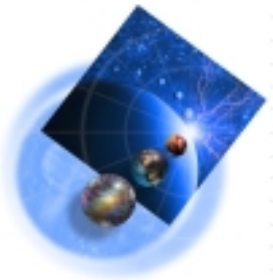
XML, COBOL and Application Modernization

Bringing XML to the application

Tom Ross
Nick Tindall
Share Tampa
February, 2007



Agenda

- **Some XML terminology**
- **Scenarios**
- **XML support in COBOL**
- **WebSphere Developer – XSE**
  - XML Services for the Enterprise
- **Discussion, questions**

Agenda...

- **Some XML terminology**

Some XML terminology

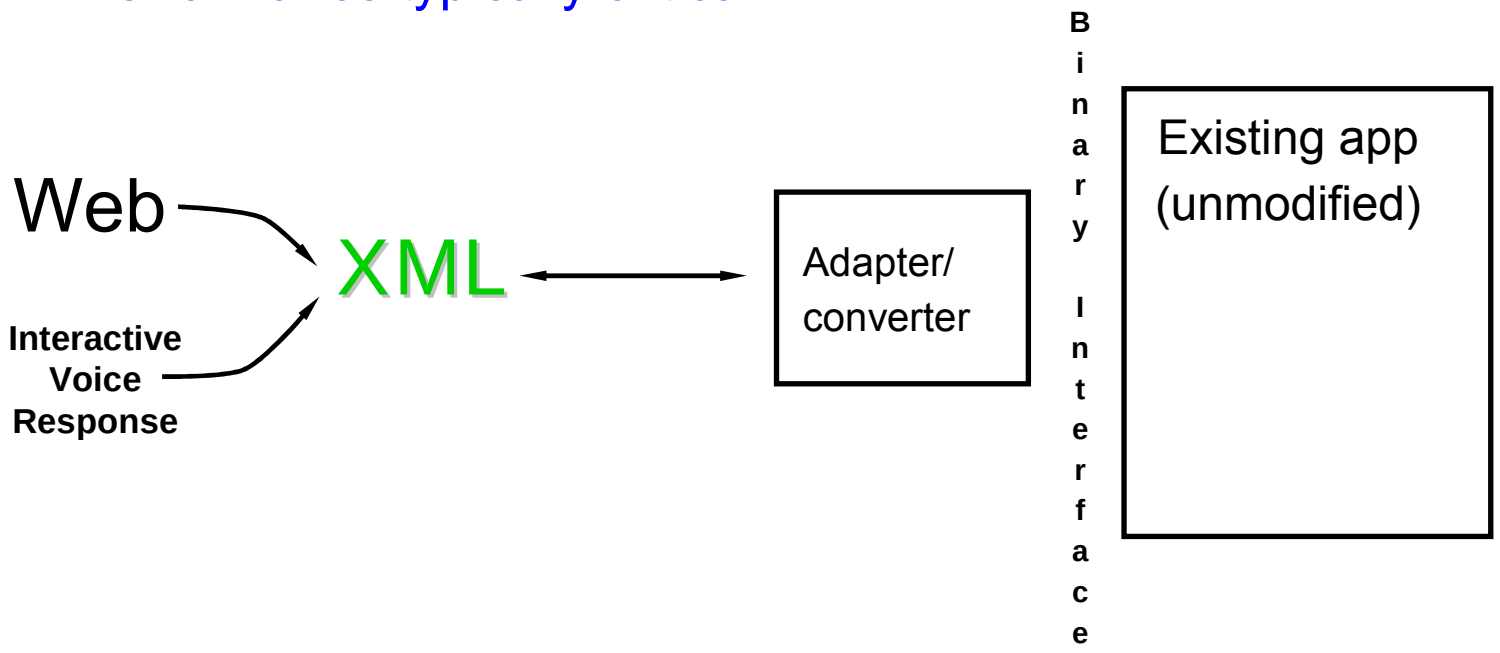
- **XML processing programs ("parsers"):**
  - SAX - Simple API for XML
    - ► Callbacks to your program for each XML "event" (document fragment)
  - DOM - Document Object Model
    - ► In effect the XML document becomes a miniature data base that supports navigation and simple queries
- **Well-formed XML document - basic grammar rules:**
  - exactly one "root" element
  - a matching end tag for every start tag
  - elements properly nested, etc., etc.
- **Valid XML document (wrt a DTD or schema):**
  - Well-formedness, plus...
  - Element order and containment (structure constraints)
  - Proper content (type constraints)

Agenda...

- Some XML terminology
- **Scenarios**

Customer scenarios/problems - 1

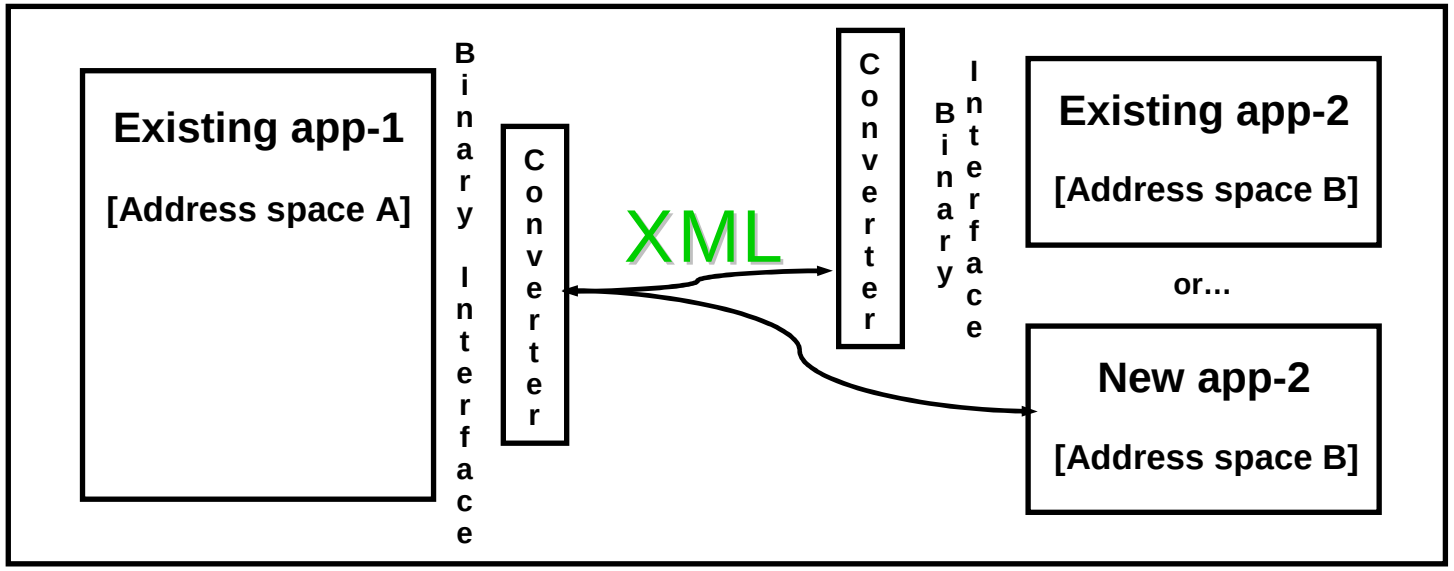
- **Refacing existing applications/transactions**
  - To allow non-traditional clients to use them
  - A step towards componentization
  - In particular to allow access as a Web Service
  - Various transports/protocols
  - Performance typically critical



Customer scenarios/problems - 2

- **Inter-application communication**
  - Need arises because of mergers/acquisitions, or
  - Between different purchased applications
  - Interfaces need only be compatible, not identical
  - Allows schema evolution without synchronized updates
  - Would previously have been done with file readers/writers

Single Enterprise system/complex



Agenda...

- Some XML terminology
- Scenarios
- **XML support in COBOL**

Why process XML in applications?

- **Coherent development context and methodology**
- **Centralizes business logic within the application**
  - Versus some business function here, some there, ...
- **Independent of middleware choices, characteristics**
- **Allows business logic to be conveniently applied during and after message acquisition/generation**
- **Incremental step from existing application design**
- **Can process XML messages as such**
  - Versus forcing conversion to traditional data structures

Why process XML in COBOL?

