# Managing XML Data in a CA IDMS™ Application

Paul McRoberts
MPO

CA IDMS™ Technical Conference

Framingham MA
December 2-5, 2014

## Abstract


- We are modernizing our key CA IDMS order entry application by adding XML-based transactions. This session covers the architecture and processing used for XML input, XML output and SQL-based tables and procedures for data persistence and standardization of common routines.

## Biography

Mr. McRoberts began with CA IDMS release 4.5 in 1979.
CA IDMS has been central to his entire career. He has worked for consulting companies and CA supporting CA IDMS as a programmer, teacher, application DBA and system DBA. He is now a US Government employee.

He teaches ballroom dancing, plays the guitar, and likes long walks on the beach.




## Agenda

- Processing XML Input coming in through CA IDMS Server / JDBC
  - COBOL XML Parser statement
  - IBM COBOL program using the XML Parse verb running as a CA IDMS SQL Procedure
  - Provide sample syntax
- Generating XML Output and sending CA IDMS Server / JDBC
  - Convert a print report to XML
  - Generate XML for transaction processing and data replication
  - Provide sample syntax
- Word Document available to supplement this Power Point

# Processing XML Input

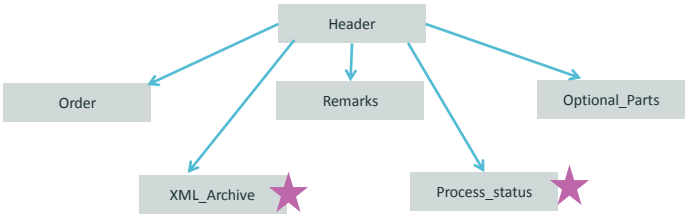
## Processing XML Input
## IBM COBOL program using the XML Parse verb

- Running as a non validating parser
- Nothing magic about writing the code
  - Every XML tag has to be managed and the data saved
- XML structures are similar to a hierarchical database
  - Structures that can occur 0, 1, or many times
- XML Tags are not necessarily unique - context is key
- DISPLAY commands for tracing the code.
  - DISPLAY command output gets written to the CA IDMS LOG
- Handles over 10 different XML transactions
- This is not an XML tutorial

## Processing XML Input

Sample table structure built by the parser

- ***Sample table structures***




- ***The ★ identifies tables that are managed by an SQL Procedure***




## Sample XML

```
<ProductionOrder xmlns="http://sampleIUA.xml.schema">
  <OrderType>Custom</OrderType>
  <OrderNumber>2014-10-01-00001</OrderNumber>
  <ModelName>Tundra</ModelName>
  <ModelId>4781</ModelId>
  <MoreStuff>...</MoreStuff>
  <ProductionRemarks>
    <Remark>Do NOT begin schedule production until
     all the selected options are available.</Remark>
</ProductionRemarks>
<SpecialPartsList>
<SpecialPart>
  <Type>Exterior</Type>
  <PartNum>1412</PartNum>
  <Name>Power rear window</Name>
 <Remark>Standard with the SR5 and Limited package</Remark>
</SpecialPart>
```

## Sample XML

**Tag "Remark" is in two different structures.**
**It likely gets placed into different database records or tables.**

```
...
<SpecialPart>
  <Type>Interior</Type>
  <PartNum>2251</PartNum>
  <Name>2 way adjustable passenger seat</Name>
  <Remark>Standard equipment</Remark>
</SpecialPart>
  <MoreStuff>...</MoreStuff>
</SpecialPartsList>
</ProductionOrder>
```




## COBOL XML Parser statement

Introduction

- The IBM manuals, COBOL Reference and COBOL Programmer's Guide, provide good examples and sample programs
- Another team developed and defined the XML transactions that are discussed here and in the output example 2
- The following examples provide some insight into a way to use the XML PARSE verb

## COBOL XML Parser statement

Syntax to invoke the parser

```
XML PARSE XML-INPUT
   PROCESSING PROCEDURE
   006-EVALUATE-STRUCTURE THRU 006-EXIT
ON EXCEPTION
   PERFORM 010-EXCEPTION THRU 010-EXIT
NOT ON EXCEPTION
   MOVE 'XML PARSED SUCCESSFULLY '
   TO PARSER-MESSAGE
   IF TRACE-PARSER THEN
      DISPLAY 'XML PARSED SUCCESSFULLY ' XML-CODE
   END-IF
END-XML.
```




## COBOL XML Parser statement

How does it work?

