Redvers Consulting Ltd

Redvers COBOL XML Interface

User Guide

Standalone Generator

RCIMSXML Version 2.8

Contents

Preface

This document describes the installation and operation of the Redvers COBOL XML Interface program RCIMSXML. It is designed for use by Information Technology departments familiar with the COBOL and XML computer languages.

The program generates XML documents for COBOL batch and on-line applications using a user defined COBOL Record Definition (CRD) without the need for any external file access. The generated well-formed XML standalone document conforms to the World Wide Web Consortium (W3C) Extensible Markup Language (XML) 1.0 (Second Edition) definition.

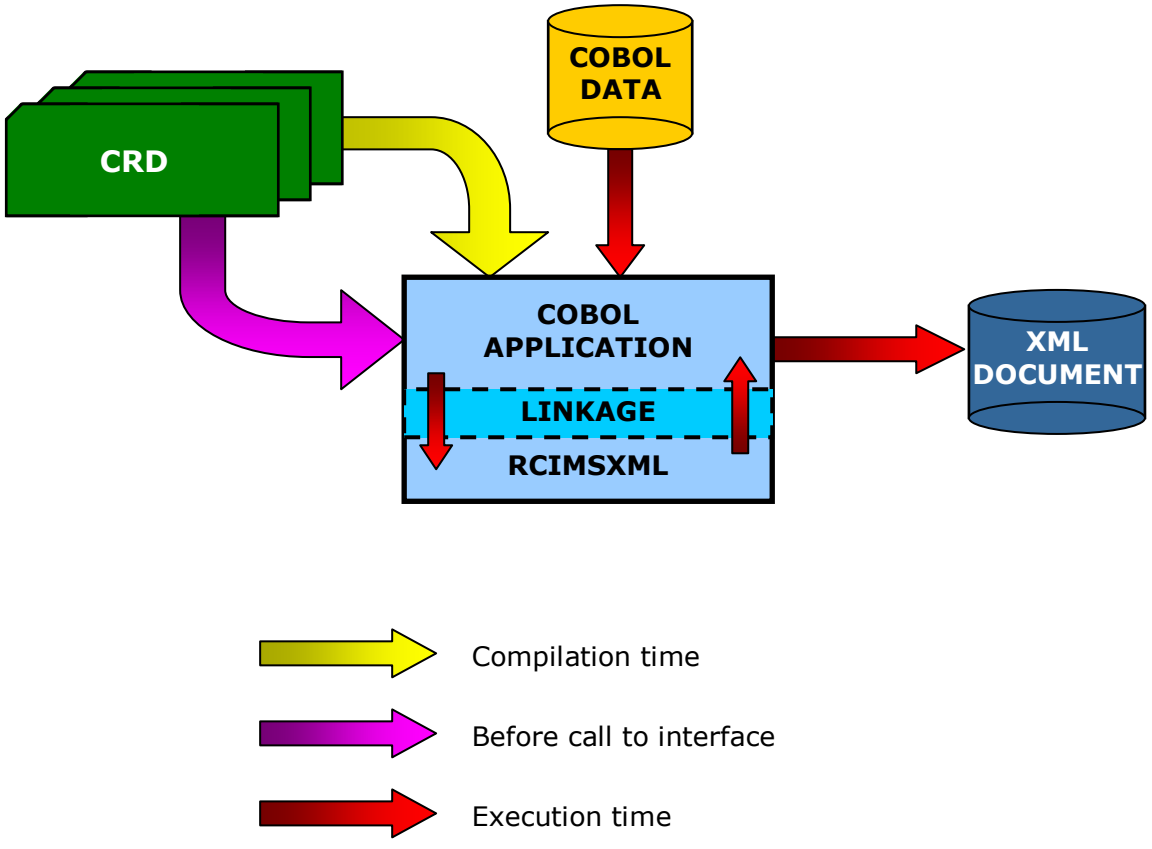
RCIMSXML is the counterpart to RCXMLIMS which uses a similar process to parse XML documents, returning the data in the form of a COBOL record.

This User Guide can be found on the internet at:
http://www.redversconsulting.com/downloads/user_guides/RCIMSXML_2.8_user.pdf

Overview

RCIMSXML is a COBOL subroutine that is compiled and linked into application programs in order to generate well-formed XML documents. This particular interface module was built with on-line IMS users in mind so that they could call a "logic only" routine that required no external file access. However, this version can be used in any application environment and is frequently a component of an Enterprise Service Bus (ESB) in Service Oriented Architecture (SOA) projects.

Programs that call RCIMSXML are compiled with a user defined COBOL Record Definition (CRD) copybook member describing the format of the COBOL record in linkage. The COBOL source code for the CRD must also be loaded into the application program's storage before the first call to RCIMSXML. This can be done by hard coding the source into working storage (Redvers Consulting's RCIMSCRD tool can be used for this) or by reading the CRD source records into a working storage table (shown below). At execution time, the application populates the COBOL record held in linkage and passes it to the interface routine along with the CRD source code. RCIMSXML then converts the data in the COBOL record to an XML document and returns it to the application.



Installation

RCIMSXML is a self-contained source program that requires no copybooks or objects. Before the product was despatched, the source code was passed through the Redvers COBOL Cloaking Device which removes the intellectual property within the source code without changing the logic.

To install RCIMSXML, copy the "cloaked" source code into your source code library and paste the activation key (sent in an email when ordering/downloading) over the question marks in the last **VALUE** clause in **WORKING-STORAGE**. The program can then be compiled using your standard COBOL compiler.

When running a free 30 day trial, the sample calling program RCIXCALL can also be copied to your source code library, compiled and linked. Under normal circumstances RCIMSXML will be called from your COBOL application.

If your site uses single quotes (apostrophes) rather than double quotes (speech marks) to delimit literals, a global change of all double quotes to single quotes can be made. However, following the change, any single quotes within the XML declaration must be changed to double quotes before compilation. The XML declaration can be found at the start of **WORKING-STORAGE** just above a comment line containing: "**<UMV> XML-declaration**".

Various parameters, including the **PROGRAM-ID**, are defined as User Maintained Variables in the source code so that they may be set to alternative values if desired. See User Maintained Variables for details.

If you should encounter any problems during installation, please contact your account manager or use our "Contact" facility at: http://www.redversconsulting.com/contact.php.

Coding the COBOL Record Definition

The COBOL Record Definition (CRD) defines the layout of the COBOL record passed to the generator subroutine (RCIMSXML). This CRD is also used to define element and attribute tag names, data formatting and general structure of the XML document to be generated. Redvers Consulting offers two free tools to assist in drafting a CRD. These tools require either an XML W3C schema or Document Type Definition (DTD) as input and can be provided by your account manager or downloaded from the Partners area of our website.

Field Names

Default Tag Names

The COBOL field names in the CRD are used as the default XML start/end tags and the data in the field becomes the XML element content.

Eg:

Field in CRD	Content
`03 TV-program PIC X(20).`	`"Sunday Night Live "`

Generates:

XML Document
<TV-program>Sunday Night Live</TV-program>

Note: *Upper/lower case settings in the tag/field name are preserved.*

If there is no data in a field, the default result is for an empty element to be generated (see Empty Fields for details on how empty fields are recognised).

Eg:

Field in CRD	Content
`03 TV-program PIC X(20).`	`spaces`

Generates:

XML Document
<TV-program/>

Override Tag Names

Because COBOL field names are subject to COBOL compiler rules such as character sets and the maximum length of a field name, the default tag may be overridden by coding an alternate name within "<" and ">" in the comment line or lines immediately following the field definition.

Eg:

Field in CRD	Content

`03 TV-PROGRAM PIC X(20).`
`* <The_next_program_on_`
`* Channel-5_>`
	`"Sunday Night Live "`

Generates:

XML Document
<The_next_program_on_Channel-5_>Sunday Night Live</The_next_program_on_Channel-5_>

XML Attributes

If an XML attribute is required, this can be achieved by coding an "=" at the end of an override tag name in a subordinate field immediately after the group definition.