- **Keeps development control in one place/style**
- **Guarantees correct language semantics**
  - sign configuration
  - layout/padding
  - picture constraints
- **Exploits your existing assets/skills/literacy**
- **Supports a variety of scenarios:**
  - Converters, bridging to existing (unchanged) applications
  - Direct use of XML in new or enhanced applications
- **High performance: CPU and elapsed time**

Enterprise COBOL XML support

- **Inbound XML parsing and outbound XML generation:**
  - XML PARSE xml-doc PROCESSING PROCEDURE xml-handler
  - XML GENERATE xml-doc FROM data-item COUNT IN char-count
- **XML parsing**
  - Much faster than general purpose parsers
    - ► Designed for high-speed transaction processing
  - Runs in all COBOL run-time environments:
    - ► CICS, IMS, batch, TSO, USS, ...
  - Works with any transport mechanism for XML documents
    - ► Use MQSeries, CICS transient queue or COMMAREA, IMS message processing queue, WebSphere, etc.
  - Provides basic SAX-style parsing
  - Checks well-formedness (but not validity)
  - XML Parser is part of the run-time library
    - ► Can be used from Enterprise COBOL or Enterprise PL/I
- **XML generation:**
  - Single statement transforms entire data structure (group)

XML PARSE

- **Parses XML documents that are in memory, in a COBOL alphanumeric or national data item**
- **Discovers the individual pieces of XML documents**
  - Passes each piece to user-written processing procedure
- **During parsing you can populate COBOL data structures with the data from XML messages**
  - Advantage: non-COBOL programs can communicate data to/from COBOL without having to know the COBOL data structure formats!

XML PARSE…

- **XML PARSE statement**
  - The COBOL interface to the high-speed XML parser
- **XML special registers**
  - XML-CODE: communicates status of parsing
  - XML-EVENT: describes each event during parsing
  - XML-TEXT: contains XML document fragments
  - XML-NTEXT: contains NATIONAL XML doc fragments

```
   XML PARSE XMLDOCUMENT
     PROCESSING PROCEDURE XMLEVENT-HANDLER
   END-XML
   ...
XMLEVENT-HANDLER.
   EVALUATE TRUE
     WHEN XML-EVENT = 'START-OF-ELEMENT' AND
          XML-TEXT = 'TRADE'
       DISPLAY 'Processing new stock trade'
   ...
```

Hello XML World — inbound

```
Identification division.
  Program-id. HelloXML.
Data division.
 Working-storage section.
  1 M.
   2 pic x(21) value '<?xml version="1.0"?>'.
   2 pic x(40) value '<msg type="succinct">Hello, World!</msg>'.
Procedure division.
    Display 'XML Event              XML Text'
    XML Parse M
      Processing procedure P
    End-XML
    Goback.
  P.
    If XML-Code = 0
      Display XML-Event XML-Text
    End-if.
End program HelloXML.
```

```
XML Event             XML Text
START-OF-DOCUMENT     <?xml version="1.0"?><msg type="succinct">Hello, World!</msg>
VERSION-INFORMATION   1.0
START-OF-ELEMENT      msg
ATTRIBUTE-NAME        type
ATTRIBUTE-CHARACTERS  succinct
CONTENT-CHARACTERS    Hello, World!
END-OF-ELEMENT        msg
END-OF-DOCUMENT
```

XML GENERATE … FROM

- **Generates a complete XML document:**
  - Into an alphanumeric or national data item
  - From a group or elementary item
- **Data values are "trimmed:"**
  - Trailing spaces removed from alphanumeric values (leading spaces from right-justified items)
  - Leading zeroes removed from numeric values
- **XML tag names are the mixed-case data names**
- **Supports all data types except -pointer and object reference**
- **Unnamed and redefining items are ignored**
- **For more details see V3.4 Programming Guide:**
  - www.ibm.com/software/awdtools/cobol/zos/library/

Hello XML World — outbound

```
Identification division.
  Program-id. HelloXML.
Data division.
 Working-storage section.
  1 Hello-doc pic x(200).
  1 Greeting.
   2 msg pic x(100) value 'Hello, World!    '.
  1 Num-chars pic 999.
Procedure division.
    XML Generate Hello-doc From Greeting
        Count in Num-chars
    Display '|' Hello-doc(1:Num-chars) '|'
    Goback.
End program HelloXML.
```

```
|<Greeting><msg>Hello, World!</msg></Greeting>|
```

Hello XML World — outbound …

```
1 Hello-doc pic x(200).
1 Greeting-Grp.
 2 Contact-info.
   3 Name      PIC x(20) Value 'Tom'.
   3 Addr      PIC x(18) Value '555 Bailey Ave'.
   3 Telephone PIC 9(12) Value 4084634242.
 2 Contact-redef REDEFINES Contact-info.
   3 Junk PIC X(50).
 2 msg pic x(100) value 'Hello, World!    '.
1 Greeting-redef REDEFINES Greeting-Grp.
 2 Name         PIC x(20).
 2              PIC x(18).
 2 Phone        PIC 9(12).
 2 Short-Msg    PIC X(10).
1 Num-chars binary pic 9(9).
```

Hello XML World — outbound …

XML Generate Hello-doc From Greeting-grp Count in Num-chars

```
<Greeting-grp>
  <Contact-info>
   <Name>Tom</Name>
   <Addr>555 Bailey Ave</Addr>
   <Telephone>4084634242</Telephone>
   <msg>Hello, World!</msg>
  </Contact-info>
</Greeting-grp>
```

XML Generate Hello-doc From Greeting-redef Count in Num-chars

```
<Greeting-redef>
  <Name>Tom</Name>
  <Phone>4084634242</Phone>
  <Short-Msg>Hello, World!</Short-Msg>
</Greeting-redef>
```

Agenda...

- Some XML terminology
- Scenarios
- XML support in COBOL
- **WebSphere Developer for zSeries (WD for z)**
  XML Services for the Enterprise (XSE)

What is WD XSE?

- **Allows COBOL applications to consume and produce XML messages**
- **Leverages XML parsing capabilities of IBM Enterprise COBOL V3**
- **Creates:**
  - *Inbound* converter program, to convert XML messages into native COBOL data
  - *Outbound* converter program, to convert native COBOL data into XML messages
  - Sample COBOL driver program:
    - ► Illustrates the invocation of converters
    - ► Illustrates the interaction with existing application
    - ► Needs to be modified before use
  - XML Schema, for validation and generating samples
    - ► Input to the Web Service Description (WSDL)
- **Enables communication with XML-based systems**

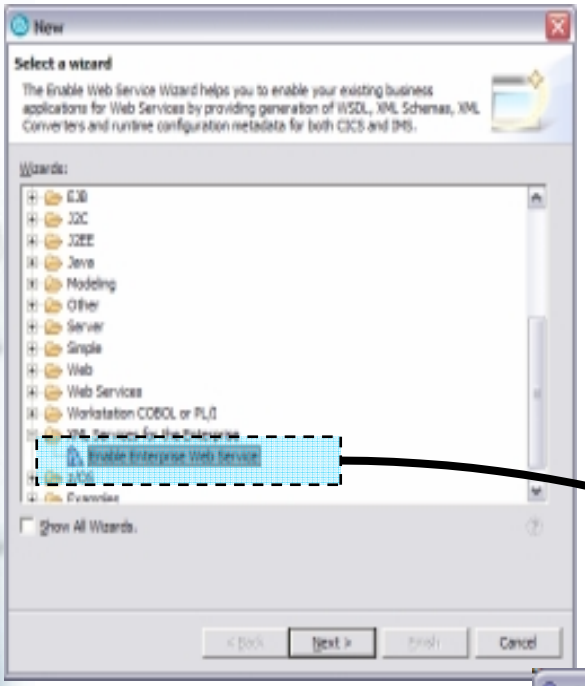
WD XSE

- **Web Service Runtime and Scenario Selection** dialog
- **Create New Service Interface (bottom-up**) wizard
  - Compiled XML conversion type
    - Language structures page
    - Generation options page
    - BIDI conversion options dialog
    - Runtime specific pages
      - Web Services for CICS page
      - IMS SOAP Access page
    - File, data set or member selection
  - Interpretive XML conversion type (bottom-up)
    - CICS Web Services Assistant pages (bottom-up)
- **Create New Service Implementation (top-down)** wizard
  - Interpretive XML conversion type (top-down)
    - CICS Web Services Assistant pages (top-down)
- **Map to an Existing Service Interface (meet-in-the-middle)**
  - **Mapping Converter Generator** wizard
- **Generating artifacts remotely**

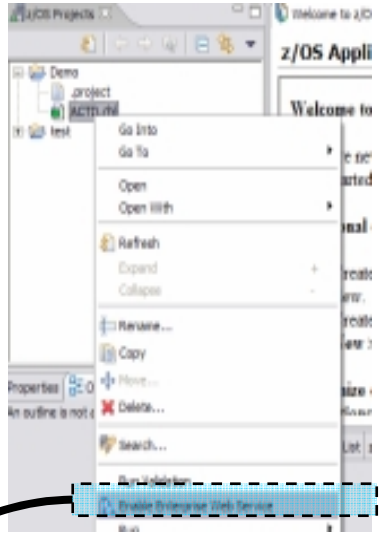
WD XSE

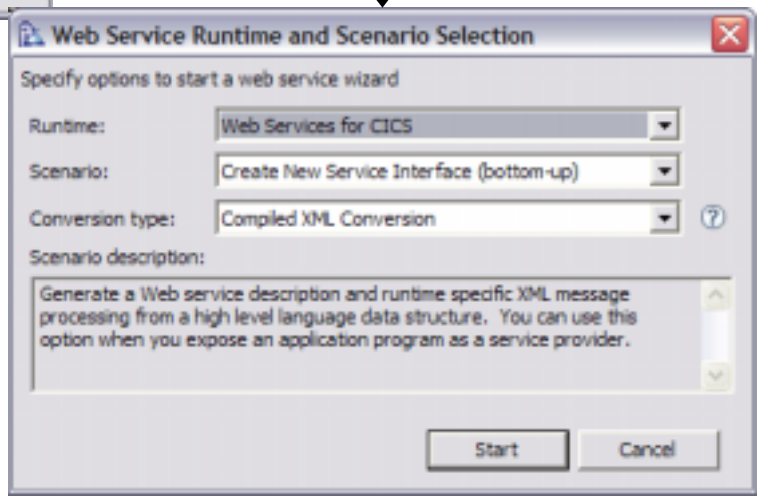
Web Service Runtime and Scenario Selection dialog