- For each iteration, these two SPECIAL REGISTERS are populated and used in the “006” paragraph
  - XML-EVENT specifies the structural component
  - XML-TEXT specifies the data value
  - There are others
- <ProductionOrder xmlns="**http://sampleIUA.xml.schema**">
  - XML-EVENT is set to **START-OF-ELEMENT**
  - XML-TEXT is set to ‘**ProductionOrder**’
  - This is what we will track through the sample code

## COBOL XML Parser statement

XML-EVENT reference list

- Most processing deals with these first three
  - **START-OF-ELEMENT, CONTENT-CHARACTERS, END-OF-ELEMENT**
  - START-OF-DOCUMENT, END-OF-DOCUMENT, NAMESPACE-DECLARATION
  - VERSION-INFORMATION, ENCODING-DECLARATION, STANDALONE-DECLARATION
  - ATTRIBUTE-NAME, ATTRIBUTE-CHARACTERS, ATTRIBUTE-CHARACTER
  - START-OF-CDATA-SECTION, END-OF-CDATA-SECTION, CONTENT-CHARACTER
  - PROCESSING-INSTRUCTION-TARGET, PROCESSING-INSTRUCTION-DATA, COMMENT
- The EVALUATE statement should handle all these values




## COBOL XML Parser statement

Evaluate the XML-EVENT value

```
006-EVALUATE-STRUCTURE.
     ADD 1 TO ITERATION-COUNT.
     EVALUATE XML-EVENT
* ==> ORDER XML EVENTS MOST FREQUENT FIRST
*    These are the most common!
       WHEN 'START-OF-ELEMENT'
          ADD 1 TO START-OF-ELEMENT-COUNT
          PERFORM 100-START-OF-ELEMENT       THRU 100-EXIT
       WHEN 'CONTENT-CHARACTERS'
          ADD 1 TO CONTENT-CHARACTERS-COUNT
          PERFORM 110-CONTENT-CHARACTERS     THRU 110-EXIT
       WHEN 'END-OF-ELEMENT'
          ADD 1 TO END-OF-ELEMENT-COUNT
          PERFORM 120-END-OF-ELEMENT         THRU 120-EXIT
```

- Other XML-EVENT types follow with their own paragraphs.
  - You may not need or use some of the XML-EVENT and XML-TEXT
  - NAMESPACE-DECLARATION is sent, handled, but not used

## COBOL XML Parser statement

Track the XML structure and data

- Each XML tag uses these Data Division entries

```
12 ORDERTYPE                  PIC X(09) VALUE
   'OrderType'.
12 ORDERTYPE-SW               PIC X VALUE 'N'.
  88 ORDERTYPE-OPEN           VALUE 'Y'.
  88 ORDERTYPE-CLOSE          VALUE 'N'.
12 PRODUCTIONORDER            PIC X(15) VALUE
   'ProductionOrder'.
12 PRODUCTIONORDER-SW         PIC X VALUE 'N'.
  88 PRODUCTIONORDER-OPEN     VALUE 'Y'.
  88 PRODUCTIONORDER-CLOSE    VALUE 'N'.
```

- The first entry documents the XML tag
- The "88's" keep track of what tag is currently being referenced
  - Multiple tags can be open at one time
- **The data values move to the target database records**




## COBOL XML Parser statement

Evaluate the XML-EVENT value – 1st tag

```
100-START-OF-ELEMENT.
  IF TRACE-PARSER THEN
     DISPLAY '--100 -Start of Element:Level =' OPEN-TAG-COUNT
     DISPLAY '- START TAG=<' XML-TEXT '>'
  END-IF.
  MOVE XML-TEXT   TO LAST-TAG.
*    First check to see if we have identified a transaction.
  IF PRODUCTIONORDER-OPEN          THEN
      PERFORM 400-PRODUCTION-ORDER
         THRU 400-EXIT
      GO TO 100-EXIT.
  IF PRODUCTIONORDERUPDATE-OPEN          THEN
      PERFORM 410-PRODUCTION-ORDER-UPDATE
         THRU 410-EXIT
      GO TO 100-EXIT.
   …
```

- Neither of the IF statements are true, continue …

## COBOL XML Parser statement

Use the EVALUATE statement to identify the 1st tag value (XML-TEXT)

