IBM Software Group

IMS25

XML, IMS, COBOL - All Together at the Same Time?

I Would Like to See That!

Pete.Sadler @ uk.ibm.com

Abstract

- IMS Version 9 provides the ability to store XML information in your IMS databases.
- This session describes how you can
  - Store XML information into your favorite IMS database
  - Retrieve XML
  - Retrieve the Information
    - Not just the XML
- Even better, we will see how to do it in COBOL.

Agenda

- Introduction to XML
- How do you store XML in an IMS database? An introduction to XML Support in IMS V9
  - DLIMODEL Utility
  - Decomposed XML documents in an IMS DB
  - Intact XML Documents in an IMS DB
- Program Access to IMS data
  - Access to existing IMS databases
    - COBOL access to IMS data (which is XML information)
  - Accessing intact XML information
    - Using COBOL XML Support to extract the information
    - Using Java IMS XML Support to extract the information
      - Java classes provided by IMS
      - Calling Java routines from COBOL

Introduction to XML

- What is XML?
  - **A markup language, for describing data (rather than its presentation)**
  - **Each piece of data is identified via the markup language**
  - **Unlimited number of tags can be defined**
- Why XML?
  - **It is becoming the interconnection layer of e-business**
  - **The industry direction for application integration and platform independent data interchange**
    - **e.g., for Web Services**
  - **Allows sender and receiver to evolve independently of each other (flexible interface)**
    - **as opposed to Electronic Data Interchange (EDI) for example**

What is XML

- A Standardized, Simple, and Self-Describing Markup Language for documents containing structured or semi-structured information.

```
<A>
   <f1> ~~~ </f1>
   <f2> ~~~ </f2>
   <f3> ~~~ </f3>
   <B>
      <f4> ~~~ </f4>
      <f5> ~~~ </f5>
   </B>
   <B>
      <f4> ~~~ </f4>
      <f5> ~~~ </f5>
   </B>
</A>
```

Why is XML…

- Standard Internet Data Exchange Format
  - Handles encoding
    <xml? version="1.1" encoding="ebcdic-cp-us"?>
  - Handles byte ordering
    <OrderNumber>**110203**</OrderNumber>
  - Human Legible?
  - Easily Parsed
  - Standard

What is XML

- Data-centric
  - Highly structured
  - Limited size and strongly typed data elements
  - Order of elements generally insignificant
  - Invoices, purchase orders, etc.
- Document-centric
  - Loosely structured
  - Unpredictable sizes with mostly character data
  - Order of elements significant
  - Newspaper articles, manuals, etc.

Well formed vs. Valid XML Document

- Well formed – Obeys the XML Syntax Rules
  - must begin with the XML declaration
  - must have one unique root element
  - all start tags must match end-tags
  - XML tags are case sensitive
  - all elements must be closed
  - all elements must be properly nested
  - all attribute values must be quoted
  - XML entities must be used for special characters
- Valid – Conforms to a specific XML Schema

The XML Schema Definition Language

An XML language for defining the legal building blocks of a valid XML document

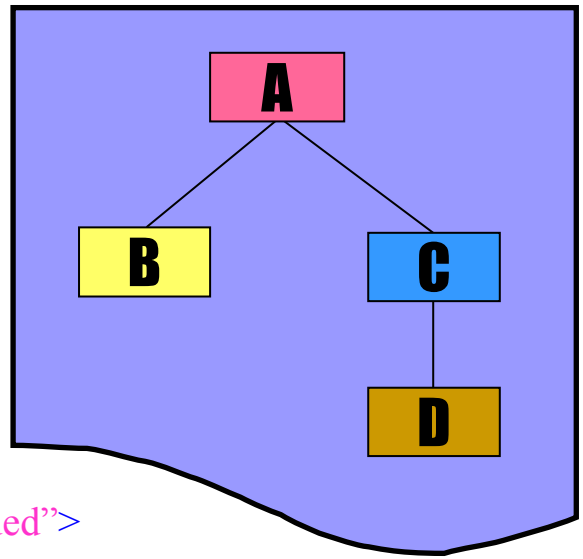
An XML Schema:

- defines elements and attributes that can appear in a document
- defines which elements are child elements
- defines the order and number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Defines an agreed upon communication contract for exchanging XML documents

XML Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.myNamespace.net"
        targetNamespace="http://www.myNamespace.net"
        elementFormDefault="qualified">


  <xsd:element name="A">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Ainteger" type="xsd:int"/>
        <xsd:element name="Astring" type="xsd:string"/>
        <xsd:element name="B" minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Bfield" type="xsd:string"/>
          …
        </xsd:element>
        <xsd:element name="C" minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="D" minOccurs="0" maxOccurs="unbounded">
          </xsd:element>
      …
</xsd:schema>
```

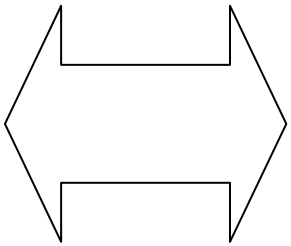


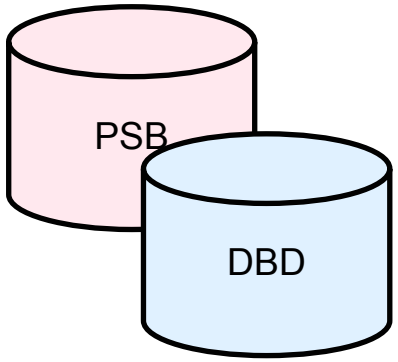
XML Storage in IMS

- Natural mapping between hierarchic XML data and hierarchic IMS database definitions.

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ims="http://www.ibm.com/ims"
        xmlns="http://www.ibm.com/ims/PSBName/PCBName"
        targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
        elementFormDefault="qualified">

  <xsd:annotation>
    <xsd:appinfo>
      <ims:DLI mode="store" PSB="AUTPSB11" PCB="AUTOLPCB"
            dsg="DATASETG" meanLength="1000" numDocs="100"/>
    </xsd:appinfo>
  </xsd:annotation>

  <xsd:element name="A">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:int"/>
        <xsd:element name="field2">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:maxLength value="30"/>
            </xsd:restriction>
  ...
```





IMS to XML mapping metadata

- Physical Metadata
  - Segment Hierarchy (*field relationships – 1-to-1, 1-to-n*)
  - DBD Defined Fields

  - Application Defined Fields
  - Field Type, Type Length, Byte Ordering, Encoding, etc.
  - Offer Field/Segment Renaming *(lift 8 char restriction)*

- Logical Metadata
  - XML layout for fields (*field relationships must still match*)
  - Element vs. Attribute (*names must match*)
  - Type Restrictions, Enumerations, etc.