Select artifact then File->New->Other…

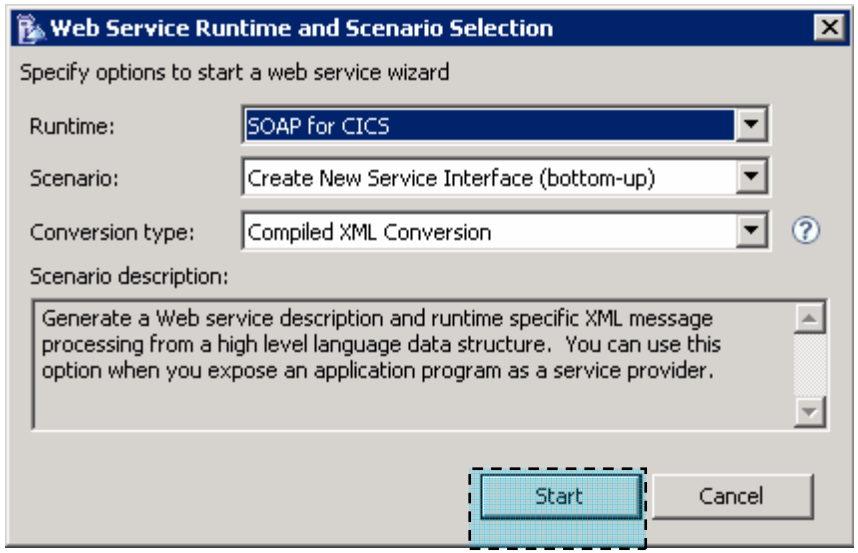


OR Enable Enterprise Web service from pop-up menu



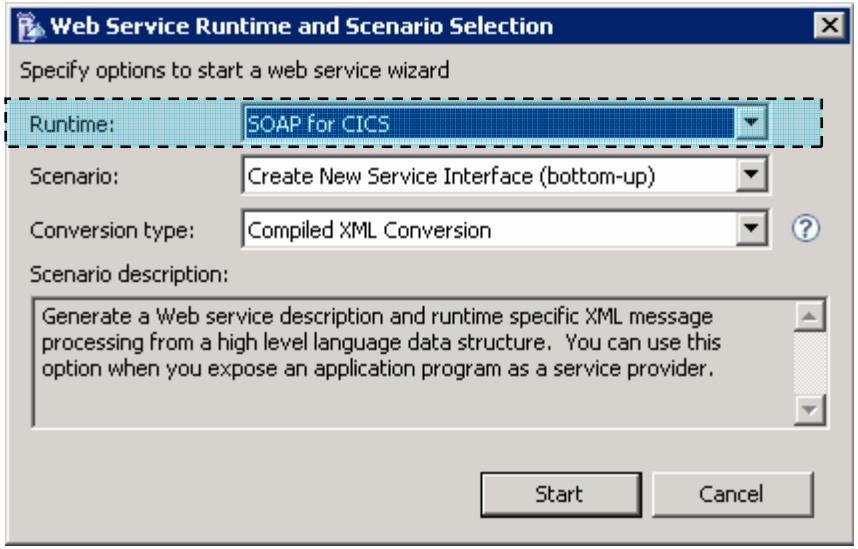


Web Service Runtime and Scenario Selection *dialog*



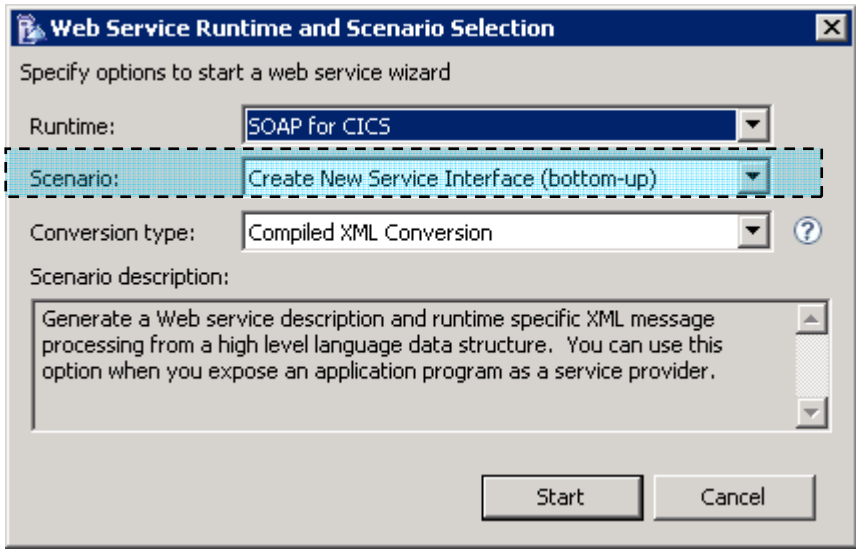
Start the appropriate XSE wizard for selected combination of target runtime, web service development scenario and XML conversion technology type.

Web Service Runtime and Scenario Selection *dialog*



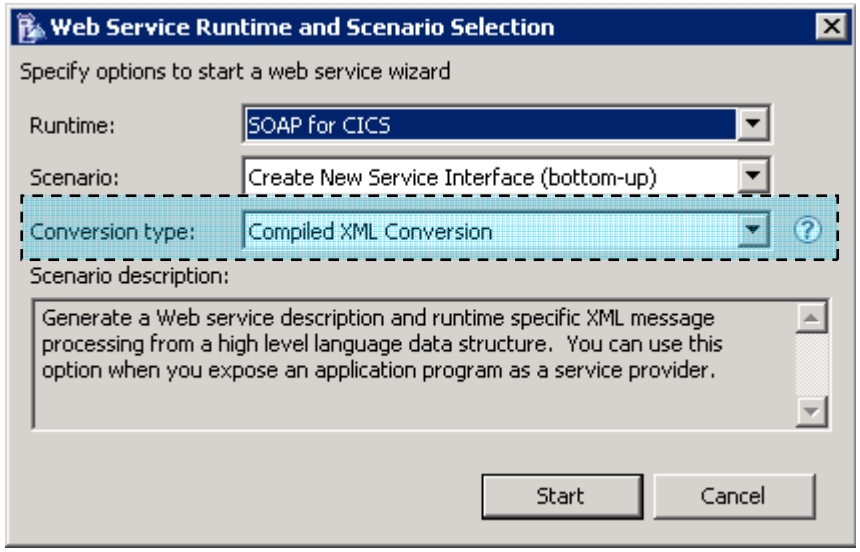
Supported Runtimes
Web Services for CICS (CICS TS 3.1)
SOAP for CICS (CICS TS 2.2/2.3/3.1)
IMS SOAP Gateway
Batch, TSO and USS

Web Service Runtime and Scenario Selection *dialog*



Scenario name	Scenario nick-name
Create New Service Interface	**Bottom-up**
Map an Existing Service Interface	**Meet-in-middle**
Create New Service Implementation	**Top-down**

Web Service Runtime and Scenario Selection *dialog*

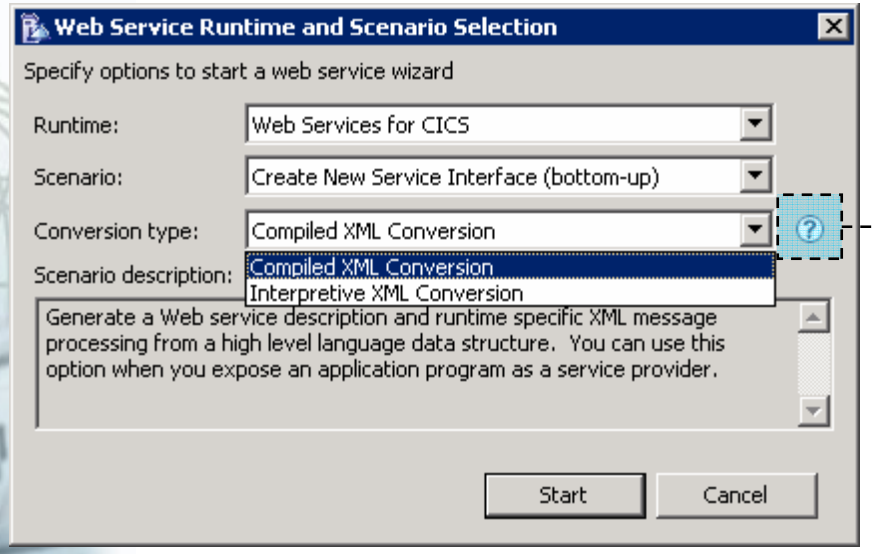


Supported Conversion Types

Compiled XML Conversion

Interpretive XML Conversion (CICS TS 3.1)

Web Service Runtime and Scenario Selection *dialog*



Help on the XML Conversion technology type is available. Interpretive is *only* available for Web Services for CICS runtime.

