



*Redvers Consulting Ltd*

# Redvers COBOL XML Interface

## User Guide

Superfast Generator  
RCFSTXML Version 2.9

---

# Contents

Preface.....	5
Overview.....	6
Installation.....	7
Coding the COBOL Record Definition.....	9
Field Names.....	9
Default Tag Names.....	9
Override Tag Names.....	10
XML Attributes.....	10
XML Namespaces and SOAP.....	11
PICTURE Clause.....	12
Binary / Packed Fields.....	13
Other Clauses.....	14
BLANK WHEN ZERO.....	14
JUSTIFIED RIGHT.....	14
OCCURS.....	14
Clauses Not Supported.....	14
Structure.....	15
Mixed Content Elements.....	16
Non-tagged group level fields.....	17
Advanced Techniques.....	18
Formatting.....	18
Required Attributes and Elements.....	19
Optional Elements.....	20
Excluded Elements.....	21
Multiple XML Elements in One Field.....	22
XML Declaration Override.....	23
Document Type Definition Override.....	24
Repeating Groups.....	25
Using OCCURS.....	25
Using Repeated Calls.....	26
Controlling the Point of Change.....	28
Orphan Repeats.....	30
Calling RCFSTXML.....	32
Parameters.....	32
CRD-OBJECT-AREA (input).....	32

---

CRD-RECORD-COUNT (input) .....	32
COBOL-RECORD (input).....	32
COBOL-RECORD-LENGTH (input) .....	32
XML-DOCUMENT (output) .....	32
XML-DOCUMENT-LENGTH (input & output).....	33
FEEDBACK-CODE (output) .....	33
FEEDBACK-TEXT (output) .....	33
Calls to RCFSTXML.....	34
The First Call .....	34
Subsequent Calls .....	34
The Last Call .....	34
Sample Program Calling RCFSTXML.....	35
Structure Break.....	39
Normal Operation .....	39
What is a Structure Break? .....	40
How to code a Structure Break .....	40
When to use a Structure Break.....	41
Data Integrity .....	45
Character Range.....	45
Character References .....	45
Entity References.....	45
Empty Fields .....	46
CDATA.....	46
Maximum Document Size.....	47
Processing Instructions.....	47
Comments .....	47
User Maintained Variables.....	48
Program-Id .....	48
SELECT Statements .....	48
File Definition Statements .....	48
Maximum-COBOL-record-length .....	49
XML-declaration and XML-headers .....	49
Number-of-XML-headers.....	49
DTD-headers.....	49
Number-of-DTD-headers .....	50
End-of-line-chtrs.....	50
Maximum-number-of-fields .....	50
RCFSTCMP Compile Errors .....	51

---

---

RCFSTXML Feedback Messages.....	54
Index.....	56

## Preface

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

Program RCFSTCMP is a batch compile process that reads a COBOL Record Definition (CRD) in order to produce a CRD object file for use in Redvers COBOL XML Interface programs RCFSTXML and RCFSTCOB.

Program RCFSTXML is a COBOL subroutine that generates XML documents from COBOL format data passed from customer applications. Output from RCFSTCMP provides RCFSTXML with all the information it needs to produce application specific XML. The generated well-formed XML standalone documents conform to the World Wide Web Consortium (W3C) Extensible Markup Language (XML) 1.0 (Second Edition) definition.

RCFSTXML is the counterpart to RCFSTCOB 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:

