR
- WEBSERVICE
  - PIPELINE
  - WSBIND
  - WSDLFILE
- dynamic install

CICS
CICS TS runtime support and specifications

- Runtime support for:
  - WSDL V1.1
  - SOAP V1.1 and SOAP V1.2
  - WS-I Basic Profile V1.1
  - XML V1.0
  - WS-I Simple SOAP Binding Profile V1.0

- Specifications:
  - WSDL V1.1 specification: [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)
  - SOAP V1.2 specification: [http://www.w3.org/TR/soap12-part0/](http://www.w3.org/TR/soap12-part0/)
Summary

- Web services in CICS builds on established CICS concepts and technologies such as pseudo-conversational transactions and program-to-program communication via COMMAREAs (or channels and containers)
- Web services introduce new concepts and technologies such as XML, SOAP, and WSDL
- Web services involve a combination of CICS resource definitions (PIPELINE, WEBSERVICE, URIMAP)
- CICS TS provides utilities for converting existing CICS applications into web services
- IBM provides other tools to help develop web services