Defined in DBD

Defined in Copylibs (IMS Java)

Defined in XML Schema

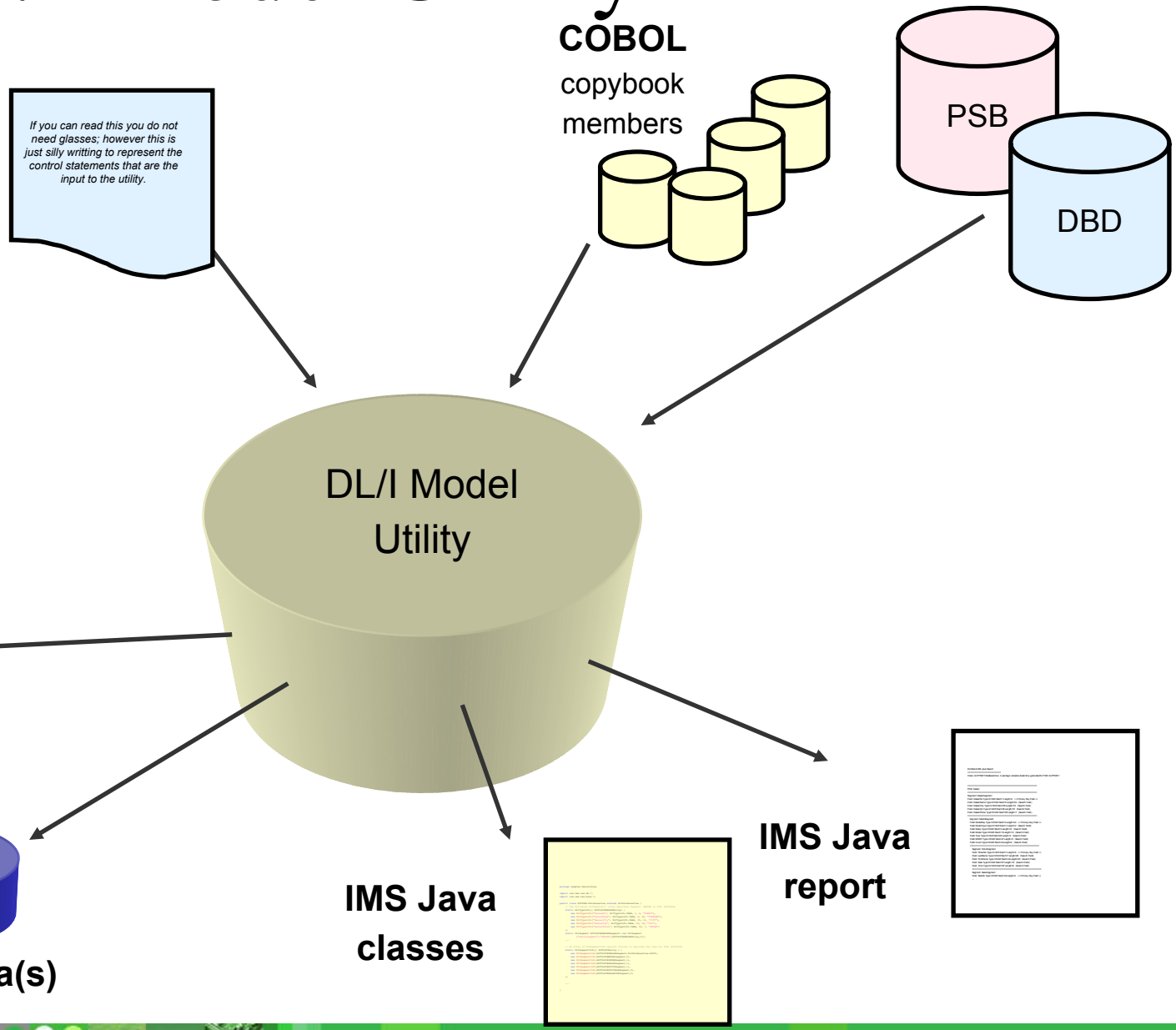
DL/I Model Utility

Control statements:

1) Choose PSBs/DBDs

2) Choose copybook members

3) Aliases, data types, new fields.



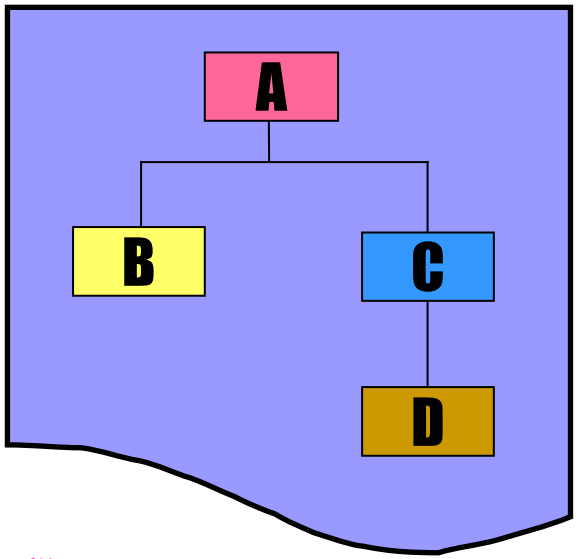
DL/I Model Schema Generation

- Additional Control Statements Keywords

```
OPTIONS PSBds=PSB.SOURCE.PDS     DBDds=DBD.SOURCE.PDS
        GenJavaSource=YES        JavaSourcePath=output/dir
        Package=test.db.psb4     ReportPath=output/dir
        GenXMLSchema=YES         XMLSchemaPath=output/dir
        Outpath=output/dir

PSB psbName=AUTPSB4 Javaname=AutoDealershipDatabase
    PCB PCBName=PCB1 JavaName=MyXMLView GenXMLSchema=YES

// Physical Segments for DEALERDB
SEGM DBDName=DEALERDB SegmentName=DEALER
    FIELD Name=DLRNO     JavaType=INTEGER JavaName=DealerNo
    FIELD Name=DLRNAME JavaType=CHAR     JavaName=DealerName
            …
            …
```

Logical Metadata (XML Schema)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.ibm.com/ims/PSBName/PCBName"
        targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
        elementFormDefault="qualified">