Compiled XML Conversion:

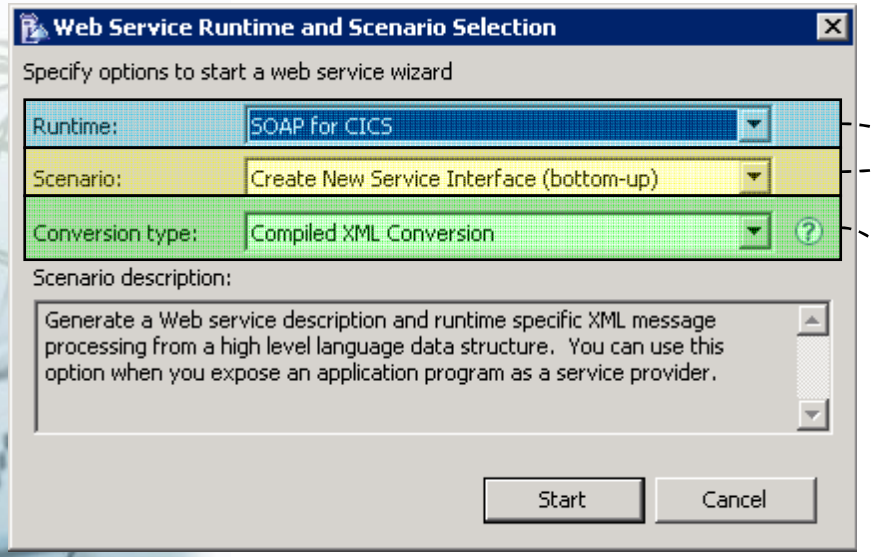
XML conversion is accomplished using a suite of generated high-level language (HLL) programs. A driver program interacts with the runtime and invokes bundled programs to provide conversion of XML to and from language structures. The Compiled XML Conversion type provides more extensive support for language structure data types and constructs than the Interpetive XML Conversion type, however compilation of XML Converters is required and there are more artifacts to manage.

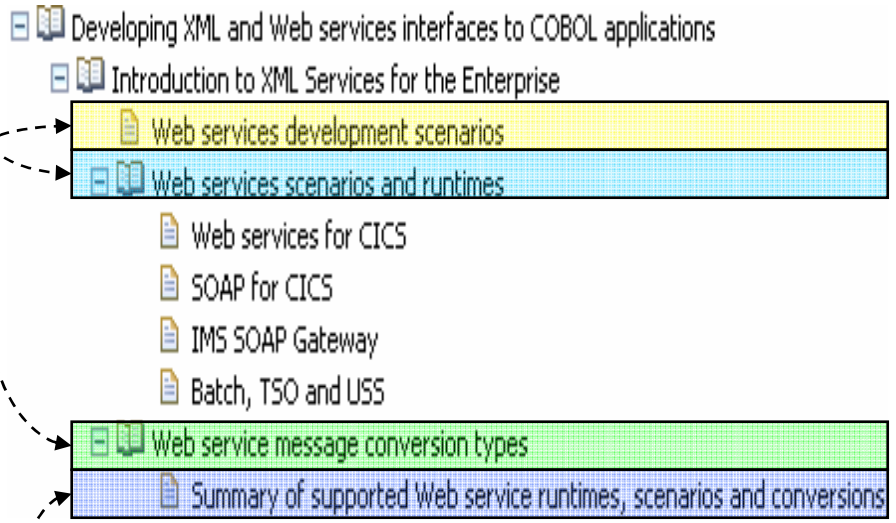
Interpretive XML Conversion:

XML conversion is accomplished using generated metadata and an XML conversion component built into the runtime. While the Interpretive XML Conversion type has limited support for language structure data types and constructs its requires fewer artifacts and does not involve compilation of XML Converters.

Introduction to XML Services for the Enterprise

Web Service Runtime and Scenario Selection *dialog*





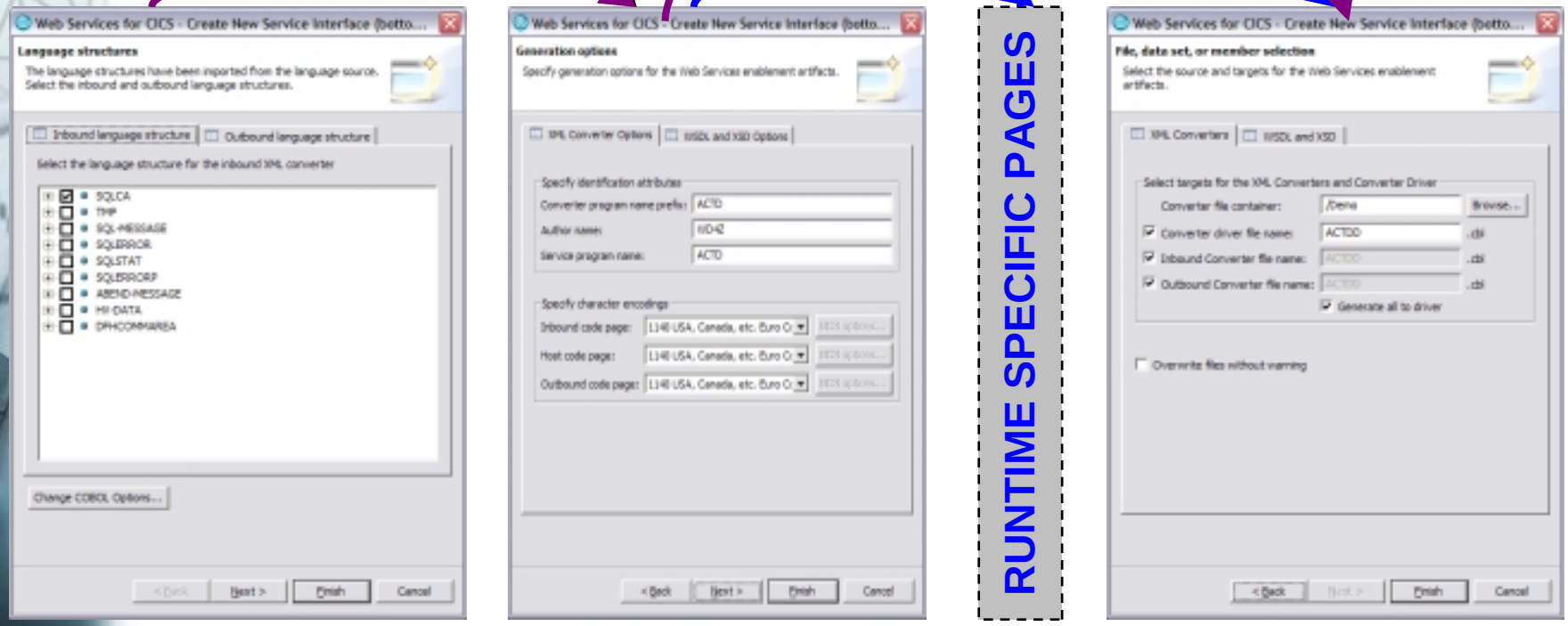
To learn about the Runtimes, Scenarios and Conversion types, refer to the WD/z online help. Note that not all possible combinations are supported. The WD/z online help has a table of the supported combinations.