- **If nothing was "OPEN", EVALUATE the tag**
- **Later in paragraph 100…**

```
IF TRACE-PARSER THEN
    DISPLAY '--100 - No tag open.  Begin the EVALUATE!'
    DISPLAY '--100 - LOOKING FOR:<' XML-TEXT  '> end of text.'
END-IF.

EVALUATE XML-TEXT
    WHEN PRODUCTIONORDER
       MOVE 'READY' TO XML-TRANS-STATUS
       ADD 1 TO OPEN-TAG-COUNT
       SET PRODUCTIONORDER-OPEN   TO TRUE

    WHEN PRODUCTIONORDERUPDATE
    …
```




## COBOL XML Parser statement

Use the EVALUATE statement - handle unknown tags

- Update the following list when a new XML transaction is completed.

```
         WHEN OTHER
           MOVE -1 to XML-CODE
           SET  XML-PARSE-ERROR    TO   TRUE
           MOVE 'Invalid Opening Tag '
             TO TRACKING-MESSAGE
           DISPLAY '100-Invalid opening tag: ' XML-TEXT
           DISPLAY 'These are the valid values: '
           DISPLAY '-- ' PRODUCTIONORDER
           DISPLAY '-- ' PRODUCTIONORDERUPDATE
           DISPLAY 'IUACXML0 is ending. '
       END-EVALUATE.
100-EXIT.
    EXIT.
```

## COBOL XML Parser statement

Process the next tag that has data - <OrderType>**Custom**</OrderType>

- First tag completed, Now process OrderType.
- ```
  EVALUATE XML-EVENT
      WHEN 'START-OF-ELEMENT'
          ADD 1 TO START-OF-ELEMENT-COUNT
          PERFORM 100-START-OF-ELEMENT  THRU 100-EXIT
  ```
- ```
  100-START-OF-ELEMENT.
  …
       IF PRODUCTIONORDER-OPEN          THEN
          PERFORM 400-PRODUCTION-ORDER
             THRU 400-EXIT
          GO TO 100-EXIT.
   …
  100-EXIT.
      EXIT.
  ```




## COBOL XML Parser statement

Programmatically track the XML structure

- This paragraph processes all tags that are directly subordinate to the **ProductionOrder** tag.

```
400-PRODUCTION-ORDER.
     IF TRACE-PARSER THEN
         DISPLAY '-- 400-PRODUCTION-ORDER '
     END-IF.

     EVALUATE XML-TEXT
       WHEN ORDERTYPE
          ADD 1 TO OPEN-TAG-COUNT
          SET ORDERTYPE-OPEN            TO TRUE
          GO TO 400-EXIT
       WHEN ORDERNUMBER
          ADD 1 TO OPEN-TAG-COUNT
          SET ORDERNUMBER -OPEN    TO TRUE
          GO TO 400-EXIT
```

… **Many other tags.**

## COBOL XML Parser statement

Programmatically track the XML structure

- Always look for the unexpected. This is especially helpful when testing

```
WHEN OTHER
   IF TRACE-PARSER THEN
       DISPLAY '-- 400-PRODUCTION-ORDER `
       DISPLAY 'UNEXPECTED XML TAG:<' XML-TEXT  '>.'
       DISPLAY 'Verify no new tags.'
   END-IF
 END-EVALUATE.

 400-EXIT.
     EXIT.
```




## COBOL XML Parser statement

**The next item is XML-EVENT, 'CONTENT-CHARACTERS' for 'OrderType'**

- Back to paragraph **006-EVALUATE-STRUCTURE**

```
    EVALUATE XML-EVENT
…
    WHEN 'CONTENT-CHARACTERS'
      ADD 1 TO CONTENT-CHARACTERS-COUNT
      PERFORM 110-CONTENT-CHARACTERS     THRU 110-EXIT
…

110-CONTENT-CHARACTERS.
…
     IF PRODUCTIONORDER-OPEN                THEN
         PERFORM 401M-PRODUCTION-ORDER-MOVE  THRU 401M-EXIT
         GO TO 110-EXIT.
…
 110-EXIT.
     EXIT.
```

## COBOL XML Parser statement

Put data into the target database table column

```
401M-PRODUCTION-ORDER-MOVE.
     IF TRACE-PARSER THEN
         DISPLAY '-- 401M-PRODUCTION-ORDER-MOVE.    '
     END-IF.
     IF ORDERTYPE-OPEN
         MOVE XML-TEXT       TO ORDERTYPE-ORDER.
     IF ORDERNUMBER-OPEN
         MOVE XML-TEXT       TO ORDERNUMBER-ORDER.
…
 401M-EXIT.
     EXIT.
```




## COBOL XML Parser statement

The next XML-EVENT is 'END-OF-ELEMENT' for '</**OrderType**>'.