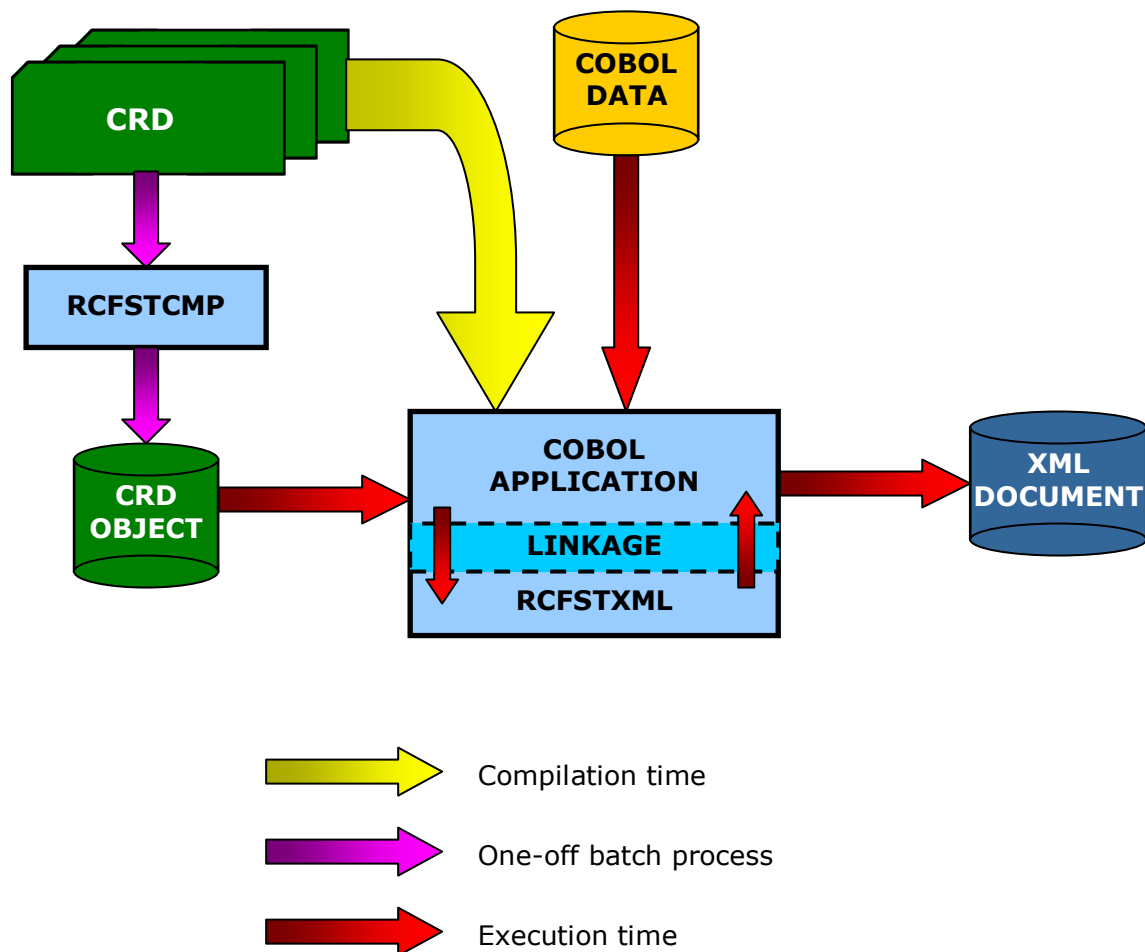
[http://www.redversconsulting.com/downloads/user\\_guides/RCFSTXML\\_2.9\\_user.pdf](http://www.redversconsulting.com/downloads/user_guides/RCFSTXML_2.9_user.pdf)



## Overview

RCFSTXML is a COBOL subroutine that is compiled and linked into application programs in order to generate well-formed XML documents. This particular interface level was built for large scale applications requiring fast and efficient XML generation. In order to achieve maximum speed and efficiency, the interpretation of the COBOL Record Definition (CRD) has been passed to an off-line batch compile process (RCFSTCMP).

Central to the operation of the interface is the COBOL Record Definition (CRD). This definition (usually a COBOL copybook member) is included in the application program source code where it describes the working storage fields that are to be passed to RCFSTXML. In addition, the CRD source is read by RCFSTCMP in a one-off batch process which creates a CRD object file containing the raw field positions, lengths, data types etc. At execution time, application programs load this CRD object file into working storage and then pass it to RCFSTXML in a CALL statement.



# Installation

RCFSTXML and RCFSTCMP are self-contained source programs that require 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 RCFSTXML and RCFSTCMP, 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 programs can then be compiled using your standard COBOL compiler.

RCFSTCMP is the batch CRD compiler program, required to pre-process CRD's used by the Redvers COBOL XML Interface (Superfast level). To run RCFSTCMP, a batch job will be required with the following file attributes:

- **RCCRDIN** – (input) fixed length 80 byte sequential file containing COBOL source code records of working storage field definitions (the CRD).
- **RCCRDOBJ** – (output) fixed length 132 byte sequential file with one record equating to each field on the input source code.
- File handling for output from **DISPLAY** statements, so that record counts or compilation errors can be viewed.