   <xsd:element name="A">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="field1" type="xsd:int"/>
            <xsd:element name="field2">
               <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                     <xsd:maxLength value="30"/>
                  </xsd:restriction>
               </xsd:simpleType>
            <xsd:element name="B" minOccurs="0" maxOccurs="unbounded">
            </xsd:element>
            <xsd:element name="C" minOccurs="0" maxOccurs="unbounded">
               <xsd:element name="D" minOccurs="0" maxOccurs="unbounded">
               </xsd:element>
        …
</xsd:schema>
```



Storing XML Data in IMS databases

- Decomposed Storage
- Intact Storage

Decomposed vs. Intact Storage

- Decomposed (*data-centric storage*)
  - XML tags are stripped from XML data
  - Identical as current IMS storage
  - Strict data-centric XML Schema validated data
  - EBCDIC encoding
  - Searching on IMS Search Fields

- Intact (*document-centric storage*)
  - Entire XML document is stored (including tags)
  - Relaxed un-validated data
  - Any desired encoding is possible
  - Searching is through XPath specified and generated Secondary Indexed Side Segments

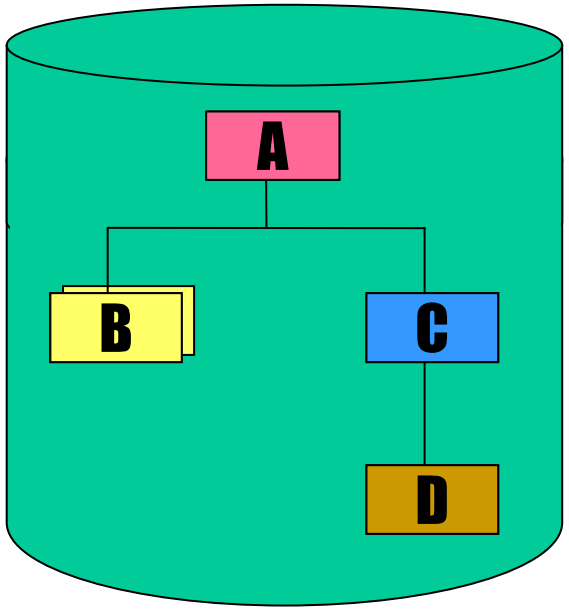
Decomposed XML Retrieval in IMS

Composed XML

```
<A>
  <f1> [ ] </f1>
  <f2> [ ] </f2>
  <f3> [ ] </f3>
  <B>
    <f4> [ ] </f4>
    <f5> [ ] </f5>
  </B>
  <B>
    <f4> [ ] </f4>
    <f5> [ ] </f5>
  </B>
  <C>
    <f6> [ ] </f6>
    <f7> [ ] </f7>
    <D>
      <f8> [ ] </f8>
      <f9> [ ] </f9>
    </D>
  </C>
</A>
```

XML Schema/ Metadata

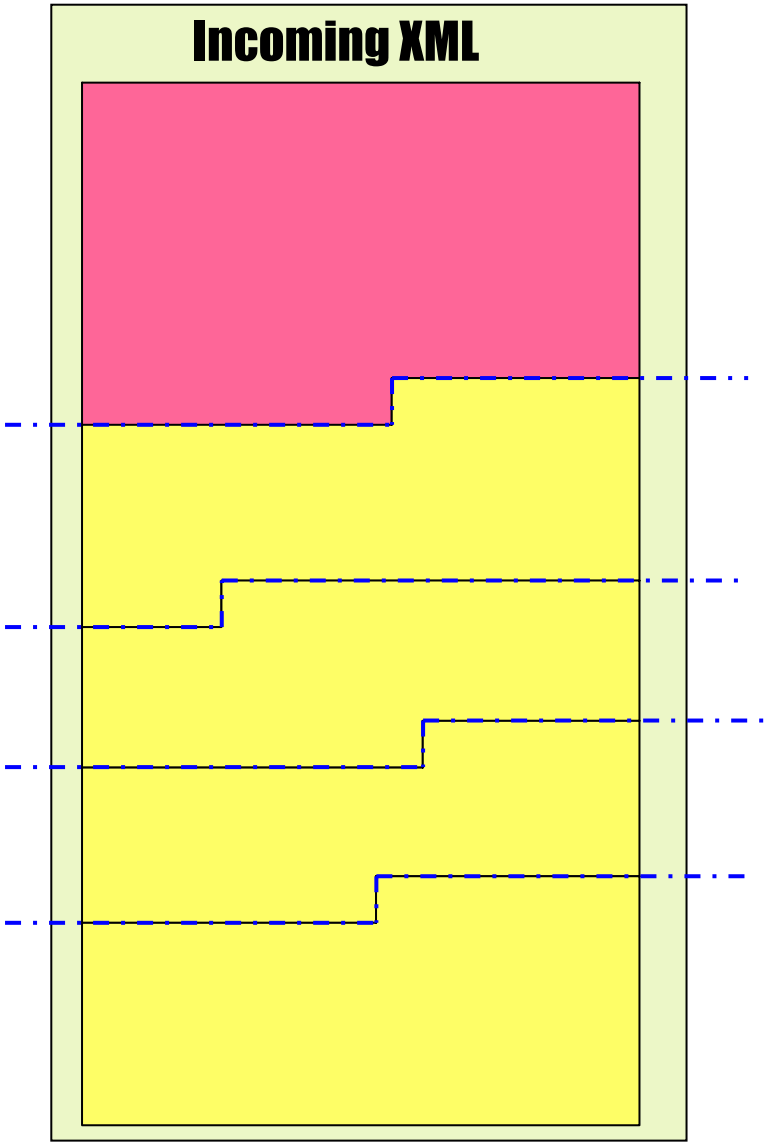
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ims="http://www.ibm.com/ims"
        xmlns="http://www.ibm.com/ims/PSBName/PCBName"
        targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
        elementFormDefault="qualified">

  <xsd:annotations go here/>

  <xsd:element name="A">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="field1" type="xsd:int"/>
        <xsd:element name="field2">
          <xsd:simpleType>
  ...
  <xsd:element name="B">
    <xsd:complexType>
  ...
  <xsd:element name = "C">
    <xsd:complexType>
  ...
  <xsd:element name = "D">
    <xsd:complexType>
```

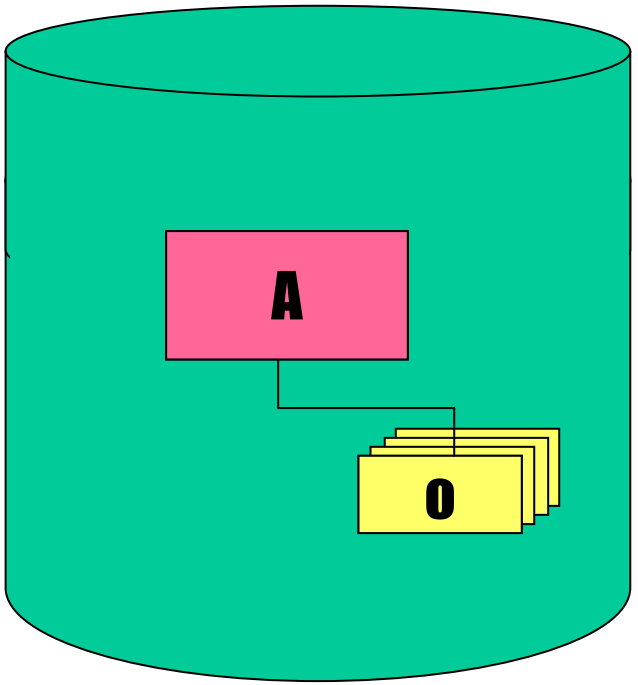


Decomposed Storage

- XML document must be parsed and validated.
- Data must be converted to *traditional* IMS types
  - COMP-1, COMP-2, etc.
  - EBCDIC CHAR, Picture Strings
- Stored data is searchable by IMS and transparently accessible by non-XML enabled applications.