- Back to paragraph 006-EVALUATE-STRUCTURE
- EVALUATE XML-EVENT

```
…
   WHEN 'END-OF-ELEMENT'
      ADD 1 TO END-OF-ELEMENT-COUNT
      PERFORM 120-END-OF-ELEMENT          THRU 120-EXIT
…
```

## COBOL XML Parser statement

The next XML-EVENT is 'END-OF-ELEMENT' for '</OrderType>'.

```
120-END-OF-ELEMENT.
   IF TRACE-PARSER THEN
      DISPLAY 'P-120 - Close Tag= </' XML-TEXT '>'
   END-IF.

   IF PRODUCTIONORDER-OPEN              THEN
      PERFORM 402-PRODUCTION-ORDER-CLOSE  THRU 402-EXIT
      GO TO 120-EXIT.

   IF PRODUCTIONORDERUPDATE-OPEN              THEN
      PERFORM 412-PRODUCTIONORDERUPDATE-CLOSE   THRU 412-EXIT
      GO TO 120-EXIT.

 120-EXIT.
     EXIT.
```




## COBOL XML Parser statement

The next XML-EVENT is 'END-OF-ELEMENT' for '</OrderType>'.

```
402-PRODUCTIONORDER-CLOSE.
  IF TRACE-PARSER THEN
     DISPLAY '-- 402-PRODUCTIONORDER-CLOSE.  '
  END-IF.
  EVALUATE XML-TEXT
     WHEN ORDERTYPE
       SUBTRACT 1 FROM OPEN-TAG-COUNT
       SET ORDERTYPE-CLOSE           TO TRUE
     WHEN ORDERNUMBER
       SUBTRACT 1 FROM OPEN-TAG-COUNT
       SET ORDERNUMBER-CLOSE   TO TRUE
…
     WHEN OTHER
       IF TRACE-PARSER THEN
          DISPLAY '402, UNEXPECTED XML TAG:<' XML-TEXT  '>.'
          END-IF
     END-EVALUATE.
 402-EXIT.
     EXIT.
```

## Processing XML Input
### Final programming thoughts - 1

- Closing a tag may also trigger an INSERT into a table
  - Like a tag that encompasses a group of other tags
- The XML PARSE statement has to complete processing the XML or XML-CODE can be set to -1 to stop parsing
  - Exiting the XML PARSE processing any other way will cause a program abend
- The program passes the XML to an SQL procedure to archive it before parsing
  - If anything goes wrong during the parsing process, the XML will already be saved




## Processing XML Input
### Final programming thoughts - 2

- XML represents a NULL as: “nil=true”
  - The parser does handle this and allows the program to process the “nil” identifier.

```
'<CancelledDate xsi:nil="true"/>'
```

- This parser implementation isn’t that complicated as it is long
- Every tag has to be processed, data and/or null, and that requires code

## Processing XML Input
### SQL Procedure definition

```
CREATE PROCEDURE IUA.XML_PARSER_00
  ( XML_INPUT               CHARACTER(29000) WITH DEFAULT,
    PARSER_MESSAGE          CHARACTER(500) WITH DEFAULT,
    PARSER_RETURN_CODE      INTEGER WITH DEFAULT,
    IUA_MESSAGE             CHARACTER(160) WITH DEFAULT,
    IUA_RETURN_CODE         INTEGER WITH DEFAULT,
    ENABLE_PARSER_TRACE     CHARACTER(1) WITH DEFAULT,
    ENABLE_SQL_TRACE        CHARACTER(1) WITH DEFAULT
  )
    EXTERNAL NAME IUACXML0
    PROTOCOL IDMS
    DEFAULT DATABASE NULL
    ESTIMATED ROWS 1
    SYSTEM MODE
    TRANSACTION SHARING OFF
    LOCAL WORK AREA 0  ;
```




## Processing XML Input
### SQL Procedure Execution

```
SELECT
   PARSER_MESSAGE
 , PARSER_RETURN_CODE
 , APPLICATION_MESSAGE
 , APPLICATION_RETURN_CODE
 , IUA_MESSAGE
 , IUA_RETURN_CODE
  FROM IUA.XML_PARSER_00
   WHERE  ENABLE_PARSER_TRACE = 'Y'
     AND  ENABLE_SQL_TRACE    = 'Y'
     AND  XML_INPUT =
      'XML text';
```