Eg:

Fields in CRD	Content
`03 TV-program.`	
`05 prog-time PIC X(5).` `* <time=>`	`"10:00"`
`05 prog-name PIC X(20).`	`"Sunday Night Live "`

Generates:

XML Document
<TV-program time="10:00"> <prog-name>Sunday Night Live</prog-name> </TV-program>

XML Namespaces and SOAP

The interface supports the generation of XML namespace declarations and Simple Object Access Protocol (SOAP) Envelopes using override tag names. Each namespace prefix is defined in the override tag name and the HTTP references are placed in each field's content.

Eg:

Fields in CRD	Content
03 TV-ENVELOPE. * <SOAP-ENV:Envelope>	
05 SOAP-NAMESP PIC X(34). * <xmlns:SOAP-ENV=>	"http://schemas.xmlsoap.org/soap "
05 UK-NAMESPACE PIC X(34). * <xmlns:uk=>	"http://www.greenwichmeantime.co.uk "
05 US-NAMESPACE PIC X(34). * <xmlns:us=>	"http://www.easternstandardtime.com "
05 TV-BODY. * <SOAP-ENV:Body>	
07 prog-name PIC X(20).	"Sunday Night Live "
07 PR-UK-TIME PIC X(05). * <uk:time>	"10:00"
07 PR-US-TIME PIC X(05). * <us:time>	"05:00"

Generates:

XML Document
< SOAP-ENV:Envelope xmlns:SOAP-ENV=" http://schemas.xmlsoap.org/soap" xmlns:uk="http://www.greenwichmeantime.co.uk" xmlns:us="http://www.easternstandardtime.com"> <SOAP-ENV:Body> <prog-name>Sunday Night Live</prog-name> <uk:time>10:00</uk:time> <us:time>05:00</us:time> </SOAP-ENV:Body> </SOAP-ENV:Envelope>

PICTURE Clause

The picture clause in the CRD uses standard COBOL data editing features to control how the data content of an XML element is formatted.

The interface also uses the picture clause to determine if the field is to be left or right justified, removing the appropriate leading/trailing spaces or zeroes. However, it will not remove blank characters if they are defined in the picture clause.

Eg:

Fields in CRD	Content
03 TV-program.	
05 prog-date PIC 99/99/9999.	31122002
05 prog-name PIC BBX(20)BB.	"Sunday Night Live "
05 prog-time PIC X(5).	"10:00"
05 prog-duration PIC ZZZ.999.	0.333
05 prog-rating PIC S99.	5
05 prog-cost PIC $$$,$$$,$$9.99DB.	-2500000

Generates:

XML Document
<TV-program> <prog-date>31/12/2002</prog-date> <prog-name> Sunday Night Live </prog-name> <prog-time>10:00</prog-time> <prog-duration>.333</prog-duration> <prog-rating>0E</prog-rating> <prog-cost>$2,500,000.00DB</prog-cost> </TV-program>

Note: *The imbedded sign in the* ***prog-rating*** *picture clause remains imbedded in the XML element content – not normally the desired result. A picture clause containing "-" or "+" (or the* ***SIGN TRAILING/LEADING SEPARATE*** *clause) would produce more readable XML.*

Binary / Packed Fields

Due to the fact that XML is a character based language, binary and packed decimal numeric data cannot be safely represented within element tags. For this reason binary and packed decimal data is converted to a format known as base64. This format produces a character range of "A" through "Z", "a" through "z", "0" through "9", "+", "/" and "=". This subset of ISO 646 characters is represented identically in all versions of ASCII and in all versions of EBCDIC, which ensures a smooth translation from ASCII to EBCDIC and vice-versa.

Eg:

Fields in CRD	Content
03 prog-rating PIC 99 PACKED-DECIMAL.	8
03 prog-revenue PIC S9(7)V99 BINARY.	1234567.89

Generates:

XML Document
<prog-rating>AI8=</prog-rating> <prog-revenue>B1vNFQ==</prog-revenue >

Note: *On Little-endian platforms the base64 characters for* ***prog-revenue*** *will actually be "Fc1bBw==" due to the different way binary values are stored.*

Other Clauses

Data definition clauses can be used to edit the appearance of XML data within an element in just the same way they are used in COBOL. Some useful clauses are listed below:

BLANK WHEN ZERO

If the content of the field is zero this clause will result in the generation of an empty XML element.

JUSTIFIED RIGHT

This clause will cause the suppression of spaces to the left of text rather than to the right in the XML element data area.

OCCURS

This clause can be used to create a single dimension array of repeating data to be written to the XML document. It can be used at group or elementary level. Unpopulated occurrences within a populated array are generated as empty elements but trailing occurrences with no data are entirely suppressed from the document.

Arrays of more than one dimension are generated by issuing multiple calls to the interface subroutine – see Repeating Groups for details. This approach allows the interface to generate an unlimited number of occurrences in unlimited dimensions (which is the case for XML documents) using only a small amount of computer storage.

An `OCCURS 1` clause can be used to suppress optional elements entirely from the XML document, if there is no data in a field – see Optional Elements for details. The use of `OCCURS 1` does not constitute an additional dimension in an array.

Clauses Not Supported

The following data definition clauses are not currently supported in the CRD:

`OCCURS DEPENDING ON`
`REDEFINES`
`SYNCHRONIZED/SYNC`
Symbol "**`P`**" in the picture clause

Structure

Just as COBOL has a hierarchical structure in the relationship between group and elementary fields in a data record, XML has hierarchy between parent and child elements in an XML document. The interface uses the COBOL hierarchy to nest XML tags and data so that element relationships can be constructed. All XML documents must start with a root element and this root element corresponds to the top level COBOL field in the CRD. Similarly, all lower COBOL levels are used to generate child XML elements within the root parent.

Eg:

Fields in CRD		Content
01 TV-listings.		
03 broadcast-date	PIC 99/99/9999.	11122002
03 channel.		
05 channel-number	PIC 99.	5
05 TV-program.		
07 prog-name	PIC X(20).	"Sunday Night Live "
07 prog-time	PIC X(5).	"10:00"
07 prog-rating	PIC Z9.	8

Generates:

XML Document
<TV-listings> <broadcast-date>11/12/2002</broadcast-date> <channel> <channel-number>05</channel-number> <TV-program> <prog-name>Sunday Night Live</prog-name> <prog-time>10:00</prog-time> <prog-rating>8</prog-rating> </TV-program> </channel> </TV-listings>

Mixed Content Elements

When an XML element is a mixed content element (i.e. it contains data *and* other subordinate elements), it needs to be logically partitioned across a COBOL group definition. This type of generation is achieved using a group level data name with subordinate non-tagged fields. Non-tagged fields are either defined using **FILLER** or a null override tag name in the CRD (**<>**).

Eg:

Fields in CRD	Content
03 TV-program. 05 FILLER PIC X(20). 05 prog-time PIC X(5). 05 END-TEXT PIC X(10). * <>	 "Sunday Night Live " "10:00" " tomorrow"

Generates:

XML Document
<TV-program>Sunday Night Live <prog-time>10:00</prog-time> tomorrow </TV-program>

Non-tagged group level fields

Non-tagged group level fields can be used in situations when fields need to be logically linked under a COBOL group level but the group level element itself is not required in the XML structure.

Eg:

Fields in CRD	Content
01 TV-listings. 03 TV-program. * <> 05 prog-name PIC X(20). 05 prog-time PIC X(5).	 "Sunday Night Live " "10:00"

Generates:

XML Document
<TV-listings> <prog-name>Sunday Night Live</prog-name> <prog-time>10:00</prog-time> </TV-listings>

Advanced Techniques

Formatting

By using override tag names, non-tagged fields, attributes and picture field editing together, a variety of formats can be generated:

Eg:

Fields in CRD	Content
01 TV-listings.	
03 broadcast-date PIC 99/99/9999.	11122002
03 channel.	
05 channel-number PIC 99. * <number=>	5
05 TV-program.	
07 prog-name PIC BBX(20)BB. * <>	"Sunday Night Live "
07 prog-time PIC X(5).	"10:00"
07 prog-rating.	
09 FILLER PIC X(20)B.	"I gave it "
09 mark PIC Z9.	8
09 out-of-text PIC BX(20). * <>	"out of ten "

Generates:

XML Document
<TV-listings> <broadcast-date>11/12/2002</broadcast-date> <channel number="05"> <TV-program> Sunday Night Live <prog-time>10:00</prog-time> <prog-rating>I gave it <mark>8</mark> out of ten </prog-rating> </TV-program> </channel> </TV-listings>

Required Attributes and Elements

Under normal circumstances, an attribute will only be generated if the corresponding CRD field contains data (see Empty Fields for details of how the interface identifies a field as containing data or as being empty). However, DTD or schema validation rules may require the presence of an attribute even though there is no data to report. Required attributes would be defined in a DTD with the **#REQUIRED** parameter or in an XML schema with **use="required"**.