RCFSTXML is the XML generator subroutine component of the Redvers COBOL XML Interface (Superfast level). To execute RCFSTXML, add a **CALL** statement to your COBOL application, passing the appropriate CRD object file created by RCFSTCMP and the COBOL data. See [Calling RCFSTXML](#) for CALL parameter details.

*When running a free 30 day trial, the sample calling program RCFXCALL can also be copied to your source code library, compiled and linked.*

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 (RCFSTXML). 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 reserved words, character sets and the maximum length of a field name, the default tag may be overridden by coding an alternate XML tag 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=> 05 prog-name PIC X(20) .	"10:00" "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). * <uk:time>	"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
<pre>&lt; SOAP-ENV:Envelope xmlns:SOAP-ENV=" http://schemas.xmlsoap.org/soap" xmlns:uk="http://www.greenwichmeantime.co.uk" xmlns:us="http://www.easternstandardtime.com"&gt; &lt;SOAP-ENV:Body&gt;   &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;   &lt;uk:time&gt;10:00&lt;/uk:time&gt;   &lt;us:time&gt;05:00&lt;/us:time&gt; &lt;/SOAP-ENV:Body&gt; &lt;/SOAP-ENV:Envelope&gt;</pre>

## 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.	31122013
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
<pre>&lt;TV-program&gt;   &lt;prog-date&gt;31/12/2013&lt;/prog-date&gt;   &lt;prog-name&gt; Sunday Night Live &lt;/prog-name&gt;   &lt;prog-time&gt;10:00&lt;/prog-time&gt;   &lt;prog-duration&gt;.333&lt;/prog-duration&gt;   &lt;prog-rating&gt;0E&lt;/prog-rating&gt;   &lt;prog-cost&gt;\$2,500,000.00DB&lt;/prog-cost&gt; &lt;/TV-program&gt;</pre>

**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
<pre>&lt;prog-rating&gt;Al8=&lt;/prog-rating&gt; &lt;prog-revenue&gt;B1vNFQ==&lt;/prog-revenue &gt;</pre>

**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.	11122013
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
<pre>&lt;TV-listings&gt; &lt;broadcast-date&gt;11/12/2013&lt;/broadcast-date&gt; &lt;channel&gt; &lt;channel-number&gt;05&lt;/channel-number&gt; &lt;TV-program&gt; &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt; &lt;prog-time&gt;10:00&lt;/prog-time&gt; &lt;prog-rating&gt;8&lt;/prog-rating&gt; &lt;/TV-program&gt; &lt;/channel&gt; &lt;/TV-listings&gt;</pre>

## 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).	"Sunday Night Live "
05 prog-time PIC X(5).	"10:00"
05 END-TEXT PIC X(10).	" 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).	"Sunday Night Live "
05 prog-time PIC X(5).	"10:00"

**Generates:**

XML Document
<pre>&lt;TV-listings&gt;   &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;   &lt;prog-time&gt;10:00&lt;/prog-time&gt; &lt;/TV-listings&gt;</pre>

## 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.	11122013
03 channel.	
05 channel-number PIC 99.	5
* <number=>	
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
<pre> &lt;TV-listings&gt; &lt;broadcast-date&gt;11/12/2013&lt;/broadcast-date&gt; &lt;channel number="05"&gt; &lt;TV-program&gt; Sunday Night Live &lt;prog-time&gt;10:00&lt;/prog-time&gt; &lt;prog-rating&gt;I gave it &lt;mark&gt;8&lt;/mark&gt; out of ten &lt;/prog-rating&gt; &lt;/TV-program&gt; &lt;/channel&gt; &lt;/TV-listings&gt; </pre>

## 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.	00
* <<number=>>	
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.	00
* <<mark>>	

**Generates:**

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

## 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
<pre>&lt;channel&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;   &lt;/TV-program&gt; &lt;/channel&gt;</pre>

**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 RCFSTXML. For more information, please contact your account manager.

---

## 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.

## Multiple XML Elements in One Field

Under normal circumstances RCFSTXML generates one XML element for every populated/changed field in the CRD. Release 2.9 introduced the capability to write entire XML structures, if these structures have previously been placed in a CRD field.