- The Trace (DISPLAY commands) go to the CA IDMS LOG

# Generating XML Output

## Generating XML Output

- Example 1 - Convert a print report to XML
  - This report accesses existing non-SQL data structures.
- Example 2 - Generate XML for transaction processing and data replication
  - The XML is built using SQL Procedures
  - SQL Procedure Language
  - These examples utilize SQL defined tables designed and populated just for this process
  - A data item can have three states: data, no data, or NULL

## Generating XML Output

Example 1 - Convert a print report to XML

- ***Requirement***
  - Convert a printed 1 page report to an XML structure
- ***SQL Tools Utilized:***
  - SQL Procedures – CA ADS and SQL procedure language
  - SQL Functions – CA ADS and SQL procedure language
  - SQL Table Procedure – COBOL
  - SQL intrinsic functions
  - SQL Views




## Generating XML Output

Example 1 - Convert a print report to XML

```
<ProductionOrder xmlns="http://sample.xml.schema">
  <OrderType>Custom</OrderType>
  <OrderNumber>2014-10-01-00001</OrderNumber>
  <ModelName>Tundra</ModelName>
  <ModelId>4781</ModelId>
  <MoreStuff>...</MoreStuff>
  <ResultMessage>Order processed</ResultMessage>
  <ResultCode>0</ResultCode>
  <ProductionRemarks>
   <Remark>Do NOT begin schedule production
until</Remark>
   <Remark>all the TRD options are
available.</Remark>
  </ProductionRemarks>
```

## Generating XML Output

Example 1 - Convert a print report to XML

```
<SpecialPartsList>
 <SpecialPart>
   <Type>Exterior</Type>
   <PartNum>1412</PartNum>
    <Name>Power rear window</Name>
   </SpecialPart>
 <SpecialPart>
   <Type>Exterior</Type>
   <PartNum>3813</PartNum>
   <Name>Power heated Mirrors</Name>
   </SpecialPart>
 <SpecialPart>
   <Type>Interior</Type>
   <PartNum>2254</PartNum>
   <Name>16 way adjustable driver seat</Name>
   </SpecialPart>
```




## Generating XML Output

Example 1 - Convert a print report to XML

```
 <SpecialPart>
   <Type>Interior</Type>
   <PartNum>2251</PartNum>
   <Name>2 way adjustable passenger seat</Name>
   </SpecialPart>
   <MoreStuff>...</MoreStuff>
 </SpecialPartsList>
</ProductionOrder>
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

- A starting point SELECT statement, then multiple nested SELECT statements to retrieve additional data
- The end user implementation executes the SQL command from an SQL procedure




## Generating XML Output

Example 1 - SQL syntax with XML Functions

- Begin the SQL command that includes the XML functions

```
CREATE VIEW IUA.ORDER_NUM_XML
  ( ORDER_NUM, XML_OUT )
  as
SELECT order_num,
   -- The XMLSERIALIZE begins the XML generation
XMLSERIALIZE(CONTENT
  XMLELEMENT(NAME "ProductionOrder",
    XMLNAMESPACES(default
       'xmlns="http://sample.xml.schema"'),
    XMLCOMMENT
      (  'Oct 1, 2014-Sample paper report to XML . '
      || 'This uses ordering a truck as an example. '
    ),  -- close the comment
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

- Begin listing the data elements that occur once / are based on the outermost

```
SELECT
XMLFOREST(ORDER_TYPE                    AS "OrderType"
         ,ORDER_Number                  AS "OrderNumber"
         ,RTRIM(Model_name)             AS "ModelName"
         ,Model_ID_NUMBER               AS "ModelId"
         ,substr(cast(current date as char(26)),1,4) ||
          substr(cast(current date as char(26)),6,2) ||
          substr(cast(current date as char(26)),9,2)
              AS "OrderDate"
```




## Generating XML Output

Example 1 - SQL syntax with XML Functions

- A user written SQL function converts the Plant_Code to the Plant_Name

```
,RTRIM(IUA.CONVERT_Plant(PLANT_CODE)) -- ADS SQL Function
              AS "PlantName"
  ...
 ,'Order processed'          AS "ResultMessage"
 ,'0' AS "ResultCode"
)       -- close the XMLFOREST
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

- Process an owner member structure. Note the nested SELECT