The introduction of release 2.5 enables the forced generation of *required* XML attributes by the coding of double "<" and ">" marks surrounding the override tag name in the CRD. Whenever the parent of a *required* attribute is generated, the interface will also generate the *required* attribute regardless of content.

Elements can also be marked as *required* using the double "<" and ">" marks, even though elements are generated by default anyway. The effect of defining empty elements as *required* causes the generation of separate start and end tags **(<tagname></tagname>)** rather than the usual empty tag **(<tagname/>).** For numeric elements, this may place zeroes in the element content, depending on the picture clause.

Eg:

Fields in CRD	Content
03 channel.	
05 channel-number PIC 99. * <<number=>>	00
05 TV-program.	
07 prog-name PIC X(20).	"Sunday Night Live "
07 prog-time PIC X(5).	spaces
07 prog-rating.	
09 PROG-MARK PIC Z9. * <<mark>>	00

Generates:

XML Document
<channel number="00"> <TV-program> <prog-name>Sunday Night Live</prog-name> <prog-time/> <prog-rating> <mark>0</mark> </prog-rating> </TV-program> </channel>

Optional Elements

As mentioned in the Occurs section of Other Clauses, trailing occurrences in a CRD array that are empty, are suppressed from the XML document. Therefore, using the same logic, an **OCCURS 1** clause can be used to suppress any element entirely from the XML document if it is deemed to be empty (see Empty Fields for details of how the interface identifies empty fields). The interface treats fields defined with **OCCURS 1** as it would the final occurrence of any multiply occurring field and will therefore not generate the element, if it is empty.

The **OCCURS 1** clause can therefore be coded on all optional elements, at group or elementary level, in order to remove them from the XML document when there is no data to report. Optional elements would be defined with an occurrence indicator of "?" or "*" in a DTD or with **minOccurs="0"** in an XML schema.

Eg:

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	"Sunday Night Live "
07 prog-time OCCURS 1	PIC X(5).	spaces
07 prog-rating OCCURS 1.		
09 PROG-MARK	PIC Z9.	00
* <mark>		

Generates:

XML Document
<channel> <TV-program> <prog-name>Sunday Night Live</prog-name> </TV-program> </channel>

Note: *If the use of OCCURS 1 causes multiple value 1 subscripts in the application, these subscripts can be avoided by placing a "**REPLACING == OCCURS 1.== BY ==.==**" on the **COPY** statement for the CRD.*

Note2: *From release 2.8, it is possible to make all elements optional, by default, removing the need for OCCURS 1 in the CRD. This processing can be switched on by the use of a special processing flag set in your copy of RCIMSXML. For more information, please contact your account manager or use our "Contact" facility at:* http://www.redversconsulting.com/contact.php.

Excluded Elements

A COBOL Record Definition (CRD), used by the Redvers COBOL XML Interface is frequently in the form of a COBOL "copybook" or "include" and therefore may be used by an application for a variety of purposes, outside the calling of the Redvers XML generator routine. As a result, there may be fields in the CRD which relate to application processes outside the Redvers COBOL XML Interface and therefore should not result in the generation of an XML element for such a field.

From release 2.7, this issue has been addressed by providing a new type of CRD override tag name, within parentheses, in the form: **<(tagname)>**. Under these circumstances, the entry is still defined as a COBOL field and can still be used by the application for other purposes but no XML will be produced for this field, regardless of its content. In effect, the tag name specified becomes useful for documentation purposes only.

An example of an excluded field in a CRD can be seen below:

Field in the CRD
`03 TV-PROGRAM PIC X(20).` `* <(Exclude_Me)>`

Note: *Excluded fields must still be included in the total length of the CRD when populating the* ***COBOL-RECORD-LENGTH*** *parameter, otherwise a* ***FEEDBACK-CODE*** *of +0110 will be returned.*

XML Declaration Override

The default XML document declaration (eg: "<?xml version='1.0' encoding='UTF-8'?>") is defined by the XML-declaration User Maintained Variable (See User Maintained Variables for details), coded near the start of **WORKING-STORAGE**. As there is only one default value for each interface generator subroutine, this can be a limitation for applications requiring different declarations for each XML document type.

To override the default declaration, a comment line must be coded before any field definition in the CRD, starting with the characters "**<?xml**". This comment string will then replace the default declaration at the start of the output XML document.

Eg:

CRD

```
***  Start of COBOL Record Definition (CRD)
* An override XML declaration follows:
*<?xml version='1.0' encoding='UTF-9999' standalone='no'?>
 01  TV-listings.
     03  channel.
…
```

Generates:

XML Document

```
<?xml version="1.0" encoding "UTF-9999" standalone="no"?>
<!-- This XML document was generated by RCIMSXML -->
<TV-listings>
 <channel>
…
```

Repeating Groups

In business applications, it would be rare for an XML document to contain only a single set of information details. Elements and element groups are often repeated to reflect multiple sets and subsets of information. In order to generate this repeating data in XML, single dimension arrays can be defined in the CRD using the COBOL OCCURS clause. Alternatively, when the number of occurrences is unknown or more than one dimension of repeating data is present, multiple calls can be made to RCIMSXML and only the changed information will be generated in XML. Using the latter technique, XML documents up to 99MB in length can be generated.

Using OCCURS

A single dimension array of repeating information can be passed to RCIMSXML in a CRD table which results in the generation of multiple sets of XML elements until all remaining occurrences in the CRD are unpopulated.

Eg:

Fields in CRD		Occurr-ence	Content
03 channel.			
05 channel-number	PIC 99.		5
05 TV-program	OCCURS 5.	1	
07 prog-name	PIC X(17).	1	"Sunday Night Live"
07 prog-time	PIC X(5).	1	"10:00"
07 prog-rating	PIC Z9 OCCURS 1.	1	8
05 TV-program	OCCURS 5.	2	
07 prog-name	PIC X(17).	2	"News "
07 prog-time	PIC X(5).	2	"11:30"
07 prog-rating	PIC Z9 OCCURS 1.	2	0
05 TV-program	OCCURS 5.	3	
07 prog-name	PIC X(17).	3	spaces
07 prog-time	PIC X(5).	3	spaces
07 prog-rating	PIC Z9 OCCURS 1.	3	0
05 TV-program	OCCURS 5.	4	
07 prog-name	PIC X(17).	4	"Weather "
07 prog-time	PIC X(5).	4	"11:55"
07 prog-rating	PIC Z9 OCCURS 1.	4	0
05 TV-program	OCCURS 5.	5	
07 prog-name	PIC X(17).	5	spaces
07 prog-time	PIC X(5).	5	spaces
07 prog-rating	PIC Z9 OCCURS 1.	5	0

Generates:

XML Document
<channel> <channel-number>05</channel-number> <TV-program> <prog-name>Sunday Night Live</prog-name> <prog-time>10:00</prog-time> <prog-rating>8</prog-rating> </TV-program> <TV-program> <prog-name>News</prog-name> <prog-time>11:30</prog-time> </TV-program> <TV-program/> <TV-program> <prog-name>Weather</prog-name> <prog-time>11:55</prog-time> </TV-program> </channel>

Note: *The* ***<prog-rating>*** *field and the fifth occurrence of* ***<TV-program>*** *are suppressed from the document when they are not populated but the third occurrence of* ***<TV-program>*** *is generated because subsequent occurrences exist.*

Using Repeated Calls

An unlimited number of occurrences and dimensions (which is the case for XML documents) can be generated most efficiently by the use of repeated calls to the interface generator module. For each successive call, elements are generated for the lowest level covering all changed fields in the CRD along with the minimum of related parent and child elements necessary to maintain the data relationships before and after the change. These repeated sets of related data items are collectively known to the program as the Point of Change (POC) for that call.