To enable a CRD field to be used in this way (avoiding any conversion of "<" characters to "&LT;" etc.), an override tag name must be coded for the CRD field, followed by "/C" (upper case or lower case). i.e.: <override\_name/C>.

**Eg:**

Fields in CRD	Populated with
01 TV-listings.	
03 broadcast-date PIC 99/99/9999.	10012013
03 TV-PROGRAM PIC X(100). * <TV-program/C>	"<prog-name>Sunday Night Live</prog-name><prog-time>10:00</prog-time><prog-rating>8</prog-rating>"

**Generates:**

XML Document
<pre>&lt;TV-listings&gt;   &lt;broadcast-date&gt;10/01/2013&lt;/broadcast-date&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;     &lt;prog-time&gt;10:00&lt;/prog-time&gt;     &lt;prog-rating&gt;8&lt;/prog-rating&gt;   &lt;/TV-program&gt; &lt;/TV-listings&gt;</pre>

**Note:** If the COBOL field name doesn't need to be overridden, the override tag name can be omitted. Eg: </c>.

## 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
<pre>*** Start of COBOL Record Definition (CRD) * An override XML declaration follows: *&lt;?xml version='1.0' encoding='ISO-8859-1'?&gt;   01 TV-listings.     03 channel. ... </pre>

### Generates:

XML Document
<pre>&lt;?xml version="1.0" encoding "ISO-8859-1"?&gt; &lt;!-- This XML document was generated by RCFSTXML --&gt; &lt;TV-listings&gt;   &lt;channel&gt; ... </pre>

## Document Type Definition Override

A default external (SYSTEM) Document Type Definition (DTD) can be specified in the [XML-Header](#) UMV (see [User Maintained Variables](#) for details) or an internal DTD can be added to the XML document by coding a "DTD CALL" in the calling application (see [Calls to RCFSTXML](#) for details). However, the hard-coding of external DTD information in working storage can make application specific DTD maintenance difficult.

From Release 2.9, an application specific, external DTD can be coded in comment lines at the start of the CRD, starting with the characters "<!DOCTYPE" (similar to the way XML declaration overrides are specified).

**Eg:**

CRD
<pre>000010***  Start of COBOL Record Definition (CRD) 000020*  An override DTD follows: 000030* &lt;!DOCTYPE TV-listings SYSTEM 'http://devtest.devsys.version2 000040* .dtd'&gt; 000050 01  TV-listings. 000060      03  channel. ...</pre>

**Generates:**

XML Document
<pre>&lt;?xml version="1.0" encoding "UTF-8"?&gt; &lt;!-- This XML document was generated by RCFSTXML --&gt; &lt;!DOCTYPE TV-listings SYSTEM 'http://devtest.devsys.version2.dtd'&gt; &lt;TV-listings&gt;   &lt;channel&gt; ...</pre>



## 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 RCFSTXML 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 RCFSTXML 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	Occurrence	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
<pre> &lt;channel&gt;   &lt;channel-number&gt;05&lt;/channel-number&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;     &lt;prog-time&gt;10:00&lt;/prog-time&gt;     &lt;prog-rating&gt;8&lt;/prog-rating&gt;   &lt;/TV-program&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;News&lt;/prog-name&gt;     &lt;prog-time&gt;11:30&lt;/prog-time&gt;   &lt;/TV-program&gt;   &lt;TV-program/&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Weather&lt;/prog-name&gt;     &lt;prog-time&gt;11:55&lt;/prog-time&gt;   &lt;/TV-program&gt; &lt;/channel&gt; </pre>

**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
<pre> &lt;channel&gt;   &lt;channel-number&gt;05&lt;/channel-number&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;     &lt;prog-time&gt;10:00&lt;/prog-time&gt;   &lt;/TV-program&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;News&lt;/prog-name&gt;     &lt;prog-time&gt;11:30&lt;/prog-time&gt;   &lt;/TV-program&gt; &lt;/channel&gt; &lt;channel&gt;   &lt;channel-number&gt;06&lt;/channel-number&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Westenders&lt;/prog-name&gt;     &lt;prog-time&gt;08:00&lt;/prog-time&gt;   &lt;/TV-program&gt; &lt;/channel&gt; </pre>