```
,XMLELEMENT(NAME "ProductionRemarks"
  ,select XMLAGG(                    -- < new SELECT
      XMLELEMENT(NAME "Remark",
            rtrim(Remark_text)
         OPTION ABSENT ON NULL
      )     -- close XMLELEMENT
   )     -- close XMLAGG
   FROM ORDRSCHM."Order" AS Order_RMK
      ,ORDRSCHM."REMARKS"
   WHERE
This inner SELECT gets a key from the outer SELECT
        outer_ORDER.ORDER_NUM =
        Order_RMK.ORDER_NUM
     AND "ORDER_REMARKS"
```




## Generating XML Output

Example 1 - SQL syntax with XML Functions

- Five options on how to document no remarks / handle a NULL indicator

```
OPTION NULL ON NULL
OPTION EMPTY ON NULL <- default
OPTION ABSENT ON NULL
OPTION NIL ON NULL          -- This is an single item
    '<Remark xsi:nil="true"/>'
OPTION NIL ON NO CONTENT    -- This handles a group
    '< ProductionRemarks xsi:nil="true"/>'
```

## Generating XML Output

Example 1 - syntax with XML Functions

- This SELECT is driven by a Table Procedure

```
,XMLELEMENT(NAME "SpecialPartsList"
  ,select XMLAGG(                    -- < new SELECT
      XMLELEMENT(NAME "SpecialPart",
         XMLFOREST(
            RTRIM(Part_TYPE)     AS "Type"
          , Part_Number          AS "PartNum"
          , RTRIM(Part_Name)     AS "Name"
         )  -- close XMLFOREST
         OPTION ABSENT ON NULL
      )    -- close XMLELEMENT
   )    -- close XMLAGG
   FROM IUA.SpecialPartTblProc AS SPECIAL_TBP
   WHERE
       SPECIAL_TBP.SPEC_NUM = SPECS_OUTER.SPEC_NUM
 )-- close the XMLELEMENT
```




## Generating XML Output

Example 1 - SQL syntax with XML Functions

- This ends the outer most SELECT

```
)            -- close the XMLELEMENT
 AS VARCHAR(15000)
)       -- this closes the XMLSERIALIZE
     AS "XML_OUT"
  FROM
     ORDRSCHM."ORDER-MODEL-JCT"
   , ORDRSCHM."Model"
   , ORDRSCHM."ORDER" as OUTER_ORDER
   , ORDRSCHM."MODEL-OPTIONS" as SPECS_OUTER
 ... Other records, views
WHERE "ORDER-MOD-JCT"
  AND "MODEL-ORD-JCT"
 ... other sets, constraints, etc     ;
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

- Execution
  - We always want to get back XML – even when the order number is not found
  - Reference the VIEW IUA.ORDER_NUM_XML in an SQL procedure
  - XML is always returned
  - This SQL view is compiled!




## Generating XML Output

Example 1 - SQL syntax with XML Functions

- **Sample SQL procedure to always return something**

```
SET OPTIONS COMMAND DELIMITER '++';
CREATE PROCEDURE IUA.ORDER_LOOKUP
    ( ORDER_NUM_IN            CHAR(6) WITH DEFAULT
    , ORDERS_FOUND            INTEGER  WITH DEFAULT
    , XML_OUT                 CHAR(15000) WITH DEFAULT
    , ORDER_LOOKUP_MSG        CHAR(80) WITH DEFAULT
    , ORDER_LOOKUP_RC         INTEGER  WITH DEFAULT
    )
             EXTERNAL NAME IUAORDL0
             LANGUAGE SQL
             PROTOCOL ADS
             DEFAULT DATABASE NULL
             SYSTEM MODE
             TRANSACTION SHARING ON
             LOCAL WORK AREA 1024
    BEGIN NOT ATOMIC
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

```
DECLARE XMLOUT_P1   CHAR(400);
  /* Determine if the order is valid.  */
SELECT COUNT(*)
   INTO ORDERS_FOUND
  FROM IUA.ORDER_NUM_XML
  WHERE ORDER_NUM = ORDER_NUM_IN
      ;
```

- My convention is to try and use the “COUNT(*)” function instead of SQLCODE = 100




## Generating XML Output

Example 1 - SQL syntax with XML Functions

```
IF ORDERS_FOUND = 0 THEN
    SET ORDER_LOOKUP_MSG = 'Order not found. Order_found
count=zero.' ;
    SET ORDER_LOOKUP_RC  = 2  ;
    EXEC ADS
        SNAP TITLE ' Order not found'. ;
    SET XMLOUT_P1 = '<ProductionOrder `
       || 'xmlns="http://sample.xml.schema">`
       || '<OrderNumber>`
       ||  ORDER_NUM_IN
       || '<ResultCode>2</ResultCode>'
       || '<ResultMessage>Order number not
found</ResultMessage>`
       || '</ProductionOrder>'
           ;
    SET XML_OUT = RTRIM(XMLOUT_P1) ;
END IF;
```

## Generating XML Output

Example 1 - SQL syntax with XML Functions