Intact XML Storage in IMS



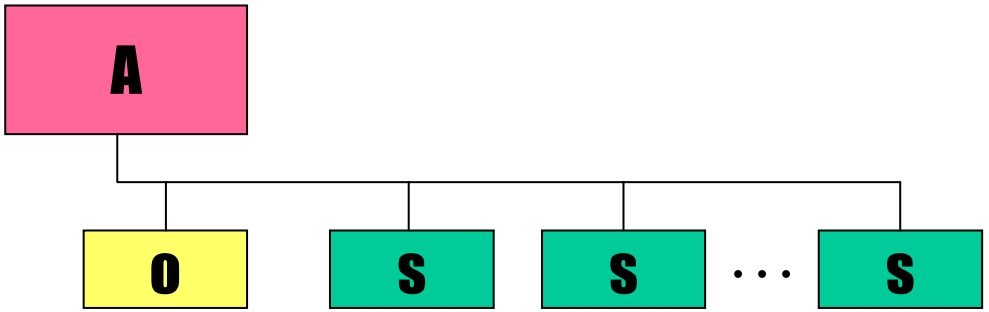


Intact Storage

- No (or little) XML Parsing or Schema validation
  - Storage and Retrieval Performance
- No (or little) data type conversions
  - Unicode storage
- Stored documents are no longer searchable by IMS and only accessible to XML-enabled applications
  - XPath side segments

Intact Storage Secondary Indexing

- XPath expression identifying Side Segments
  - Side segment is converted to *traditional* data type and copied into segment.
- Side Segments are secondary indexed with documents root as target.

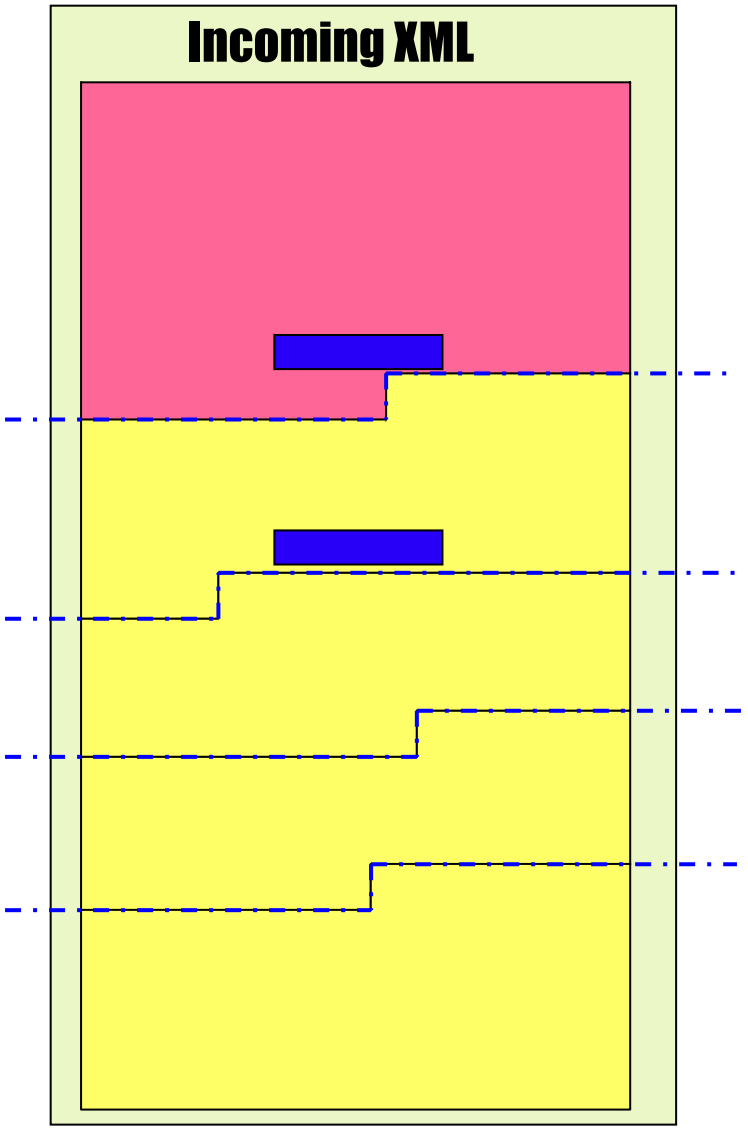


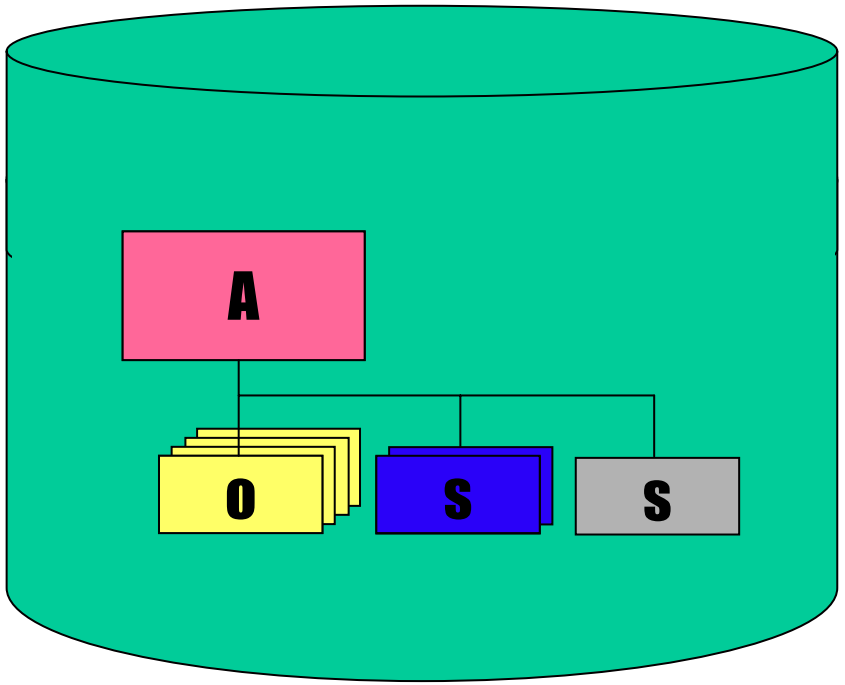
Example:

XPath="/Dealer/DealerName"

XPath="/Dealer/Model[Year>1995]/Order/LastName"

Intact XML Storage in IMS





Example:

XPath="/A/B/f4"
XPath="/A/E/f1"

How can we access the Information?