Create New Service Interface (bottom-up) wizard

- Compiled XML conversion type
  - **Common pages**

- **SOAP for CICS**
- **Batch, TSO, and USS**

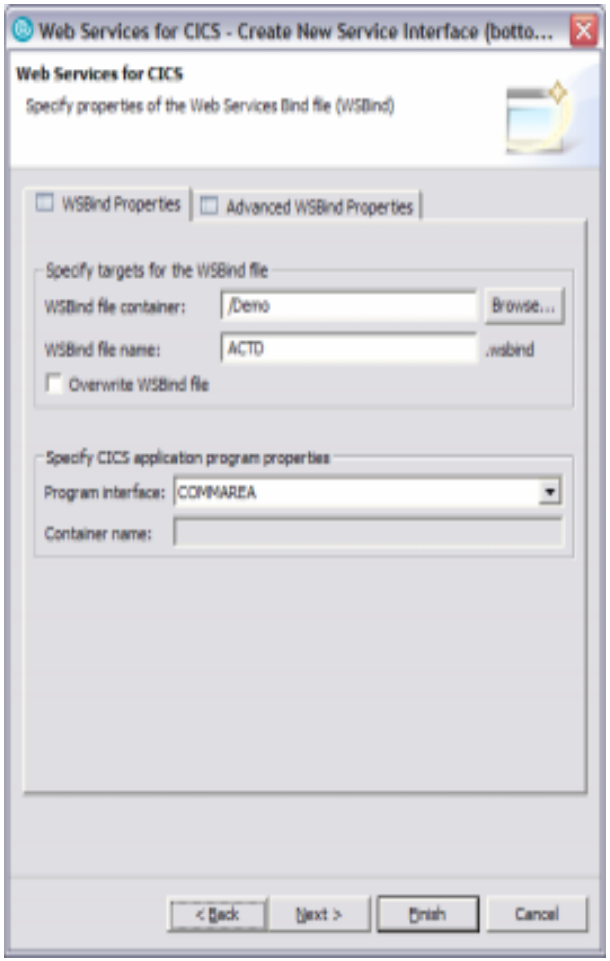
- **Web services for CICS**
- **IMS SOAP Gateway**



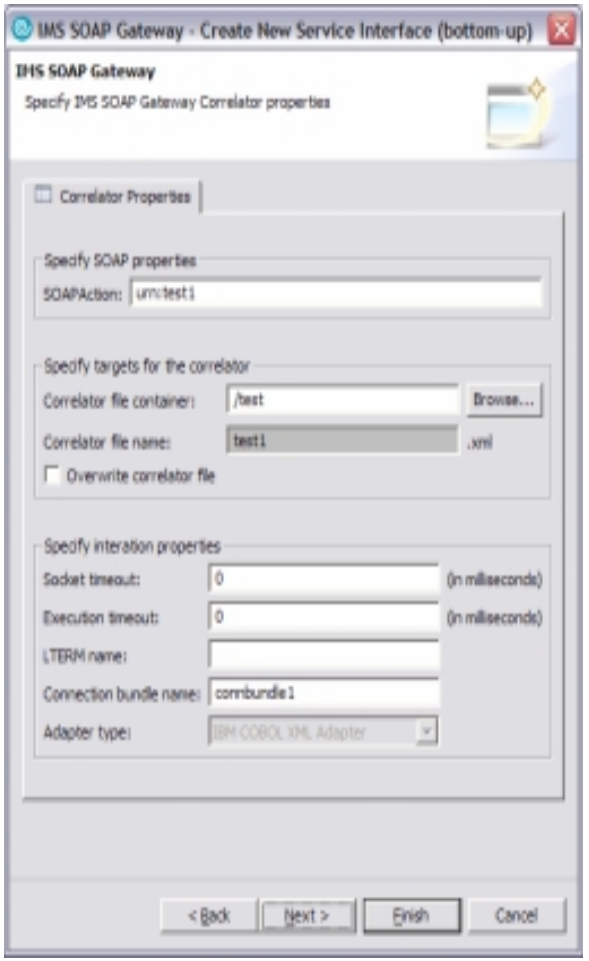
Create New Service Interface (bottom-up) wizard

- Compiled XML conversion type
  - **Runtime specific pages**

- **Web services for CICS**

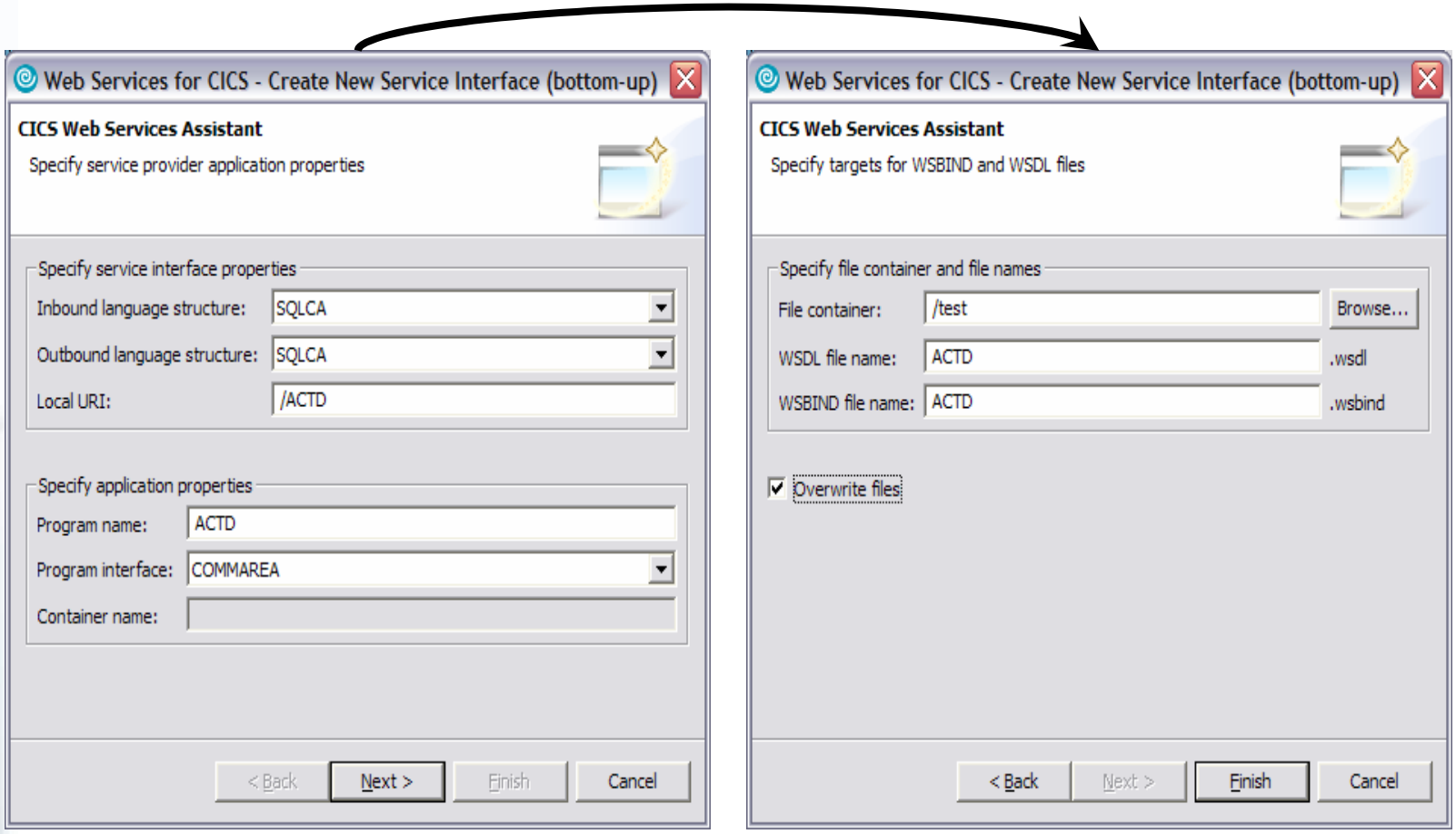


- **IMS SOAP Gateway**



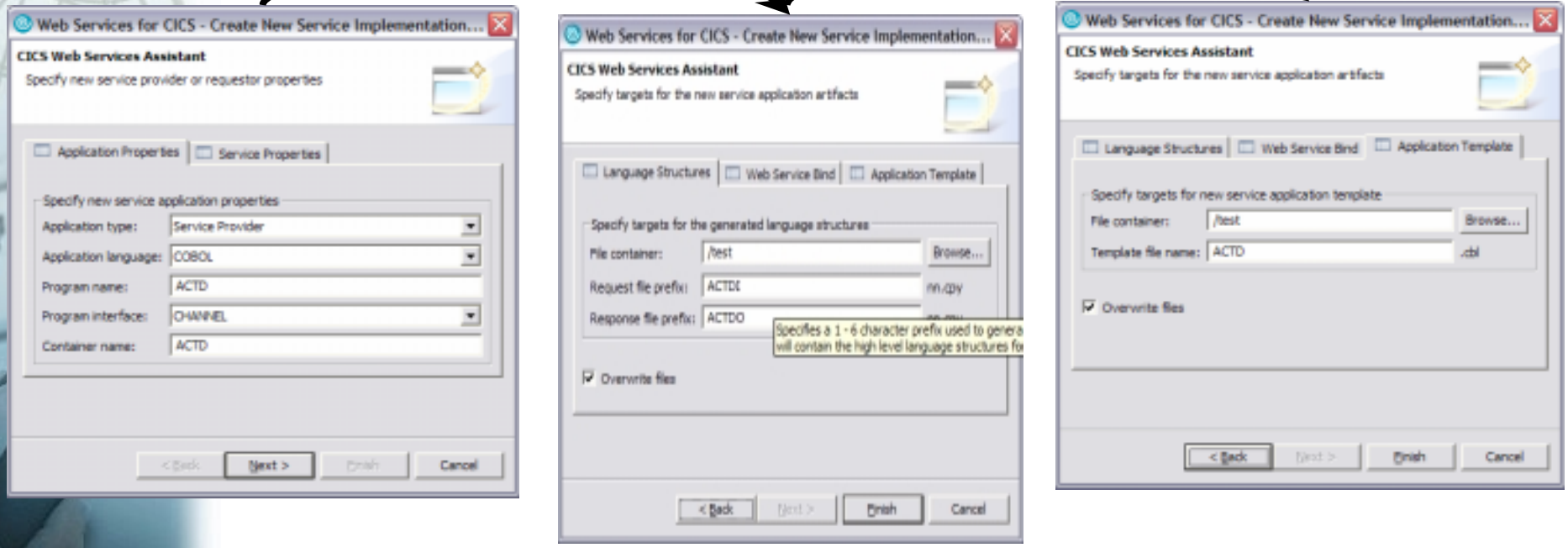
Create New Service Interface (bottom-up) wizard