## 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 RCFSTXML 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 RCFSTXML 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.	11122013
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.	11122013
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
<pre> &lt;TV-listings&gt;   &lt;TV-today&gt;     &lt;broadcast-date&gt;11/12/2013&lt;/broadcast-date&gt;     &lt;channel&gt;       &lt;channel-number&gt;05&lt;/channel-number&gt;       &lt;TV-program&gt;         &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;         &lt;prog-time&gt;10:00&lt;/prog-time&gt;       &lt;/TV-program&gt;     &lt;/channel&gt;   &lt;/TV-today&gt;   &lt;TV-today&gt;     &lt;broadcast-date&gt;11/12/2013&lt;/broadcast-date&gt;     &lt;channel&gt;       &lt;channel-number&gt;05&lt;/channel-number&gt;       &lt;TV-program&gt;         &lt;prog-name&gt;News&lt;/prog-name&gt;         &lt;prog-time&gt;11:30&lt;/prog-time&gt;       &lt;/TV-program&gt;     &lt;/channel&gt;   &lt;/TV-today&gt; &lt;/TV-listings&gt; </pre>

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
<pre>&lt;channel&gt;   &lt;channel-number&gt;05&lt;/channel-number&gt;   &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;   &lt;prog-time&gt;10:00&lt;/prog-time&gt;   &lt;prog-name&gt;News&lt;/prog-name&gt;   &lt;prog-time/&gt;   &lt;prog-name&gt;Weather&lt;/prog-name&gt;   &lt;prog-time&gt;11:55&lt;/prog-time&gt; &lt;/channel&gt;</pre>

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

# Calling RCFSTXML

## Parameters

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

### **CRD-OBJECT-AREA (input)**

This is the area of storage, loaded by the application, which holds the CRD object output from RCFSTCMP. The CRD defined for the XML document will have been passed through RCFSTCMP in a one-off batch process and the output placed here.

### **CRD-RECORD-COUNT (input)**

This S9(8) binary field contains a count of the number of 132 byte records that make up **CRD-OBJECT-AREA**.

### **COBOL-RECORD (input)**

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

### **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-OBJECT-AREA**.

If the logical length of **COBOL-RECORD** is difficult to determine (perhaps the same **COBOL-RECORD** storage area is being used to process several CRD's of different lengths), this value can be found in the appropriate **CRD-OBJECT-AREA**. The value is defined as an unsigned 8 digit binary field (`PIC 9(8) COMP`) in positions 111 through 114 of **CRD-OBJECT-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 RCFSTXML 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 RCFSTXML 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 RCFSTXML 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:** *The 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 RCFSTXML

### The First Call

If a Document Type Definition (DTD) is required at the start of the XML document, the first call to RCFSTXML 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 RCFSTXML 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 CRD.

### Subsequent Calls

Calls should be made to RCFSTXML (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 RCFSTXML, 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 RCFSTXML remains in memory after the last call, it can be reused to generate another XML document for the same, or a different, CRD by repopulating the calling parameters and restarting the call sequence from [The First Call](#).*

**Note2:** If there is a requirement to generate multiple XML documents simultaneously, this can be done by calling a copy of RCFSTXML, compiled under a different Program-Id using different calling parameters ([Program-id](#) is a [User Maintained Variable](#)).

## Sample Program Calling RCFSTXML

```

000040*****
000050*   This sample program shows how RCFSTXML is called to generate   *
000060*   an XML document in storage from COBOL record input.           *
000070*                                                                 *
000080*   The CRD associated with this application will need to be       *
000090*   passed through the Redvers CRD compiler (RCFSTCMP) before      *
000100*   being loaded by the application. The CRD object file can      *
000110*   exist in storage, on a database or on a flat file. This       *
000120*   sample program uses a flat file (RCCRDOBJ).                   *
000130*****
000132
000140 ENVIRONMENT DIVISION.
000150 INPUT-OUTPUT SECTION.
000160 FILE-CONTROL.
000170
000180     SELECT CRD-FILE                               ASSIGN RCCRDOBJ.
000190
000200 DATA DIVISION.
000210 FILE SECTION.
000220
000230 FD   CRD-FILE
000240     BLOCK CONTAINS 0 CHARACTERS
000250     LABEL RECORDS STANDARD.
000260 01   CRD-RECORD                                PIC X(132).
000270
000280 WORKING-STORAGE SECTION.
000290
000300***   Storage area for the CRD object file:
000310 01   CRD-OBJECT-AREA.
000320     03   CRD-OBJECT-RECORD                        PIC X(132) OCCURS 400.
000330
000340***   Start of COBOL Record Definition (CRD)
000350 01   TV-listings.
000360     03   channel.
000370         05   channel-number                        PIC 99.
000380*         <number=>
000390         05   channel-name                          PIC X(20).
000400*         <>
000410         05   TV-program                            OCCURS 10.
000420         07   prog-name                             PIC BBX(20)BB.
000430         07   prog-time                             PIC X(05).
000440         07   prog-rating                           OCCURS 1.
000450*         <rating>