The example on the next page generates a two dimensional array in XML for multiple **<TV-program>** elements within multiple **<channel>** elements using repeated calls and a smaller CRD.

Eg: The first call:

Fields in CRD	Content
03 channel.	
05 channel-number PIC 99.	5
05 TV-program.	
07 prog-name PIC X(20).	"Sunday Night Live "
07 prog-time PIC X(5).	"10:00"

The second call:

Fields in CRD	Content
03 channel.	
05 channel-number PIC 99.	5
05 TV-program.	
07 prog-name PIC X(20).	"News "
07 prog-time PIC X(5).	"11:30"

The third call:

Fields in CRD	Content
03 channel.	
05 channel-number PIC 99.	6
05 TV-program.	
07 prog-name PIC X(20).	"Westenders "
07 prog-time PIC X(5).	"08:00"

Generates:

XML Document

```
<channel>
 <channel-number>05</channel-number>
 <TV-program>
  <prog-name>Sunday Night Live</prog-name>
  <prog-time>10:00</prog-time>
 </TV-program>
 <TV-program>
  <prog-name>News</prog-name>
  <prog-time>11:30</prog-time>
 </TV-program>
</channel>
<channel>
 <channel-number>06</channel-number>
 <TV-program>
  <prog-name>Westenders</prog-name>
  <prog-time>08:00</prog-time>
 </TV-program>
</channel>
```

Controlling the Point of Change

In order to provide a facility for the calling application to control the POC and force the regeneration of unchanged XML elements higher in the record hierarchy, a technique using dummy fields can be applied. When RCIMSXML reads a CRD, it converts any low-value characters in non-binary fields to spaces as it stores the information internally. Low-values remain in the application's CRD but the internal image of the data passed now contains spaces instead. The result of this process causes RCIMSXML to regard the dummy field as a changed field and therefore it generates additional XML to reflect this implied change. Of course, no XML is generated for the dummy field itself (having no tag and no data) but XML is generated for the parent of the dummy field. This implied change will continue for all subsequent calls, until low-values are removed from the dummy field by the application program.

In the example below, the dummy field in **TV-today** forces the second call to consider **TV-today** as the POC. Consequentially XML is generated for the entire **TV-today** group even though only **prog-name** and **prog-time** have changed.

Eg:

The first call:

Fields in CRD			Content
01 TV-listings.			
03 TV-today.			
05 Dummy-field	PIC X.		Low-values
* <>			
05 broadcast-date	PIC 99/99/9999.		11122002
05 channel.			
07 channel-number	PIC 99.		5
07 TV-program.			
09 prog-name	PIC X(20).		"Sunday Night Live "
09 prog-time	PIC X(5).		"10:00"

The second call:

Fields in CRD			Content
01 TV-listings.			
03 TV-today.			
05 Dummy-field	PIC X.		Low-values
* <>			
05 broadcast-date	PIC 99/99/9999.		11122002
05 channel.			
07 channel-number	PIC 99.		5

```
       07  TV-program.
         09  prog-name      PIC X(20).      "News                "
         09  prog-time      PIC X(5).       "11:30"
```

Generates:

XML Document

```
<TV-listings>
 <TV-today>
  <broadcast-date>11/12/2002</broadcast-date>
  <channel>
   <channel-number>05</channel-number>
   <TV-program>
    <prog-name>Sunday Night Live</prog-name>
    <prog-time>10:00</prog-time>
   </TV-program>
  </channel>
 </TV-today>
 <TV-today>
  <broadcast-date>11/12/2002</broadcast-date>
  <channel>
   <channel-number>05</channel-number>
   <TV-program>
    <prog-name>News</prog-name>
    <prog-time>11:30</prog-time>
   </TV-program>
  </channel>
 </TV-today>
</TV-listings>
```

If there is a possibility that the CRD could be populated with exactly the same data in consecutive calls (duplicate records on the input), a similar technique can be used to generate a minimum of XML regardless of the COBOL input. This is done by placing a non-tagged dummy field, populated with low-values, just below the group level to be used as the minimum POC.

Orphan Repeats

Some XML documents require elements or groups of elements to repeat without the presence of a parent element for each occurrence. This type of generation is achieved using a non-tagged group level. Non-tagged group level fields are either defined using **FILLER** or a null override name in the CRD.

Eg:

The first call:

Fields in CRD			Content
03	channel.		
05	channel-number	PIC 99.	05
05	TV-program		
*	<>		
07	prog-name	PIC X(20).	"Sunday Night Live "
07	prog-time	PIC X(5).	"10:00"

The second call:

Fields in CRD			Content
03	channel.		
05	channel-number	PIC 99.	05
05	TV-program		
*	<>		
07	prog-name	PIC X(20).	"News "
07	prog-time	PIC X(5).	spaces

The third call:

Fields in CRD			Content
03	channel.		
05	channel-number	PIC 99.	05
05	TV-program		
*	<>		
07	prog-name	PIC X(20).	"Weather "
07	prog-time	PIC X(5).	"11:55"

Generates:

XML Document
<channel> <channel-number>05</channel-number> <prog-name>Sunday Night Live</prog-name> <prog-time>10:00</prog-time> <prog-name>News</prog-name> <prog-time/> <prog-name>Weather</prog-name> <prog-time>11:55</prog-time> </channel>

Note: *The non-tagged* ***TV-program*** *group is the POC even though it doesn't appear on the XML document.*

Calling RCIMSXML

Parameters

A call to RCIMSXML requires eight parameters in the following sequence:

CRD-SOURCE-AREA (input)

This is the area of storage within the calling application containing the contiguous series of 80 byte logical records populated with the COBOL Record Definition (CRD) source code. This area of storage can either be defined empty and then loaded by the application from the CRD source copybook or it can be defined with the CRD hard-coded into it using the **VALUE** clause. To assist with the latter option, Redvers has built a simple tool (RCIMSCRD) which can be downloaded for free from the Partners area of our web site: http://www.redversconsulting.com/partners.php

CRD-RECORD-COUNT (input)

This S9(8) binary field must be set to the number of 80 byte logical records in **CRD-SOURCE-AREA**. If this field is set to zero RCIMSXML will attempt to use the previous **CRD-SOURCE-AREA** to process the CRD (see Generating Multiple XML Documents for more details).

COBOL-RECORD (input)

This is the top level field name in the CRD. It will hold the COBOL records passed to RCIMSXML.

COBOL-RECORD-LENGTH (input)

This S9(8) binary field must be set to the logical length of **COBOL-RECORD**. It is used in validation only, to ensure **COBOL-RECORD** reflects the CRD in **CRD-SOURCE-AREA**.

XML-DOCUMENT (output)

This is the field name of the XML document area capable of holding the entire XML document to be generated.

XML-DOCUMENT-LENGTH (input & output)

This S9(8) binary field must be set to the length of the **XML-DOCUMENT** field and not changed during the generation process. When RCIMSXML has completed building the document, it will change this value to reflect the actual XML text length of the newly built document. The maximum XML document length supported by the interface is 99,999,999 characters.

FEEDBACK-CODE (output)

This S9(4) binary field is set by RCIMSXML to return the status of a call to the interface. This field should be checked for non-zero values after each call. If the call was successful this field will be set to zero otherwise it will contain an error code.

See Feedback Messages at the end of this document for further information.

FEEDBACK-TEXT (output)

This eighty byte text field is set by RCIMSXML with diagnostic information on the results of each call. For the first successful call this area contains CRD parsing information, for subsequent calls it contains the number of XML lines generated so far and the call count. For unsuccessful calls it contains an error message.

See Feedback Messages at the end of this document for further information.

Note: T*he parameter names used in this manual are suggestions only and may be changed to names more suitable to the application making the call.*

Calls to RCIMSXML

The First Call

When the interface is called for the first time, RCIMSXML will validate and store the CRD source code information in **CRD-SOURCE-AREA**.