- **Interpretive XML conversion type**
  - **CICS Web Services Assistant pages (bottom up)**



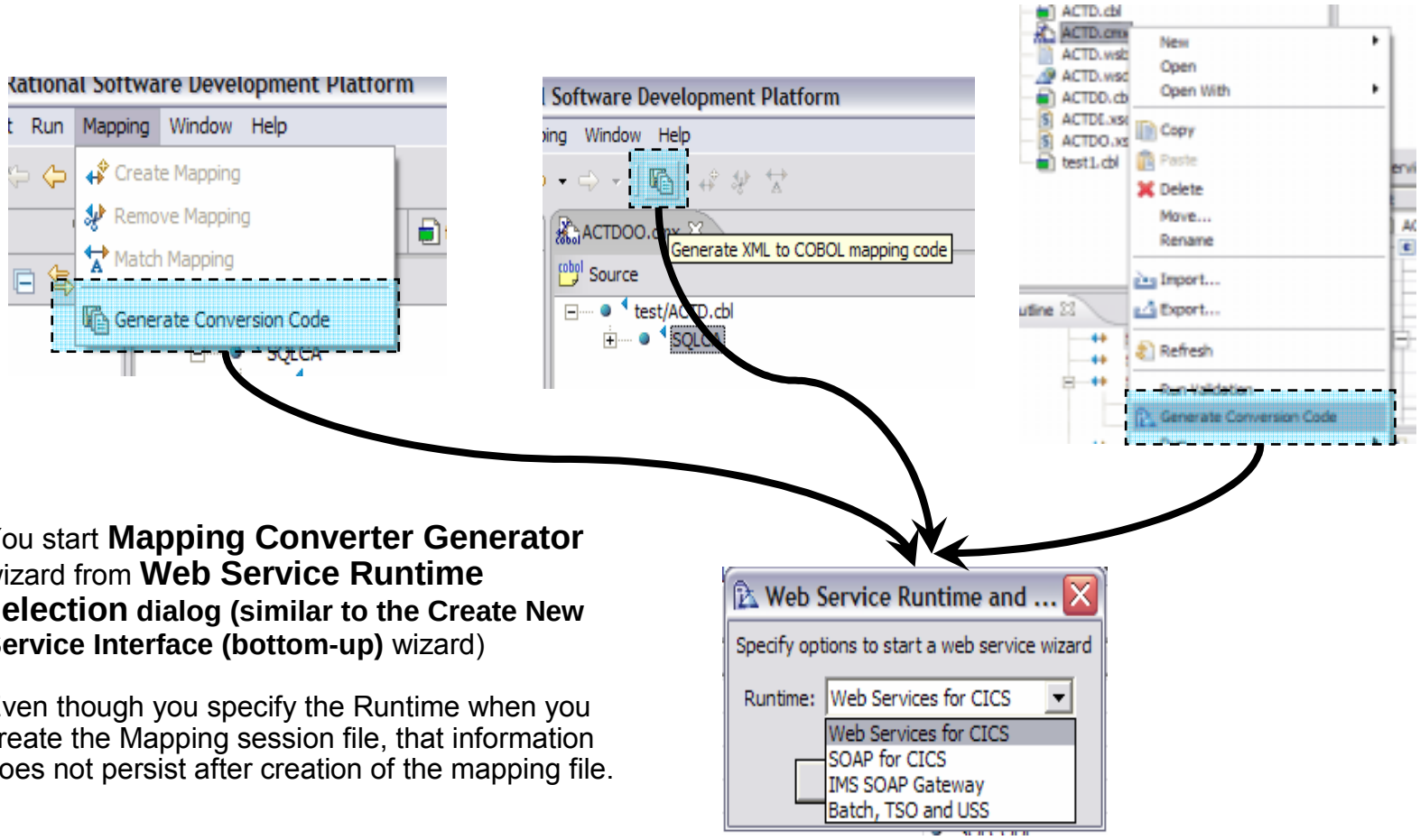
Create New Service Implementation (top-down) **wizard**

- **Interpretive XML conversion type**
  - **CICS Web Services Assistant pages (top-down)**



Map to an Existing Service Interface (meet-in-the-middle)

Generate Conversion Code for mapping

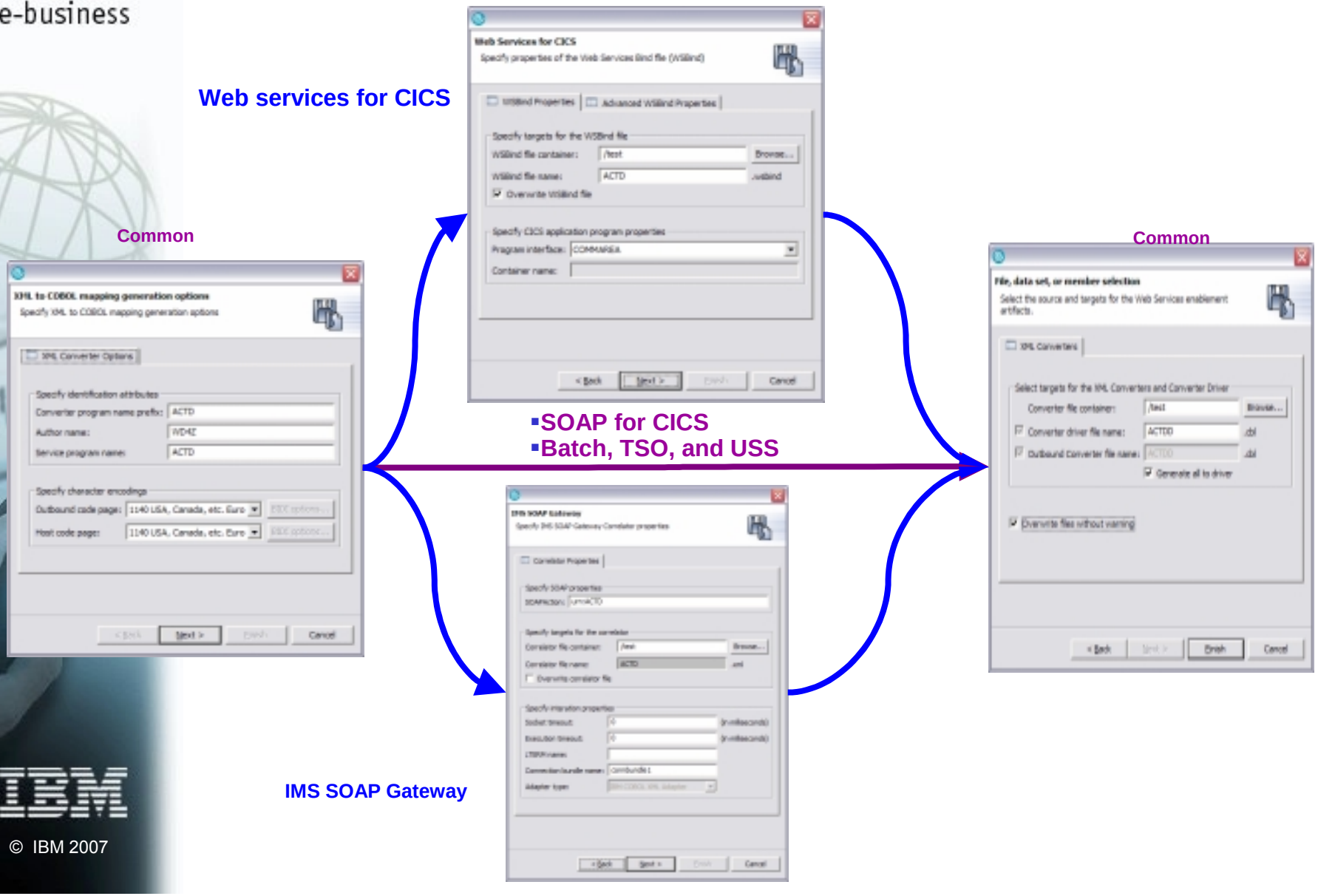


You start **Mapping Converter Generator** wizard from **Web Service Runtime selection dialog (similar to the Create New Service Interface (bottom-up)** wizard)

Even though you specify the Runtime when you create the Mapping session file, that information does not persist after creation of the mapping file.