```
IF ORDERS_FOUND = 1 THEN
    EXEC ADS
         SNAP TITLE ' Order found = 1. Before SELECT'. ;
    SELECT XMLEXTRACT.XML_OUT
      INTO XML_OUT
      FROM IUA.ORDER_NUM_XML as xmlextract
     WHERE ORDER_NUM = ORDER_NUM_IN    ;
    SET ORDER_LOOKUP_MSG = 'Order found. Count=1.' ;
    SET ORDER_LOOKUP_RC  = 0  ;
    EXEC ADS
        SNAP TITLE ' Order found. After SELECT'. ;
END IF;
```




## Generating XML Output

Example 1 - SQL syntax with XML Functions

```
IF ORDERS_FOUND > 1 THEN
   EXEC ADS
    SNAP RECORD (SQLLOC0000IUAORDL0)
        TITLE ' Order found > 1  MAJOR PROBLEM'. ;
    SET XML_OUT =
    'SQL routine error. More than 1 row found. contact the
   DBA';
    SET ORDER_LOOKUP_MSG = 'SQL routine error. More than 1
   row.';
    SET ORDER_LOOKUP_RC  = 99 ;
    EXEC ADS
        SNAP RECORD (SQLLOC000IUAORDL0)
             TITLE ' Order found. After SELECT'. ;

END IF;

END       ++
```

# Generating XML Output Example 2

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- *Requirement*
  - Create and manage over 10 different XML outgoing transactions
  - The requesting application asks if any transaction is available – not knowing which type
  - One SQL request will return the oldest transaction ready to be processed
  - Create the XML from the data in the tables, archive the XML, and pass the XML to the requestor
  - Document the final status of the transaction, success or failure

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- ***Transaction processing structure:***
  - Other application programs populate the tables used to build the outgoing XML
  - All XML work is performed by SQL procedures written using the SQL procedure language
  - There is one driver or top level SQL procedure that is started by the end user through JDBC / CA IDMS Server
  - There are many SQL procedures to handle the different XML transactions and components of a transaction




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- SQL Tools Utilized:
  - SQL Procedures (Many) – CA ADS and SQL procedure language
    - **Transaction Sharing On**
  - SQL Functions –CA ADS and SQL procedure language
  - SQL intrinsic functions
  - SQL Views
  - SQL Tables to hold the outgoing data
- SQL WHENEVER
  - Use the WHENEVER to issue the SQL equivalent of PERFORM IDMS-STATUS and provide debugging information

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- SQL WHENEVER
  - Use the WHENEVER to issue the SQL equivalent of PERFORM IDMS-STATUS
  - Very helpful during development
  - Include columns like these to the SQL Procedure definition like:

```
ERROR_CODE        INTEGER,
ERROR_MESSAGE     CHAR(160),
ERROR_TRACE       CHAR(500)
```

  - At key points in the code, document where you are

```
SET ERROR_TRACE = 'Select TRANS_ID is not NULL.';
… more code
SET ERROR_TRACE =  ERROR_TRACE ||' COUNT the rows.';
```




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- SQL WHENEVER
  - Code something like this

```
DECLARE EXIT HANDLER FOR SQLWARNING
  LABEL_WARNING:
  BEGIN NOT ATOMIC
    SET ERROR_CODE = 512;
    SET ERROR_MESSAGE =
   'IUA_PROC_00: SQL Warning-SQLSTATE:' || SQLSTATE;
    SET ERROR_TRACE = 'Error_trace: ' || ERROR_TRACE;
END;
```

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

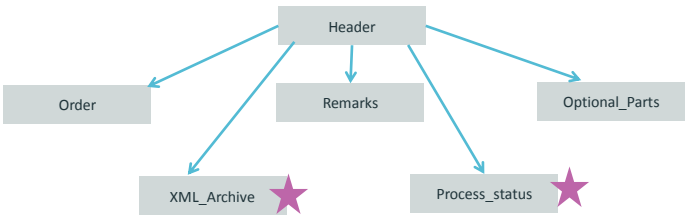
- ***Table structures***
  - Header / Parent table – holds data common to all transactions including a unique transaction identifier that is the primary key in other tables
  - Child tables defined to support the unique data requirements of each transaction
  - A table is used to archive the generated XML used for reconciliation, error research, and because we always keep everything
  - The same archive XML procedure that is used for incoming also handles outgoing




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- ***Sample table structures***




- ***The ★ identifies tables that are managed by an SQL Procedure with Transaction Sharing turned off.***

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- Beginning of the XML