If a Document Type Definition (DTD) is required at the start of the XML document, the first call to RCIMSXML must be made with the characters "DTD" in position one of **COBOL-RECORD** followed by spaces. **COBOL-RECORD-LENGTH** must be set to the full logical length of the CRD. If a Document Type Definition (DTD) is not required, the first call to RCIMSXML is made in the normal way - with a fully populated **COBOL-RECORD** as defined in the CRD.

Once processing is complete, **FEEDBACK-CODE** is set to zero and **FEEDBACK-TEXT** is populated with details of the stored CRD.

Subsequent Calls

Calls should be made to RCIMSXML (while **FEEDBACK-CODE** remains zero) with a fully populated **COBOL-RECORD** until the application data is exhausted.

After successful completion of each call, **FEEDBACK-CODE** is set to zero and **FEEDBACK-TEXT** is populated with the number of calls made and number of XML lines generated so far.

The Last Call

After all the application data has been passed to RCIMSXML, a final call must be made to allow the interface to complete the XML document. This final call is done, either by moving **HIGH-VALUES** to **COBOL-RECORD** or by moving **ZERO** to the **COBOL-RECORD-LENGTH** parameter – either option will produce the same result.

After successful completion of this call, **XML-DOCUMENT-LENGTH** is set to the actual length of the generated XML document, **FEEDBACK-CODE** is set to zero and **FEEDBACK-TEXT** is populated with the total number of calls made and total number of XML lines generated.

Note: *If more than one XML document is to be generated, the parameters can be repopulated and processing restarted from the first call – see* Generating Multiple XML Documents for more details).

Sample Program Calling RCIMSXML

```
...
000220***  Start of COBOL Record Definition (CRD)
000230 01  TV-listings.
000240     03  channel.
000250         05  channel-number      PIC 99.
000260*            <number=>
000270         05  channel-name        PIC X(20).
000280*            <>
000290         05  TV-program          OCCURS 10.
000300             07  prog-name       PIC BBX(20)BB.
000310             07  prog-time       PIC X(05).
000320             07  prog-rating     OCCURS 1.
000330*                <rating>
000340                 09  rating-txt1 PIC X(20)B.
000350*                    <>
000360                 09  mark        PIC Z9.
000370                 09  rating-txt2 PIC BX(20).
000380*                    <>
000390         05  channel-owner       PIC X(20).
000400*            <>
000410***  End of COBOL Record Definition (CRD)
000420
000430***  Start of the CRD SOURCE AREA
000440* This sample program defines the CRD source using VALUE clauses,
000450* alternatively, the area can be loaded from an external file.
000460 01  TV-LISTINGS-SOURCE.
000470     03  FILLER                  PIC X(80) VALUE
000480     "000000 01  TV-listings.                                    ".
000490     03  FILLER                  PIC X(80) VALUE
000500     "000000     03  channel.                                    ".
000510     03  FILLER                  PIC X(80) VALUE
000520     "000000         05  channel-number      PIC 99.             ".
000530     03  FILLER                  PIC X(80) VALUE
000540     "000000*            <number=>                               ".
000550     03  FILLER                  PIC X(80) VALUE
000560     "000000         05  channel-name        PIC X(20).          ".
000570     03  FILLER                  PIC X(80) VALUE
000580     "000000*            <>                                      ".
000590     03  FILLER                  PIC X(80) VALUE
000600     "000000         05  TV-program          OCCURS 10.          ".
000610     03  FILLER                  PIC X(80) VALUE
000620     "000000             07  prog-name       PIC BBX(20)BB.      ".
000630     03  FILLER                  PIC X(80) VALUE
000640     "000000             07  prog-time       PIC X(05).          ".
000650     03  FILLER                  PIC X(80) VALUE
000660     "000000             07  prog-rating     OCCURS 1.           ".
000670     03  FILLER                  PIC X(80) VALUE
000680     "000000*                <rating>                            ".
000690     03  FILLER                  PIC X(80) VALUE
```

```
000700     "000000                 09  rating-txt1 PIC X(20)B.        ".
000710     03  FILLER                  PIC X(80) VALUE
000720     "000000*                    <>                             ".
000730     03  FILLER                  PIC X(80) VALUE
000740     "000000                 09  mark        PIC Z9.            ".
000750     03  FILLER                  PIC X(80) VALUE
000760     "000000                 09  rating-txt2 PIC BX(20).        ".
000770     03  FILLER                  PIC X(80) VALUE
000780     "000000*                    <>                             ".
000790     03  FILLER                  PIC X(80) VALUE
000800     "000000         05  channel-owner       PIC X(20).         ".
000810     03  FILLER                  PIC X(80) VALUE
000820     "000000*            <>                                     ".
000830***  End of the CRD SOURCE AREA
000840
000850***  Storage area for the largest possible XML document:
000860 01  XML-DOCUMENT                PIC X(16000)     VALUE SPACE.
000870
000880 01  OTHER-PARAMETER-FIELDS.
000890     03  CRD-RECORD-COUNT        PIC S9(8) BINARY VALUE ZERO.
000900     03  COBOL-RECORD-LENGTH     PIC S9(8) BINARY VALUE ZERO.
000910     03  XML-DOCUMENT-LENGTH     PIC S9(8) BINARY VALUE ZERO.
000920     03  FEEDBACK-CODE           PIC S9(4) BINARY VALUE ZERO.
000930     03  FEEDBACK-TEXT           PIC X(80)        VALUE SPACE.
000940
000950 PROCEDURE DIVISION.
000960
000970 TOP-LEVEL SECTION.
000980*****************************************************************
000990*  This section will populate the COBOL record and call RCIMSXML *
001000*  to generate an XML document containing TV program listings.   *
001010*****************************************************************
001020 TOP-ENTER.
001030
001040     COMPUTE CRD-RECORD-COUNT = LENGTH OF TV-LISTINGS-SOURCE / 80.
001050     MOVE LENGTH OF TV-LISTINGS  TO COBOL-RECORD-LENGTH.
001060     MOVE LENGTH OF XML-DOCUMENT TO XML-DOCUMENT-LENGTH.
001070
001130     INITIALIZE                     TV-LISTINGS.
001140
001150     MOVE 3                      TO CHANNEL-NUMBER.
001160     MOVE "Popular TV"           TO CHANNEL-NAME.
001170     MOVE "Mr Nice Guy "         TO CHANNEL-OWNER.
001180
001190     MOVE "Sunday Night Live"    TO PROG-NAME (1).
001200     MOVE "10:00"                TO PROG-TIME (1).
001210     MOVE "I gave it   "         TO RATING-TXT1 (1 1).
001220     MOVE 8                      TO MARK (1 1).
001230     MOVE "out of ten  "         TO RATING-TXT2 (1 1).
001240
001250     MOVE "News"                 TO PROG-NAME (2).
001260     MOVE "11:30"                TO PROG-TIME (2).
```

```
001270
001280     MOVE "Weather"              TO PROG-NAME (3).
001290     MOVE "11:55"                TO PROG-TIME (3).
001300
001310***  The following call will generate XML for three programs on
001320***  channel 3 using the TV-program array.
001330     PERFORM A-CALL-RCIMSXML.
001340
001342
001350     INITIALIZE                     CHANNEL.
001360
001370     MOVE 5                      TO CHANNEL-NUMBER.
001380     MOVE "Soaps & Soaps"        TO CHANNEL-NAME.
001390     MOVE "Steady Eddie"         TO CHANNEL-OWNER.
001400
001410     MOVE "Westenders"           TO PROG-NAME (1).
001420     MOVE "09:00"                TO PROG-TIME (1).
001430
001440***  The following (repeated) call will generate XML for all
001450***  changed and populated elements. ie: channel 5.
001460     PERFORM A-CALL-RCIMSXML.
001470
001480     MOVE HIGH-VALUES            TO TV-LISTINGS.
001490
001500***  This final call completes the document.
001510     PERFORM A-CALL-RCIMSXML.
001520
001530***  Process XML-DOCUMENT (1:XML-DOCUMENT-LENGTH)
001590
001600     STOP RUN.
001610
001620 TOP-EXIT.
001630     EXIT.
001640
001650
001660 A-CALL-RCIMSXML SECTION.
001670*****************************************************************
001680*  This section executes the CALL to the interface and checks    *
001690*  the feedback code.                                            *
001700*****************************************************************
001710 A-ENTER.
001720
001730     CALL "RCIMSXML"          USING TV-LISTINGS-SOURCE
001740                                    CRD-RECORD-COUNT
001750                                    TV-LISTINGS
001760                                    COBOL-RECORD-LENGTH
001770                                    XML-DOCUMENT
001780                                    XML-DOCUMENT-LENGTH
001790                                    FEEDBACK-CODE
001800                                    FEEDBACK-TEXT.
001810
001820     IF  FEEDBACK-CODE > ZERO
```

```
001830         DISPLAY "BAD RETURN FROM RCIMSXML - FEEDBACK CODE IS "
001840                                       FEEDBACK-CODE
001850         DISPLAY " MESSAGE READS: " FEEDBACK-TEXT
001860         STOP RUN
001870     END-IF.
001880
001890 A-EXIT.
001900     EXIT.
001910
```