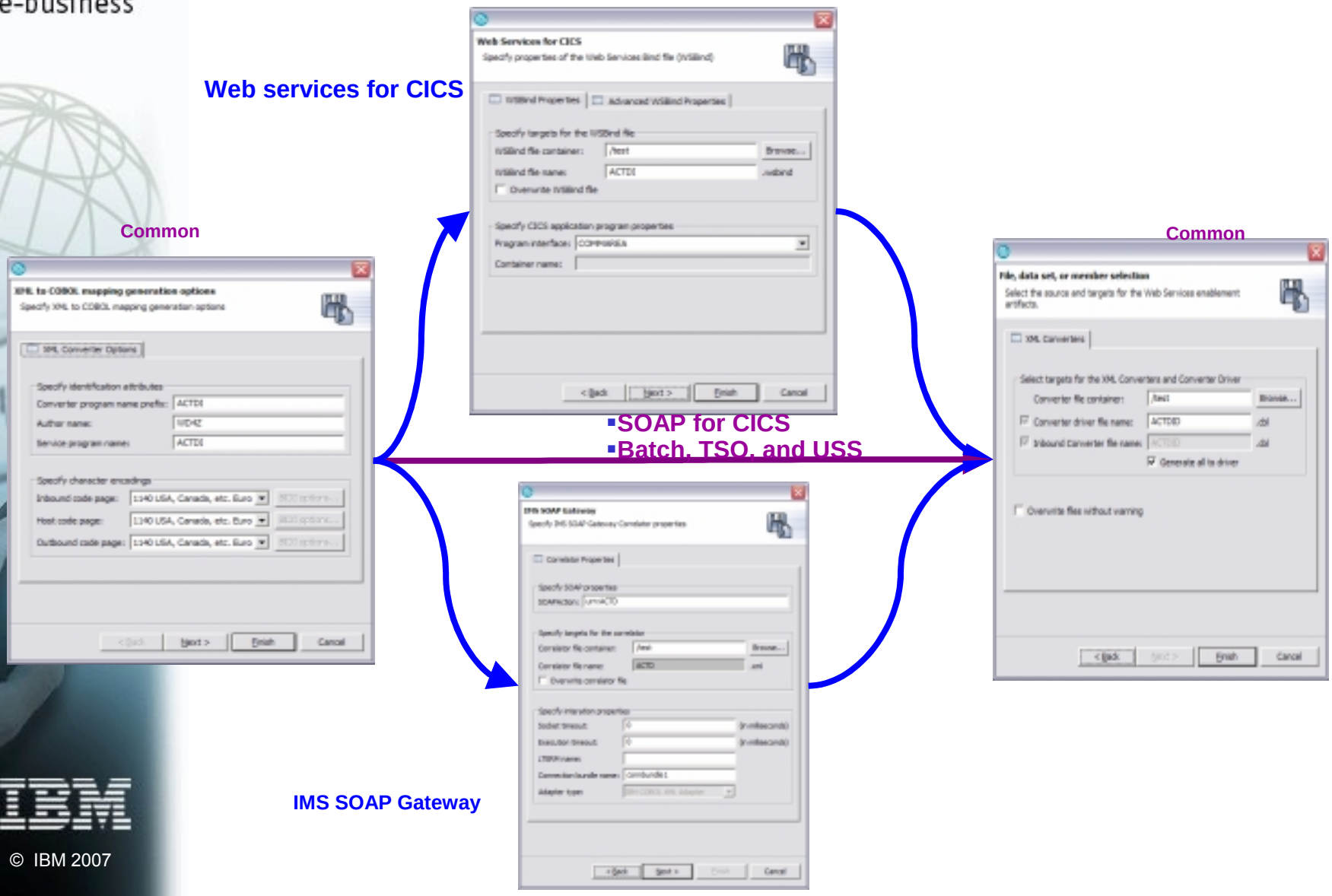
Map to an Existing Service Interface (meet-in-the-middle)

Mapping Converter Generator wizard (Outbound mapping)



Map to an Existing Service Interface (meet-in-the-middle)

Mapping Converter Generator wizard (Inbound mapping)



Example XML and COBOL data structure

```
<?xml version="1.0"?>
<usedcar>
  <price>9,347.99</price>
  <make>AMC</make>
  <model>Gremlin</model>
  <vin>GV39JFGLKM09Y</vin>
  <color>Red</color>
  <mileage>39,000</mileage>
  <inscode>I</inscode>
  <numclaims>1</numclaims>
  <claim>
    <claimno>S8430M4D20030226</claimno>
    <claimamt>1,234</claimamt>
    <insurer>IndemnitiesAreWe</insurer>
    <details>Engine fell out</details>
  </claim>
</usedcar>
```

```
01 USEDCAR.
  05 PRICE COMP-3 PIC 9(5)V99.
  05 SUMMARY.
    10 MAKE PIC X(36).
    10 MODEL PIC X(44).
    10 VIN PIC X(13).
    10 COLOR PIC X(10).
      88 RED VALUE 'Red'.
      88 WHITE VALUE 'White'.
      88 BLUE VALUE 'Blue'.
  05 HISTORY.
    10 MILEAGE PIC 9(6).
    10 INSCODE PIC X.
    10 NUMCLAIMS BINARY PIC 9.
    10 CLAIMS.
      15 CLAIM OCCURS 0 TO 9 TIMES
           DEPENDING ON NUMCLAIMS.
        20 CLAIMNO PIC X(16).
        20 CLAIMAMT BINARY PIC 9(5).
        20 INSURER PIC X(39).
        20 DETAILS PIC X(100).
```

```
Used car information:
Price    Make      Model     VIN            Color  Mileage InsCode
NumClaims
0934799  AMC       Gremlin   GV39JFGLKM09Y  Red    039000  I       1
  ClaimNo           ClaimAmt  Insurer            Details
  S8430M4D20030226  01234     IndemnitiesAreWe   Engine fell out
```

Pros and cons: WD XSE versus direct COBOL coding

- **WD XSE—strengths:**
  - Interactive tool avoids intricate and laborious programming
    - ► Easy to “reface” existing COBOL applications to support XML
  - Wholesale conversion between documents and structures
    - ► inbound and outbound
  - Robust error recovery, using LE services
  - Highly optimized and sophisticated content processing
    - ► Uses the native z/OS high-speed parser
  - Unnecessary elements are conveniently ignored
  - Can derive XML definition from COBOL, or match independent XML and COBOL definitions
  - Generated schema can be used to validate messages and as input to the WSDL file

Pros and cons: WD XSE versus direct COBOL coding…

- **XML PARSE—strengths:**
  - Flexibility: all parts of an XML document are accessible:
    - ► Attributes, processing instructions, comments, etc.
  - Business logic can be conveniently applied during and after message acquisition/generation
  - XML definition can be independent of (any) data structure
  - Can “short-circuit” parsing early after required input is seen
- **XML GENERATE…FROM—strengths:**
  - Very simple to use
  - A single COBOL statement provides wholesale conversion from a data structure to a document
  - The generated XML precisely matches the data structure
  - Redefinition allows selective output, different tag names

Agenda...

- Some XML terminology
- Scenarios
- XML support in COBOL
- WebSphere Studio—XML Enablement
- **Discussion, questions**