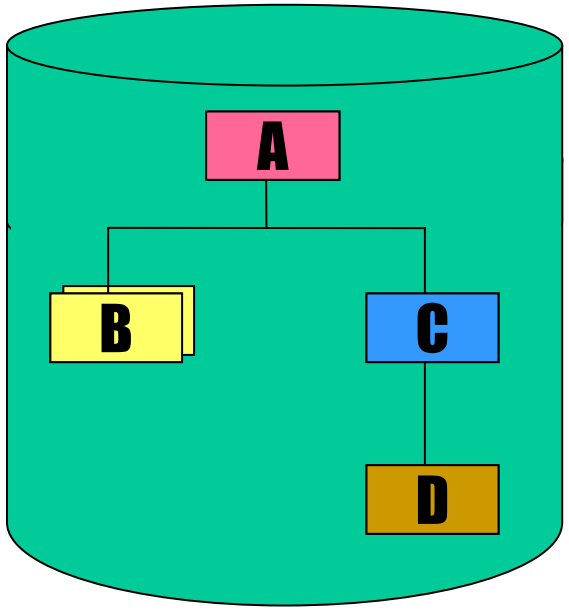
Decomposed XML Retrieval in IMS

Composed XML

```
<A>
   <f1>                </f1>
   <f2>                </f2>
   <f3>                </f3>
   <B>
      <f4>             </f4>
      <f5>             </f5>
   </B>
   <B>
      <f4>             </f4>
      <f5>             </f5>
   </B>
   <C>
      <f6>             </f6>
      <f7>             </f7>
      <D>
         <f8>          </f8>
         <f9>          </f9>
      </D>
   </C>
</A>
```

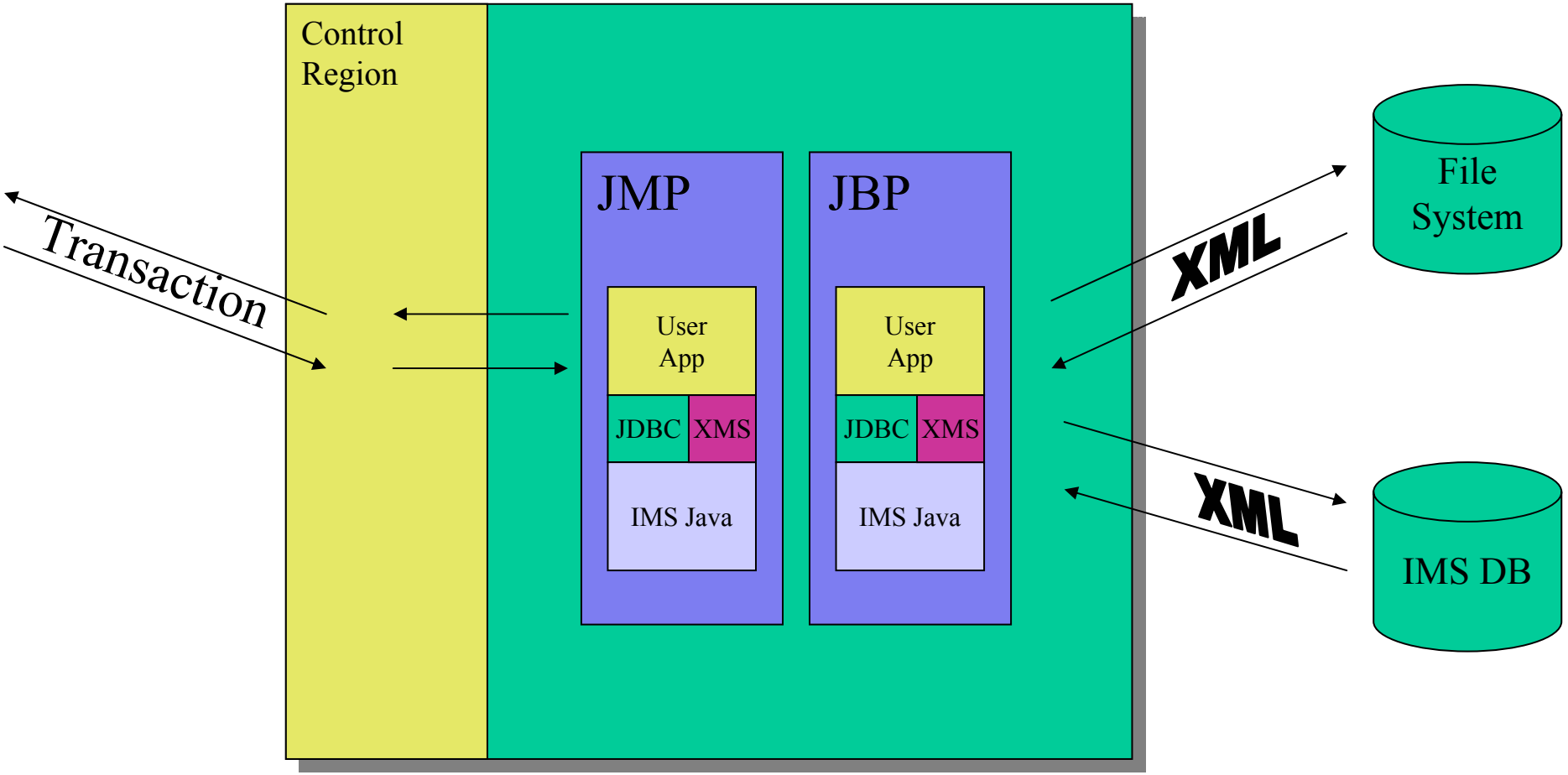
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ims="http://www.ibm.com/ims"
        xmlns="http://www.ibm.com/ims/PSBName/PCBName"
        targetNamespace="http://www.ibm.com/ims/PSBName/PCBName"
        elementFormDefault="qualified">