Generating Multiple XML Documents

The Standalone level of the Redvers COBOL XML Interface is a particularly adaptable interface level. It can be used in batch or on-line environments, it can generate a single XML document in an instant or it can be part of a continually running task, generating many XML documents throughout its execution, as in the case of an Enterprise Service Bus in a Service Oriented Architecture. If multiple documents are to be generated in a single execution, the following processing options should be observed after the first XML document has been generated:

Multiple Documents using the Same CRD

If multiple XML documents are to be generated using the same CRD as the one used previously, **CRD-RECORD-COUNT** can be set to zero so that **CRD-SOURCE-AREA** isn't processed again unnecessarily. All other calling parameters must be reset to the correct values for the new document (remember to reset **XML-DOCUMENT-LENGTH** to the full size of **XML-DOCUMENT**). Processing can then restart from **The First Call**.

Multiple Documents using a Different CRD

If multiple XML documents are to be generated using a different CRD from the one used previously, all calling parameters must be reset to the correct values for the new document (remember to reset **XML-DOCUMENT-LENGTH** to the full size of **XML-DOCUMENT**). Processing can then restart from **The First Call**.

Optimizing a Continuous Execution

If an application uses a continual or long running task, that generates multiple XML documents, efficiency will be improved by setting **CRD-RECORD-COUNT** to zero as often as possible (see Multiple Documents using the Same CRD above).

In circumstances where the long running task must generate a mix of different types of XML document, the most common document types can be identified and processed using a copy of RCIMSXML dedicated to that document type, therefore always using a **CRD-RECORD-COUNT** of zero (except the first document). Copies of RCIMSXML can be created by duplicating the RCIMSXML source code and replacing "RCIMSXML" with a different program id - see Program-ID in User Maintained Variables section for details.

Structure Break

Structure Break processing was introduced with version 2.4 of the interface. It is used when an XML document is required to hold more than one hierarchy of information, giving rise to many-to-many relationships within the document. Alternatively, a structure break can be used when totals are to be reported at the end of the document which are not known when the initial population of the CRD is performed.

Normal Operation

The interface generator module creates XML elements from the series of data images passed to it in the COBOL Record Definition (CRD). For each call, the prior image of any changed fields are translated into XML elements along with any start/end tags necessary to ensure the generation of a well formed document. In addition to this, elements are generated in order to preserve the field relationships that exist for each image of the CRD passed to the generator subroutine.

Eg:

The first call:

Fields in CRD	Content
03 TV-program. 05 prog-name PIC X(20). 05 prog-cost PIC $$$,$$9.99.	"Sunday Night Live " 2500000

The second call:

Fields in CRD	Content
03 TV-program. 05 prog-name PIC X(20). 05 prog-cost PIC $$$,$$9.99.	"Sunday Night Live " 1000000

The third call:

Fields in CRD	Content
03 TV-program. 05 prog-name PIC X(20). 05 prog-cost PIC $$$,$$9.99.	"News " 1000000

Generates:

XML Document
<TV-program> <prog-name>Sunday Night Live</prog-name> <prog-cost>$25,000.00</prog-cost> <prog-cost>$10,000.00</prog-cost> </TV-program> <TV-program> <prog-name>News</prog-name> <prog-cost>$10,000.00</prog-cost> </TV-program>

In the example above, "Sunday Night Live" has two costs associated with it and therefore two **<prog-cost>** elements are generated. This reflects the one-to-many relationship between **<prog-name>** and **<prog-cost>.** However, when "News" is passed to the interface with a `prog-cost` equal to the previous `prog-cost` (by pure coincidence) the interface does not just generate the **<prog-name>** element, instead it rebuilds the XML from the parent of **<prog-name>** and **<prog-cost>.** The interface does this because it records the fact that while "Sunday Night Live" existed in the CRD for both $25,000.00 and $10,000.00, "News" only existed for a cost of $10,000.00. This is the desired outcome 99% of the time.

What is a Structure Break?

A Structure Break causes the generator subroutine to complete the creation of XML for data passed in the prior call and then wipe clean its record of what values previously existed on the CRD. Generation is then restarted by the next normal call.

How to code a Structure Break

A Structure Break is triggered by moving `LOW-VALUES` (binary zeroes) to the entire CRD area and calling the generator subroutine. On returning from the call, the generator will repopulate the CRD with all the data values that existed prior to the Structure Break call.

When to use a Structure Break

A Structure Break can be used to create "many-to-many" relationships between elements in an XML document. Usually, if a single data file is being used to create XML, only "one-to-many" relationships exist between the fields. For example, on one day there would be many channels, and one channel would have many programs, and one program would have many costs, etc. However, if the XML document is the result of more than one source file "many-to-many" relationships may exist.

The example below shows how a Structure Break can be used to list television program information as well as all staff associated with a day's production, resulting in a "many to many" relationship between **<TV-program>** and **<staff-name>.**

Eg:

The first call:

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	"Sunday Night Live "
07 prog-cost PIC $$$,$$9.99.	2500000
05 channel-staff.	
07 staff-name PIC X(20).	spaces

The second call:

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	"Sunday Night Live "
07 prog-cost PIC $$$,$$9.99.	1000000
05 channel-staff.	
07 staff-name PIC X(20).	spaces

The third call:

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	"News "
07 prog-cost PIC $$$,$$9.99.	1000000
05 channel-staff.	
07 staff-name PIC X(20).	spaces

The fourth call (Structure Break):

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	low values
07 prog-cost PIC $$$,$$9.99.	low values
05 channel-staff.	
07 staff-name PIC X(20).	low values

The fifth call:

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	"News "
07 prog-cost PIC $$$,$$9.99.	1000000
05 channel-staff.	
07 staff-name PIC X(20).	John Smith

The sixth call:

Fields in CRD	Content
03 channel.	
05 TV-program.	
07 prog-name PIC X(20).	"News "
07 prog-cost PIC $$$,$$9.99.	1000000
05 channel-staff.	
07 staff-name PIC X(20).	Jane Jones

Generates:

XML Document

```
<channel>
 <TV-program>
  <prog-name>Sunday Night Live</prog-name>
  <prog-cost>$25,000.00</prog-cost>
  <prog-cost>$10,000.00</prog-cost>
 </TV-program>
 <TV-program>
  <prog-name>News</prog-name>
  <prog-cost>$10,000.00</prog-cost>
 </TV-program>
 <channel-staff>
  <staff-name>John Smith</staff-name>
  <staff-name>Jane Jones</staff-name>
 </channel-staff>
</channel>
```

In the previous example, the content of **staff-name** was spaces for all calls prior to the Structure Break, yet no empty element (**<staff-name/>**) was generated. This is because all record of the spaces in **staff-name** was erased by the Structure Break call. This leads to the other application for Structure Breaks: trailer totals.

If an XML element is required at the end of the document, containing a control total of a numeric field, or even a hash total of several fields, this can be generated after a Structure Break, thereby delaying the need to populate the field in the CRD until the end of generation – when the value of the field is known.

The example below shows how a Structure Break can be used to generate the **<total-cost>** element for a day's programs only when the value of the total is known to the application.