```
create VIEW IUA.TRAN_HEADER (transaction_id, xml_header) as
   SELECT
       HEADER.TRANSACTION_ID,
   XMLSERIALIZE(CONTENT
       XMLELEMENT(NAME "IUA2Transaction"          ,
    --  XMLNAMESPACES information removed
    XMLELEMENT(NAME "IUA-com:Header",
     XMLFOREST( ACTIONTIME AS "IUA-com:ActionTime"
     ,rtrim(cast(TRANSID as char(20))) AS "IUA-com:TransactionId"
               ) ),  -- close the XMLFOREST, XMLELEMENT  HEADER
                 )   --  close the outermost XMLELEMENT
   AS VARCHAR(2000)  ) -- this closes the xmlserialize
      FROM IUA.HEADER , IUA.ORDER
   WHERE HEADER.TRANSACTION_ID  = ORDER.TRANSACTION_ID   ;
```




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- XML is built in pieces and spliced together
- Strip off the closing tag, to splice on at the end

```
set string_position =
    locate('</IUA2Transaction>', XML_HEADER );
set string_position_Out = string_position;
SET transaction_out =
rtrim(substr(XML_HEADER,1,string_position - 1));
```

- Splice it on when you are done

```
SET transaction_out = rTrim(transaction_out) ||
'</IUA2Transaction>' ;
```

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- Some data items have three "states" when sent out
  - Data exists and is sent with the XML tag
  - No data – nothing sent
  - Target system needs to NULL / clear the value. We send "nil=true" with the tag
  - We added an additional one character column with a "_N" suffix to allow us to identify when to send the "nil=true“




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

SQL Procedure code sample:

```
IF CANCELLEDDATE_N_O  IS NULL   THEN
    select xmlserialize(content
           xmlelement(NAME "CancelledDate"
         , cancelleddate_n option nil on null)
           as varchar(99))
      into attribute_text from IUA.product
     WHERE TRANSACTION_ID = HEADER_TRANS_ID  ;
     SET transaction_out =
           RTRIM(transaction_out) || RTRIM(ATTRIBUTE_TEXT);
ELSE
    select xmlserialize(content
           xmlelement(NAME "CancelledDate"
         , cancelleddate option ABSENT on null )
           as varchar(99))
      into attribute_text from IUA.product
     WHERE TRANSACTION_ID = HEADER_TRANS_ID  ;
     SET transaction_out =
           RTRIM(transaction_out) || RTRIM(ATTRIBUTE_TEXT);
END IF;
```

## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

SQL Procedure code sample:

```
IF CANCELLEDDATE_N_O  IS NULL   THEN
    Set attribute_text = '<CancelledDate xsi:nil="true"/>';
    SET transaction_out =
           RTRIM(transaction_out) || RTRIM(ATTRIBUTE_TEXT);
ELSE
    select xmlserialize(content
           xmlelement(NAME "CancelledDate"
         , cancelleddate option ABSENT on null )
           as varchar(99))
      into attribute_text from IUA.product
     WHERE TRANSACTION_ID = HEADER_TRANS_ID  ;
     SET transaction_out =
           RTRIM(transaction_out) || RTRIM(ATTRIBUTE_TEXT);
END IF;
```




## Generating XML Output

Example 2 - Generate XML for transaction processing and data replication

- ***Large subcomponents of XML are more easily handled in their own SQL procedures***
- ***Transaction Sharing:***
  - All the XML building SQL procedures utilize the TRANSACTION SHARING functionality
  - CA IDMS SQL Programming Guide has the most detailed definition of Transaction Sharing
  - This will be discussed more in the next presentation

## Summary – My Observations

- XML Parsing is more tedious, than difficult
  - COBOL data division
  - Don't be surprised if the first attempt doesn't work out
- XML Output
  - The XML functions work, and work well with SQL and non SQL data
- SQL Procedures
  - Great for creating reusable / sharable routines
- SQL Procedure language
  - Big fan
  - I build and manage the source in TSO




## Online Session Evaluation

Please provide your feedback about this session: A5

On the CA Communities web site: http://communities.ca.com

More details in your conference bag

# Questions and Answers