   <xsd:annotations go here/>

   <xsd:element name="A">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="field1" type="xsd:int"/>
            <xsd:element name="field2">
               <xsd:simpleType>
   …
   <xsd:element name="B">
      <xsd:complexType>
   …
   <xsd:element name = "C">
      <xsd:complexType>
   …
   <xsd:element name = "D">
      <xsd:complexType>
```

XML Schema/ Metadata



Java Access to XML Data

XMS Java Interface

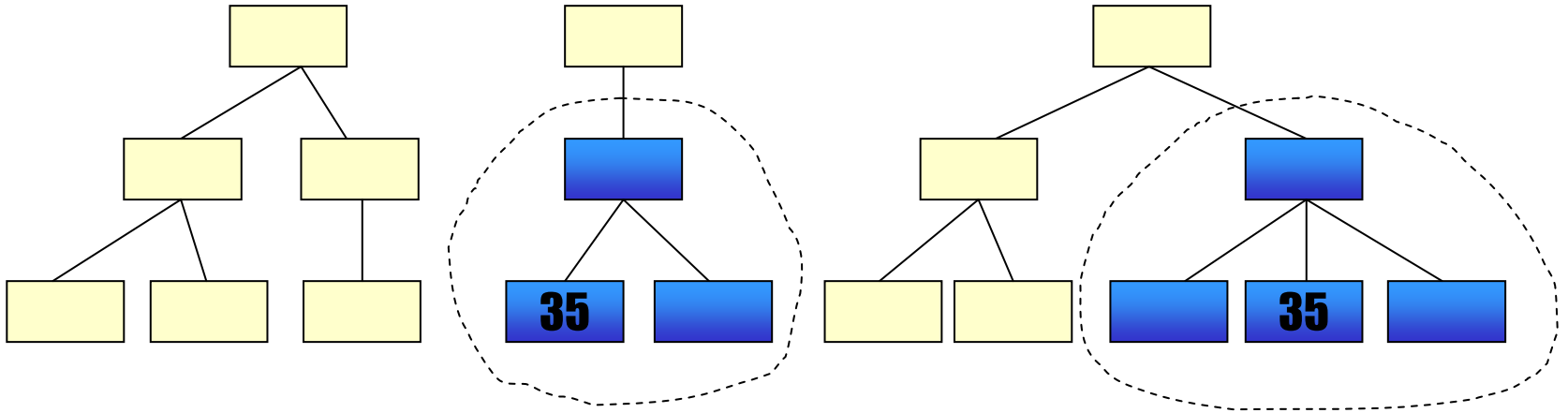


XMS Java Interface

- Adds 2 User Defined Funtions (UDF) to the IMS Java JDBC SQL interface
  - retrieveXML()
  - storeXML()
- Runs as an IMS Java Application
  - JDR (JMP, JBP)
  - DB2 Stored Procedure
  - CICS
  - WebSphere

RetrieveXML() UDF

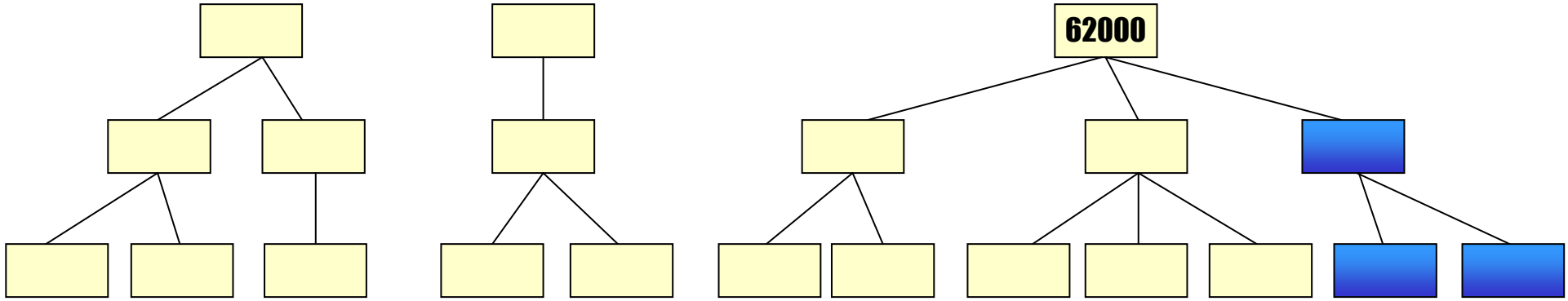
```
SELECT retrieveXML(B)
FROM C
WHERE C.fieldA = ’35’
```



**Two Rows of XML CLOBs in the ResultSet*

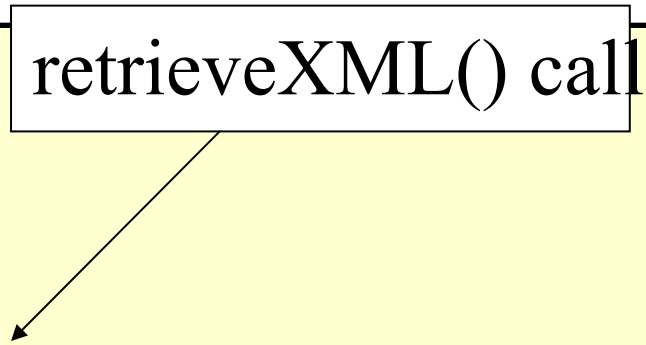
StoreXML() UDF

```
INSERT INTO B (storeXML())
VALUES (?)
WHERE A.fieldA = ’62000’
```



**Insert Statement must be a Prepared Statement*

Execute Query




```
public void processMessage(String dealerName) {
    obtain connection...

    String query =
        "SELECT DealerSegment.DealerName, retrieveXML(DealerSegment) AS DealerXMLDoc" +
        " FROM Dealer.DealerSegment" +
        " WHERE DealerSegment.DealerName = '" + dealerName + "'";

    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);

    process results...
    close connection...
}
```

Process Results

getClob() call

```
public void processMessage(String dealerName) {
     obtain connection...
     execute query...

     while (results.next()) {

          Clob xmlDoc = results.getClob("DealerXMLDoc");

          saveClobToFile(xmlDoc, results.getString("DealerName"));
     }

     close connection...
 }
```

Process Results

getCharacterStream() or
getAsciiStream()