Eg:

The first call:

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	"Sunday Night Live "
07 prog-cost	PIC $$$,$$9.99.	2500000
05 total-cost	PIC $$$,$$9.99.	zero

The second call:

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	"Sunday Night Live "
07 prog-cost	PIC $$$,$$9.99.	1000000
05 total-cost	PIC $$$,$$9.99.	zero

The third call:

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	"News "
07 prog-cost	PIC $$$,$$9.99.	1000000
05 total-cost	PIC $$$,$$9.99.	zero

The fourth call (Structure Break):

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	low values
07 prog-cost	PIC $$$,$$9.99.	low values
05 total-cost	PIC $$$,$$9.99.	Low values

The fifth call:

Fields in CRD		Content
03 channel.		
05 TV-program.		
07 prog-name	PIC X(20).	"News ”
07 prog-cost	PIC $$$,$$9.99.	1000000
05 total-cost	PIC $$$,$$9.99.	4500000

Generates:

XML Document

```
<channel>
 <TV-program>
  <prog-name>Sunday Night Live</prog-name>
  <prog-cost>$25,000.00</prog-cost>
  <prog-cost>$10,000.00</prog-cost>
 </TV-program>
 <TV-program>
  <prog-name>News</prog-name>
  <prog-cost>$10,000.00</prog-cost>
 </TV-program>
 <total-cost>$45,000.00</total-cost>
</channel>
```

Data Integrity

Character Range

RCIMSXML accepts single byte characters in the hexadecimal range "00" through "FF". However, the use of hex "00" (null/low-values) has special meaning to the interface and these characters are converted to spaces before being passed to the XML document (see Controlling the Point of Change section). The low-values character is not within the XML character range defined by the W3C Extensible Markup Language (XML) 1.0 (Second Edition) definition. It is the application's responsibility to ensure that only characters within the permitted range of the XML protocol are generated.

Character References

Unicode character references (eg: î = î) may appear in entity declarations or as part of the data passed in the CRD. No attempt is made to interpret their character form.

Entity References

Data is usually transferred from CRD fields to XML elements without alteration. However, because certain characters are interpreted as instructions by XML parsers, these are automatically translated to their predefined entity references. The characters affected and their translations are listed below:

Character	Description	Entity Reference
>	greater than	&gt;
<	less than	&lt;
&	ampersand	&amp;
'	apostrophe	&apos;
"	double quote	&quot;

RCIMSXML does not attempt to convert text in CRD fields to application specific entity references. If application specific entity references are required, they must be declared as general, internal, parsed entities in the DTD header table (see DTD-headers in the User Maintained Variables section) and passed to the interface in entity form (&xyz;). Under these circumstances the "&" character is passed unchanged to the XML document.

Empty Fields

An elementary alpha or alphanumeric field in the CRD is deemed empty if it contains only spaces and/or low values. An elementary numeric field in the CRD is deemed empty if it contains only zeroes and/or numeric edit characters (“£”, “$”, “/”, “*”, “.”, “-“, “+”, “CR”, “DB” or ”,”). A group level CRD field is deemed empty if all of its subordinate fields are deemed empty and it contains no mandatory (required) attributes and it contains no mandatory (minOccurs>”0”) child elements.

When an empty field has an override tag name ending with an “=”, the attribute is not generated unless it is defined in the CRD as *required* **(<<attrName=>>)** – see Required Attributes and Elements for more details. When an empty field doesn’t have an override tag name ending with an “**=**”, an element is generated in the form of an empty tag **(<tagName/>)** unless it’s a trailing occurrence in an array, when it is not generated.

CDATA

CDATA sections can be generated in one of two ways. The first option is to use the COBOL STRING command to string the CDATA literals directly into the CRD field around a working storage variable:

Eg:
```
STRING "<![CDATA[" WS-FIELD "]]>" DELIMITED BY SIZE INTO CRD-FIELD.
```

Alternatively, the element can be defined to the CRD in a similar way to a mixed content element and the CDATA literals moved into the CRD or predefined using the **VALUE** clause:

Eg:

Fields in CRD	Content
03 TV-program. 05 FILLER PIC X(9) VALUE "<![CDATA[". 05 prog-name PIC X(16). * <> 05 FILLER PIC X(3) VALUE "]]>".	 "<![CDATA[". “News & Weather ” "]]>"

Generates:

XML Document
<TV-program><![CDATA[News & Weather]]></TV-program>

Maximum Document Size

The Redvers COBOL XML Interface is designed to process XML documents up to 99,999,999 bytes in length. As this limit exceeds the maximum field size for most COBOL compilers, the picture clause for the the **XML-DOCUMENT** parameter in linkage is set to: `PIC X(9999999)`.

If a document length greater than 9,999,999 bytes is required, and if the platform can support a greater field length, the picture clause for the **XML-DOCUMENT** parameter in linkage may need to be changed from: `PIC X(9999999)` to a longer picture definition (up to 99,999,999 bytes).

Processing Instructions

It is not currently possible to generate processing instructions with this interface.

Comments

Comments can be coded into the start of the XML document using the **XML-headers** User Maintained Variable or the **DTD-headers** User Maintained Variable – See User Maintained Variables for details.

User Maintained Variables

One of the features of RCIMSXML is that it's delivered as COBOL source code. This means certain parameters can be adjusted to suit the requirements of individual applications. These parameters are called User Maintained Variables and can be found within the first 200 lines of the interface subroutine source code, marked by a following comment line beginning **<UMV>** with "*"s underlining the variable value.

NO PROCEDURE DIVISION CHANGES ARE EVER NECESSARY.

These variables are defaulted to values that should be adequate in most circumstances while keeping storage requirements to a minimum.

Note: *Changes to User Maintained Variables in accordance with these instructions will not invalidate the warranty.*

Program-ID

The **PROGRAM-ID** may be changed to suit site standards or to allow multiple versions of RCIMSXML with different User Maintained Variables.

Maximum-COBOL-record-length

The length of this field dictates the maximum **COBOL-RECORD-LENGTH** that can be passed to RCIMSXML in one call. This length must be increased if more than 4096 characters of data are to be passed or it may be decreased if storage is limited.

XML-declaration and XML-headers

These text values provide the facility to control XML lines written at the start of the XML document (including the declaration). For example they can be used to include external DTD's or schemas in the document. Any single quote marks (apostrophes) within the texts are converted to double quote marks before being written. If populated, each text string generates a line in the XML document.

Additional header lines may be added by coding further **FILLER** fields of 100 characters containing text values, if the **Number-of-XML-headers** UMV is correspondingly increased.

Number-of-XML-headers

This value must be set to the number of **XML-declaration** and **XML-header** fields, described above.

DTD-headers

These text values provide the facility to control XML lines written at the start of the DTD. For example they can be used to declare entity references. Any single quote marks (apostrophes) within the texts are converted to double quote marks before being written. If populated, each text string generates a line in the XML document.

Additional DTD header lines may be added by coding further **FILLER** fields of 100 characters containing text values, if the **Number-of-DTD-headers** UMV is correspondingly increased.

Number-of-DTD-headers

This value must be set to the number of **DTD-header** fields, described above.

End-of-line-chtrs

In order to produce a more readable XML document some applications may wish to include carriage return, line feed or new line characters at the end of each line in the XML document. The characters in this field provide this facility when they are populated with hexadecimal values other than space. A single occurrence of any non-space character in this area will be appended to every logical line in the XML document.

Maximum-number-of-fields

The maximum number of discrete fields in the COBOL Record Definition (CRD) is defaulted to 400 (fields using the **OCCURS** clause are counted as one discrete field). If an application requires more than 400 fields in a single CRD, the number in the **OCCURS** clause for this UMV can be increased. Similarly, if storage is limited, this value can be decreased to save on storage requirements.

Maximum-tag-length

The maximum allowable XML tag length is defaulted to 100 characters. If an application requires tag lengths of more than 100 characters, the picture clause for this UMV can be increased (maximum 400 characters). Similarly, if storage is limited, this value can be decreased to the largest expected tag length (minimum 30 characters).

Feedback Messages

A zero **FEEDBACK-CODE** indicates processing has completed successfully and that **FEEDBACK-TEXT** contains diagnostic information for the first, subsequent or final call.