Sample XML parse program...

```
Process flag(i,i)
Identification division.
  Program-id. xmldump.

Environment division.
 Input-output section.
  File-control.
    select xf assign fi file status fs.

Data division.
 File section.
  FD xf recording mode v
      record varying from 1 to 1024 characters depending on l.
  1 r.
   2 pic x occurs 1 to 1024 times depending on l.

 Working-storage section.
  1 ft.
   2 pic x(5) value ') SHR'.
   2 pic x value low-value.

 Local-storage section.
  1 fv.
   2 pic x(7) value 'FI=DSN('.
   2 fn pic x(150).
  1 in-len binary pic 999.
  1 st pic x(9).
  1 rc binary pic 9(9).
  1 l binary pic 9(5).
  1 p binary pic 9(5).
  1 fs pic 99.
  1 xml-document pic x(32767).
```


```
Procedure division.
 mainline section.
    perform get-doc
    perform until in-len = 0
      if p > 0
        xml parse xml-document(1:p)
            processing procedure xml-handler
          on exception
            display 'XML document error ' xml-code
          not on exception
            display 'XML document successfully parsed'
        end-xml
      end-if
      perform get-doc
    end-perform

    goback
    .

 get-fn section.
    display
        '  Enter XML document file name or SYSIN (null to end)'
    move space to fn
    move 0 to tally
    accept fn
    inspect function reverse(fn) tallying tally
        for leading space
    compute in-len = 150 - tally
    .
```


```
get-doc section.
   perform get-fn
   evaluate true
     when in-len = 0
       continue
     when fn(1:in-len) = 'SYSIN'
       display '  Enter XML document:'
       move spaces to xml-document(1:150)
       accept xml-document(1:150)
       move function lower-case(xml-document(1:150))
           to xml-document(1:150)
       move 0 to p
       inspect function reverse(xml-document(1:150)) tallying p
           for leading spaces
       compute p = 150 - p
     when other
       move ft to fn(in-len + 1:length of ft)
       call 'putenv' using by value address of fv returning rc
       if rc not = 0
         display 'putenv failed with rc = ' rc '.'
         stop run
       end-if
       open input xf
       if fs = 0
         read xf
       end-if
       move 1 to p
       perform until fs not = 0
         if p - 1 + length of r > length of xml-document
           display 'XML document is larger than the document '
               'buffer (' length of xml-document ' bytes).'
           move 13 to fs
         else
           string r delimited by size into xml-document
               with pointer p
           read xf
         end-if
       end-perform
       evaluate fs
         when 10
           subtract 1 from p
         when 13
           move 0 to p
         when other
           display 'Some catastrophe on file ' fn(1:in-len)
               '; status = ' fs '.'
           move 0 to p
       end-evaluate
       close xf
   end-evaluate
   .
```


```
 xml-handler section.
    evaluate xml-event
      when 'START-OF-DOCUMENT'
        compute rc = function length(xml-text)
        move rc to st
        call 'nzp' using st l
        display ' '
        display 'Start of document: length=' st(l:)
            ' characters.'
      when 'END-OF-DOCUMENT'
        display 'End of document.'
        display ' '
      when 'VERSION-INFORMATION'
        display 'Version: <' xml-text '>'
      when 'ENCODING-DECLARATION'
        display 'Encoding: <' xml-text '>'
      when 'STANDALONE-DECLARATION'
        display 'Standalone: <' xml-text '>'
      when 'START-OF-ELEMENT'
        display 'Start element tag: <' xml-text '>'
      when 'ATTRIBUTE-NAME'
        display 'Attribute name: <' xml-text '>'
      when 'ATTRIBUTE-CHARACTERS'
        display 'Attribute value characters: <' xml-text '>'
      when 'ATTRIBUTE-CHARACTER'
        display 'Attribute value character: <' xml-text '>'
      when 'END-OF-ELEMENT'
        display 'End element tag: <' xml-text '>'
      when 'START-OF-CDATA-SECTION'
        display 'Start of CData: <' xml-text '>'
      when 'END-OF-CDATA-SECTION'
        display 'End of CData: <' xml-text '>'
      when 'CONTENT-CHARACTERS'
        display 'Content characters: <' xml-text '>'
      when 'CONTENT-CHARACTER'
        display 'Content character: <' xml-text '>'
      when 'PROCESSING-INSTRUCTION-TARGET'
        display 'PI target: <' xml-text '>'
      when 'PROCESSING-INSTRUCTION-DATA'
        display 'PI data: <' xml-text '>'
      when 'COMMENT'
        display 'Comment: <' xml-text '>'
      when 'EXCEPTION'
        compute rc = function length (xml-text)
        move rc to st
        call 'nzp' using st l
        display 'Exception ' xml-code ' at offset ' st(l:) '.'
      when other
        display 'Unexpected xml event: ' xml-event '.'
    end-evaluate
    .
End program xmldump.
```


```
Identification division.
  Program-id.
    nzp.
Data division.
 Linkage section.
  1 str pic x(9).
  1 pos binary pic 9(5).
Procedure division using str pos.
    if str = '000000000'
      move 9 to pos
    else
      move 0 to pos
      inspect str tallying pos for leading '0'
      add 1 to pos
    end-if
    goback
    .
End program nzp.
```

Sample XML generate program...

```
Identification division.
  Program-id. XGFX.

Data division.
 Working-storage section.
   01 numItems pic 99 global.
   01 purchaseOrder global.
     05 orderDate pic x(10).
     05 shipTo.
       10 country pic xx value 'US'.
       10 name pic x(30).
       10 street pic x(30).
       10 city pic x(30).
       10 state pic xx.
       10 zip pic x(10).
     05 billTo.
       10 country pic xx value 'US'.
       10 name pic x(30).
       10 street pic x(30).
       10 city pic x(30).
       10 state pic xx.
       10 zip pic x(10).
     05 orderComment pic x(80).
     05 items.
       10 item occurs 0 to 20 times depending on numItems.
         15 partNum pic x(6).
         15 productName pic x(50).
         15 quantity pic 99.
         15 USPrice pic 999v99.
         15 shipDate pic x(10).
         15 itemComment pic x(40).
   01 numChars comp pic 999.
   01 xmlPO pic x(999).
```


```
Procedure division.
  m.
    Move 20 to numItems
    Move spaces to purchaseOrder

    Move '1999-10-20' to orderDate

    Move 'US' to country of shipTo
    Move 'Alice Smith' to name of shipTo
    Move '123 Maple Street' to street of shipTo
    Move 'Mill Valley' to city of shipTo
    Move 'CA' to state of shipTo
    Move '90952' to zip of shipTo

    Move 'US' to country of billTo
    Move 'Robert Smith' to name of billTo
    Move '8 Oak Avenue' to street of billTo
    Move 'Old Town' to city of billTo
    Move 'PA' to state of billTo
    Move '95819' to zip of billTo
    Move 'Hurry, my lawn is going wild!' to orderComment

    Move 0 to numItems
    Call 'addFirstItem'
    Call 'addSecondItem'
    Move space to xmlPO
    Xml generate xmlPO from purchaseOrder count in numChars
    Call 'pretty' using xmlPO value numChars
    Goback.
```


```
Identification division.
  Program-id. 'addFirstItem'.
Procedure division.
    Add 1 to numItems
    Move '872-AA' to partNum(numItems)
    Move 'Lawnmower' to productName(numItems)
    Move 1 to quantity(numItems)
    Move 148.95 to USPrice(numItems)
    Move 'Confirm this is electric' to itemComment(numItems)
    Goback.
End program 'addFirstItem'.

Identification division.
  Program-id. 'addSecondItem'.
Procedure division.
    Add 1 to numItems
    Move '926-AA' to partNum(numItems)
    Move 'Baby Monitor' to productName(numItems)
    Move 1 to quantity(numItems)
    Move 39.98 to USPrice(numItems)
    Move '1999-05-21' to shipDate(numItems)
    Goback.
End program 'addSecondItem'.

End program XGFX.
```


```
Identification division.
  Program-id. Pretty.

Data division.
 Working-storage section.
   01 prettyPrint.
     05 pose pic 999.
     05 posd pic 999.
     05 depth pic 99.
     05 element pic x(30).
     05 indent pic x(20).
     05 buffer pic x(100).

 Linkage section.
  1 doc.
   2 pic x occurs 16384 times depending on len.
  1 len comp-5 pic 9(9).

Procedure division using doc value len.
  m.
    Move space to prettyPrint
    Move 0 to depth posd
    Move 1 to pose
    Xml parse doc processing procedure p
    Goback.
```


```
p.
  Evaluate xml-event
    When 'START-OF-ELEMENT'
      If element not = space
        If depth > 1
          Display indent(1:2 * depth - 2) buffer(1:pose - 1)
        Else
          Display buffer(1:pose - 1)
        End-if
      End-if
      Move xml-text to element
      Add 1 to depth
      Move 1 to pose
      String '<' xml-text '>' delimited by size into buffer
          with pointer pose
      Move pose to posd
    When 'CONTENT-CHARACTERS'
      String xml-text delimited by size into buffer
          with pointer posd
    When 'CONTENT-CHARACTER'
      String xml-text delimited by size into buffer
          with pointer posd
    When 'END-OF-ELEMENT'
      Move space to element
      String '</' xml-text '>' delimited by size into buffer
          with pointer posd
      If depth > 1
        Display indent(1:2 * depth - 2) buffer(1:posd - 1)
      Else
        Display buffer(1:posd - 1)
      End-if
      Subtract 1 from depth
      Move 1 to posd
    When other
      Continue
  End-evaluate.
End program Pretty.
```

Output from generate program

```
<purchaseOrder>
  <orderDate>1999-10-20</orderDate>
  <shipTo>
    <country>US</country>
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo>
    <country>US</country>
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <orderComment>Hurry, my lawn is going wild!</orderComment>
```


```
<items>
    <item>
      <partNum>872-AA</partNum>
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <shipDate> </shipDate>
      <itemComment>Confirm this is electric</itemComment>
    </item>
    <item>
      <partNum>926-AA</partNum>
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
      <itemComment> </itemComment>
    </item>
  </items>
</purchaseOrder>
```