```
public void saveClobToFile(Clob clob, String fileName) throws IOException {

    Reader reader = clob.getCharacterStream();
    FileWriter writer = new FileWriter(fileName + ".xml");

    char[] line = new char[1024];
    int x = reader.read(line,0,1024);
    while (x != -1) {
        writer.write(line,0,x);
        x = reader.read(line,0,1024);
    }

    reader.close();
    writer.close();
}
```

Updates to the COBOL language

It has changed in recent years

Objectives for IBM Object Oriented COBOL

- Complement existing COBOL:Java interoperation
  - mediated by middleware, based on connectors
  - COBOL and Java running in different address spaces or machines
  - only for COBOL transactions
- Enable fine-grained interoperation of COBOL and Java within an address space, both:
  - COBOL invocation of Java
  - Java invocation of COBOL
- Fine-grained interoperation (interlanguage communication) provides:
  - better performance
  - use of non-transactional COBOL
- Improve integration of COBOL with WebSphere Application Server
  - COBOL client invocation of enterprise beans
  - Future support for COBOL execution within WebSphere server regions

Example scenarios

- Enhance existing COBOL routines to access new business logic written in Java
- Write new business logic in Java, access existing libraries of production COBOL routines
  - process existing COBOL data in QSAM/VSAM files
- COBOL programs can invoke Java for full-function XML parsing
  - (use new direct COBOL support for high-speed parsing of input XML documents!)
- COBOL programs can invoke methods (directly) on enterprise beans running in a WebSphere server region
- - or -
- COBOL programs can invoke Java routines, that act as J2EE clients accessing enterprise beans
- Write new IMS message processing or database logic in Java, mixing with COBOL to leverage existing IMS COBOL applications and IMS programming skills

Enterprise COBOL XML support

- Much faster than general purpose parsers
  - Designed for high-speed transaction processing
- Runs in all COBOL run-time environments:
  - CICS, IMS, batch, TSO, USS, ...
- Use any transport mechanism for XML documents
  - MQSeries, CICS transient queue or COMMAREA, IMS message processing queue, WebSphere, etc.
- Provides basic SAX-style parsing
- XML Parser is part of the run-time library
  - Can be used from Enterprise COBOL or Enterprise PL/I
- Inbound XML documents only for now
  - Outbound can use MOVE CORRESPONDING, STRING, group declarations, etc. to create XML documents today

Why process XML in COBOL?

- Keep development control in one place/style
- Guarantee correct language semantics
  - sign configuration
  - layout/padding
  - picture constraints
- Exploits your existing assets/skills/literacy
- Non-COBOL programs can communicate
- data to/from COBOL without having to know the COBOL data structure formats!

Hello XML World Program

```
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.
```

Output

```
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, COBOL, and Application Modernization

Bringing XML to the application

Enterprise COBOL XML support

- Much faster than general purpose parsers
  - Designed for high-speed transaction processing
- Runs in all COBOL run-time environments:
  - CICS, IMS, batch, TSO, USS, ...
- Use any transport mechanism for XML documents
  - 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
- Inbound XML documents only for now
  - Outbound can use MOVE CORRESPONDING, STRING, group declarations, etc. to create XML documents today

Enterprise COBOL XML support...

- Parses XML documents that are in memory, in a COBOL alphanumeric or national data item
- Parses XML documents into individual pieces
  - 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 Feature