The +100 series indicate fatal processing errors at file or parameter level.

The +200 series indicate fatal errors in the decoding of the COBOL Record Definition.

The +300 series indicate fatal errors encountered after the first call.

FEEDBACK-CODE	FEEDBACK-TEXT
+0000	COMPLETED FIRST CALL. CRD FIELDS PARSED:9999 RECORD LENGTH:99999 or COMPLETED CALL NO.: 99999999 XML LINES GENERATED SO FAR:99999999 or COMPLETED LAST CALL:99999999 TOTAL XML LINES GENERATED:99999999

FEEDBACK-CODE	FEEDBACK-TEXT	Reason
+0101	NO FIELDS IDENTIFIED ON RECORD DEFINITION	No fields could be identified in **CRD-SOURCE-AREA**.
+0102	TOO MANY FIELDS ON RECORD DEFINITION	The number of CRD fields exceeded the number allowed in the interface table. (See Maximum-number-of-fields in the User Maintained Variables section.)
+0103	PROCESSING EXCEPTION. PLEASE CONTACT REDVERS CONSULTING	There has been an internal logic error within the program. Please contact your Redvers Consulting account manager.
+0104	LENGTH OF RECORD DEFINITION IS > MAX COBOL RECORD	After storing all the field details from the CRD, the total length of **COBOL-RECORD** was calculated. This total exceeded the size of the maximum **COBOL-RECORD** permitted. (See Maximum-COBOL-record-length in the User Maintained Variables section.)
+0110	RECORD DEFINITION / LINKAGE MISMATCH	The **COBOL-RECORD-LENGTH** passed in linkage was not the same as the length calculated for the CRD. Probable causes are that the CRD has been changed but the calling program was not recompiled or that the layout used in the calling program is not the one in **CRD-SOURCE-AREA**.

+0151	THE UMV MAXIMUM XML RECORD LENGTH IS TOO SMALL	Two element tag names, plus level indentation, plus 28, is greater than the 1024 characters allowed. The maximum tag length must be reduced. (See Maximum-tag-length in the User Maintained Variables section.)
+0152	THE UMV TAG LENGTH IS INVALID	The User Maintained Variable for the maximum tag length must be from 30 to 400 characters. (See Maximum-tag-length in the User Maintained Variables section.)
+0180	INVALID ACTIVATION KEY. PLEASE PLACE YOUR ACTIVATION KEY IN THE LAST W.S. FIELD	The Redvers COBOL XML Interface is supplied with a 32 character activation key. Please edit the RCIMSXML source code and place this activation key in the **VALUE** clause literal for the last field definition in working storage. Then recompile.
+0190	30 DAY TRIAL PERIOD EXPIRED OR CALL LIMIT REACHED	The Redvers COBOL XML Interface can be downloaded free of charge for a thirty day trial period. This free version may be called up to 100 times in a single application execution. Either the thirty days have now elapsed or your application is trying to call RCIMSXML more than 100 times. Please contact Redvers Consulting for an additional thirty day trial or to arrange payment.
+0201	TOO MANY CHARACTERS FOR LEVEL NUMBER	The interface was expecting a COBOL level number in the CRD but found a string of more than two characters. *This message also points to the offending line number within the CRD.*
+0202	INVALID LEVEL NUMBER	A COBOL level number that was not numeric or a level number greater than 49 but not 88 was found in the CRD. *This message also points to the offending line number within the CRD.*
+0203	ILLEGAL DATA CLAUSE FOUND	An unsupported data clause was found in the field definition. See Clauses not Supported section. *This message also points to the offending line number within the CRD.*
+0204	INVALID NUMBER OF OCCURRENCES	An **OCCURS** clause was identified but it was not followed by a valid integer of more than zero and less than 9999. *This message also points to the offending line number within the CRD.*
+0205	INVALID PICTURE DEFINITION	A **PIC** clause was identified but it was not followed by a valid string of characters less than 30 in length. *This message also points to the offending line number within the CRD.*

+0207	EXCEEDED MAXIMUM TAG SIZE	An override tag name of more than the maximum number of characters was encountered. (See Maximum-tag-length in the User Maintained Variables section.) *This message also points to the offending line number within the CRD.*
+0210	TAG DOES NOT START WITH AN ALPHABETIC CHARACTER	All XML tags must start with an alphabetic character. *This message also points to the offending line number within the CRD.*
+0211	TAG NAME CONTAINS INVALID XML CHARACTERS	XML tag names are confined to using only alphabetic, numeric, "-", "_", ":" or "." characters. *This message also points to the offending line number within the CRD.*
+0214	ATTRIBUTES MAY NOT BE GROUP LEVEL ITEMS	Attributes (tag names ending with "=") must be elementary data items. *This message also points to the offending line number within the CRD.*
+0215	ATTRIBUTE TAGS MAY NOT OCCUR MORE THAN ONCE	Attribute tags (those ending with "=") may occur only once in the parent/group level tag. *This message also points to the offending line number within the CRD.*
+0220	GROUP ITEM MUST NOT HAVE A PICTURE	The field named in the message is at a higher level than the next field in the CRD and is therefore a group item. However, a picture clause was encountered for this field. *This error would normally have been caught in the compile stage.*
+0221	ELEMENTARY FIELD MUST HAVE A PICTURE	The field named is at an equal or lower level than the next field in the CRD and is therefore an elementary item. However, a picture clause was not given for this field. *This error would normally have been caught in the compile stage.*
+0222	MORE THAN ONE ROOT ELEMENT FOUND	The structure of the CRD may not have more than one root level. *This message includes the offending field tag name.*
+0224	NO TAG NAME FOR THE ROOT ELEMENT	A COBOL name or override tag name is required for the root element of any XML document.
+0225	THE ROOT ELEMENT CANNOT OCCUR MORE THAN ONCE	The maximum OCCURS value for a root element is 1.

+0226	**INVALID POSITION FOR ATTRIBUTE**	Attributes (tag names ending with a "=") must be coded within the group they relate to and they must be the first fields in that group. *This message includes the offending attribute name.*
+0227	**MISSING GROUP TAG FOR ATTRIBUTE ELEMENT**	The group level data item must have a tag if it is to hold an attribute (tag names ending with a "=") field. *This message includes the offending attribute name.*
+0230	**ENCOUNTERED MULTIPLE DIMENSION ARRAY**	An **OCCURS** clause greater than 1 has been nested within another **OCCURS** clause greater than 1. Only single dimension arrays are currently supported. To overcome this problem define a single dimension array and make multiple calls to the interface for each occurrence of the data item. *This message includes the offending field tag name.*
+0301	**CALL SEQUENCE ERROR**	A call has been made to the interface after processing has completed. Likely explanations are that **FEEDBACK-CODE** was not checked after a previous unsuccessful call or that the calling program has previously completed the XML document by executing an end-of-run call (high-values in **COBOL-RECORD**).
+0302	**PARAMETERS MUST NOT CHANGE AFTER INITIAL CALL**	One or more of the input calling parameters was found to have changed after the first call to the interface. Likely explanations are that the application has inadvertently overwritten one or more of the working storage fields used in the call to the interface or the application is attempting to generate more than one XML document without first issuing the final high-values call.
+0310	**ATTEMPTED TO GENERATE MULTIPLE ROOT DOCUMENT**	When generating repeating information the program found that the lowest group level common to the repeating information (the Point Of Change) was the root element and therefore cannot be repeated. To avoid this, code the root element as the only 01 level in the CRD followed by a single 02 level so that repeating information can be generated under multiple 02 levels. If you don't want to pass this additional level to XML call it **FILLER**.

+0330	**XML DOCUMENT LENGTH EXCEEDS DOCUMENT AREA**	When building the XML document RCIMSXML has attempted to address beyond the length of the **XML-DOCUMENT** area in the calling program. Ensure that **XML-DOCUMENT-LENGTH** contains the full length of the **XML-DOCUMENT** area in the calling program. Otherwise the size of **XML-DOCUMENT** must be increased in the calling program.

Index

T

U

W

X

Redvers Consulting Ltd

44 Broadway, London E15 1XH, UK
http://www.redversconsulting.com/