- XML PARSE statement
  - The COBOL interface to high-speed XML parser
- XML special registers
  - XML-CODE: communicates status of parsing
  - XML-EVENT: describes each event in parse
  - 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'
          ...
```

Accessing Intact XML from COBOL

- Read Intact XML from your database
  - GU, GN
- XML PARSE in COBOL
- Voila, the information

Enterprise COBOL and Java - Interoperation

A more sophisticated alternative

Objectives for IBM Object Oriented COBOL

- Enable fine-grained interoperation of COBOL and Java within an address space, both:
  - COBOL invocation of Java
  - Java invocation of COBOL
- Complement existing COBOL:Java interoperation
  - mediated by middleware, based on connectors
  - COBOL and Java running in different address spaces or machines
  - only for COBOL transactions
- Fine-grained interoperation (interlanguage communication) provides:
  - better performance
  - use of non-transactional COBOL
- Improve integration of COBOL with WebSphere Application Server
  - COBOL client invocation of enterprise beans
  - Future support for COBOL execution within WebSphere server regions

Example scenarios

- Enhance existing COBOL routines to access new business logic written in Java
- Write new business logic in Java, access existing libraries of production COBOL routines
- New business logic written in Java can invoke COBOL routines to process existing COBOL data in QSAM/VSAM files
- COBOL programs can invoke Java for full-function XML parsing
  - (use new direct COBOL support for high-speed parsing of input XML documents!)
- COBOL programs can invoke methods (directly) on enterprise beans running in a WebSphere server region
- - or -
- COBOL programs can invoke Java routines, that act as J2EE clients accessing enterprise beans
- Write new IMS message processing or database logic in Java, mixing with COBOL to leverage existing IMS COBOL applications and IMS programming skills

Java Interoperation

- Existing OO COBOL syntax re-based on Java, and the Java Native Interface (JNI)
  - COBOL INVOKE statement maps onto Java JNI calls
  - COBOL class and method definitions define Java native methods
  - SOM-based OO COBOL support removed
    - SOMObjects has been removed from z/OS
- Documentation and assistance in mapping Java data types to/from COBOL
- Support for JNI programming in COBOL
- COBOL COPY file analogous to jni.h, enables access to JNI callable services
- Java prerequisite: IBM Java 2 Technology Edition SDK 1.3.0

Client-side syntax

- Declare referenced class and full external class name
  - Configuration section.
  - Repository paragraph.
  - Class 'Employee' is 'com.acme.Employee'.
- Declare object reference
  - 01 anEmployee usage object reference Employee.
- Invoke instance method
  - Invoke anEmployee 'payRaise'
  - using by value amount
- Invoke static method
  - Invoke Employee 'getNbrEmployees'
  - returning totalEmployees
- Create instance object
  - Invoke Employee New using by value id
  - returning anEmployee

Interoperable data types for method parameters

Java	COBOL
boolean	01 B pic X. 88 B-false value X'00'. 88 B-true value X'01' through X'FF'.
byte	Pic X or Pic A
short	Pic S9(4) usage binary or comp-5
int	Pic S9(9) usage binary or comp-5
long	Pic S9(18) usage binary or comp-5
float	Usage comp-1
double	Usage comp-2
char	Pic N usage national
class types (object references) including strings and arrays	Usage object reference class-name

Compiler implementation: map COBOL syntax onto a sequence of generated JNI calls

- Syntax:
  - Invoke anEmployee 'payRaise' using by value amount
- Implementation (under the covers):
  - Construct argument list
    - automatically convert any floating point arguments to/from IEEE float
    - automatically handle implicit arguments required by JNI
  - Acquire class object, via GetObjectClass or FindClass
  - Convert method name ('payRaise') to Unicode
  - Construct method signature: null-terminated ASCII string '(I)V'
  - Call GetMethodId passing JNI env, class object, method name, signature
  - Call CallVoidMethod passing env, object, method id, and arguments

COBOL native method – syntax

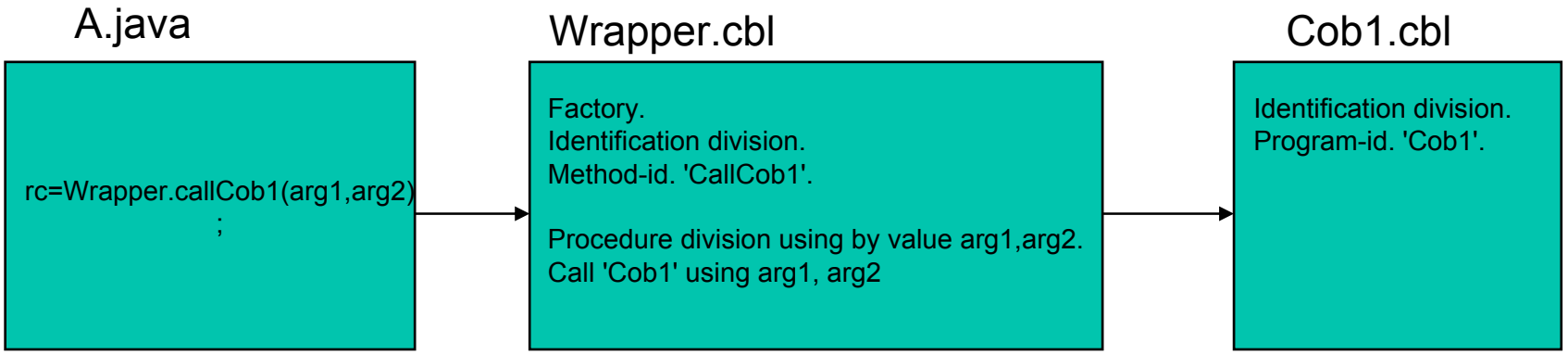
- Identification Division.
- Class-id. Manager inherits Employee.
- Environment Division.
- Configuration section.
- Repository.
- Class Manager is 'com.acme.Manager'
- Class Employee is 'com.acme.Employee'.
- Identification division.
- Object.
- Procedure Division.
- Identification Division.
- Method-id. 'Hire'.
- Data Division.
- Linkage section.
- 01 anEmployee usage object reference Employee.
- Procedure Division using anEmployee.
- …
- End method 'Hire'.
- End Object.
- End class Manager.

COBOL Classes

- OBJECT paragraph - defines object instance methods
- FACTORY paragraph - defines static methods
- COBOL classes can inherit from COBOL or Java classes
- Java classes can inherit from COBOL classes
- Methods can override inherited methods
- Methods can be overloaded
- Method names can be formed using Unicode characters, per Java rules
- Method parameters must be COBOL data types that map to Java data types

Accessing existing procedural COBOL code from Java

- Code in COBOL classes can CALL procedural COBOL code
  - Write a COBOL wrapper class for the existing procedural COBOL program
  - Define a Factory method containing a CALL to the COBOL program
- Access COBOL from Java using a static method invocation, e.g.
  - rc=Wrapper.callCob1(arg1,arg2);



Compile and link of COBOL class definition

- Compile of COBOL class definition generates two artifacts:
  - COBOL object program implementing native method(s)
  - Java class source that declares the native methods and manages DLL loading
- COBOL object program is linked to form DLL: libclassname.so
- Java class is compiled (with javac) to form classname.class

Structuring mixed COBOL and Java applications

- Application can start with:
  - a COBOL program
    - use classes written in COBOL or Java
  - a main method of a Java class
    - main method must be public, static, void with a single parameter of type java.lang.String[]
  - a main method of a COBOL class
    - main must be a Factory method with no RETURNING clause, with a single parameter that is an object reference of type array of java.lang.String

Executing COBOL and Java mixed applications

- Designed primarily to run under z/OS Unix System Services (USS) environment
  - with application components residing in the Hierarchical File System (HFS)
- Recommendations:
  - Execute COBOL:Java applications that start with a COBOL program:
    - from a Unix shell command prompt, or
    - from JCL or TSO/E, using the BPXBATCH utility
  - Execute COBOL:Java applications that start with the main method of a Java or COBOL class:
    - with the java command from a Unix shell command prompt, or using BPXBATCH from batch JCL or TSO/E
    - in an IMS Java dependent region (see below)

COBOL and Java interoperation under IMS

- IMS applications can now be written in a mixture of COBOL and Java
- Applications execute in IMS Java dependent regions
  - JMP - Java Message Processing region
  - JBP - Java Batch Processing region
- Prerequisites:
  - IMS Version 8, or
  - IMS Version 7 with PTFs for APAR PQ53944 (UQ61540) and PQ54039 (UQ61590)
  - Java 2 Technology Edition SDK 1.3.1, which provides persistent reusable JVM for transaction processing

COBOL and Java interoperation under IMS (continued)

- Application front-end must be main method of a Java or COBOL class
- Java components access IMS databases and message processing services using IMS Java class library
- COBOL components use traditional DL/I calls, using the AIB interface, e.g.
  - CALL 'CEETDLI' using GU, AIB, input-area
- See IMS Version 8 Java User's Guide, and new Enterprise COBOL Programming Guide

Getting started with COBOL and Java interoperability

- Install Java 2 Technology Edition SDK
  - SDK 1.3.1 if you will be running under IMS
  - SDK 1.3.0 or 1.3.1 otherwise
- For z/OS V1R1 or OS/390 V2R10, install OS/390 Support for Unicode (HUNI2A0).
- For either OS/390 or z/OS systems, ensure that the Unicode Conversion Services are configured for COBOL use. At minimum, configure conversion between these pairs of CCSIDs:
  - 1140 to/from 1200
  - 1140 to 1208
  - 1200 to 1208
- Configure these conversions with the default technique-search-order 'RECLM'.

Getting started with COBOL and Java interoperability (continued)

- See the Enterprise COBOL V3R2 Customization Guide for details and sample JCL for configuring Unicode Conversion Services for COBOL
- See the sample OO application and makefile that is shipped with COBOL in /usr/lpp/cobol/demo/oosample. Try compiling and running this application.

Accessing Intact XML from COBOL

- IMS runs your COBOL program
  - COBOL invokes a Java routine
  - Java routines reads Intact XML from your database
    - RetrieveXML()
  - PARSE the XML data stream
- Voila, your COBOL program has read XML

Summary

- Storing XML into your IMS Databases
  - Intact Storage
  - Decomposed Storage
- Accessing XML in your IMS Databases
  - COBOL access decomposed XML
  - Java access to intact XML
  - COBOL access to Java

Acknowledgements

- Christopher Holtz
  - IMS Development
- Kiran Challapalli
  - IMS Development
- Geoff Nicholls
  - IMS Advocate
- Tom Ross
  - Senior Developer - COBOL compiler, IBM Software Group
- Michael Paolini
  - Linux Integration Centre, IBM Software Group