```

```

000460          09  rating-txt1 PIC X(20)B.
000470*          <>
000480          09  mark          PIC Z9.
000490          09  rating-txt2 PIC BX(20).
000500*          <>
000510          05  channel-owner    PIC X(20).
000520*          <>
000530***  End of COBOL Record Definition (CRD)
000540
000550***  Storage area for the largest possible XML message:
000560 01  XML-DOCUMENT          PIC X(16000)    VALUE SPACE.
000570
000580 01  OTHER-PARAMETER-FIELDS.
000590    03  CRD-RECORD-COUNT      PIC S9(8)  BINARY VALUE ZERO.
000600    03  COBOL-RECORD-LENGTH   PIC S9(8)  BINARY VALUE ZERO.
000610    03  XML-DOCUMENT-LENGTH   PIC S9(8)  BINARY VALUE ZERO.
000620    03  FEEDBACK-CODE         PIC S9(4)  BINARY VALUE ZERO.
000630    03  FEEDBACK-TEXT        PIC X(80)    VALUE SPACE.
000640
000650 01  MISCELLANEOUS-FIELDS.
000660    03  CRD-TABLE-SIZE         PIC 9(4)    VALUE ZERO.
000670    03  CRD-FLAG              PIC X        VALUE SPACE.
000680        88  START-OF-CRD      VALUE SPACE.
000690        88  END-OF-CRD        VALUE "E".
000700
000710 PROCEDURE DIVISION.
000720
000730 TOP-LEVEL SECTION.
000740*****
000750*  This section loads the CRD object file into CRD-OBJECT-AREA  *
000760*  and populates the COBOL record (CRD). It then calls RCFSTXML *
000770*  once for each data group to generate the XML document.      *
000780*****
000790 TOP-ENTER.
000800
000810    COMPUTE CRD-TABLE-SIZE = LENGTH OF CRD-OBJECT-AREA
000820                          / LENGTH OF CRD-OBJECT-RECORD (1).
000830    MOVE LENGTH OF TV-LISTINGS TO COBOL-RECORD-LENGTH.
000840    MOVE LENGTH OF XML-DOCUMENT TO XML-DOCUMENT-LENGTH.
000850
000860    PERFORM A-LOAD-CRD.
000870
000930    INITIALIZE                TV-LISTINGS.
000940
000950    MOVE 3                     TO CHANNEL-NUMBER.
000960    MOVE "Popular TV"          TO CHANNEL-NAME.
000970    MOVE "Mr Nice Guy "       TO CHANNEL-OWNER.
000980
000990    MOVE "Sunday Night Live"   TO PROG-NAME (1).
001000    MOVE "10:00"              TO PROG-TIME (1).
001010    MOVE "I gave it "         TO RATING-TXT1 (1 1).

```

```

001020     MOVE 8                                TO MARK (1 1) .
001030     MOVE "out of ten "                   TO RATING-TXT2 (1 1) .
001040
001050     MOVE "News"                            TO PROG-NAME (2) .
001060     MOVE "11:30"                          TO PROG-TIME (2) .
001070
001080     MOVE "Weather"                         TO PROG-NAME (3) .
001090     MOVE "11:55"                          TO PROG-TIME (3) .
001100
001110***   The following call will generate XML for three programs on
001120***   channel 3 using the TV-program array.
001130     PERFORM C-CALL-RCFSTXML.
001140
001150     INITIALIZE                               CHANNEL.
001160
001170     MOVE 5                                  TO CHANNEL-NUMBER.
001180     MOVE "Soaps & Soaps"                   TO CHANNEL-NAME.
001190     MOVE "Steady Eddie"                   TO CHANNEL-OWNER.
001200
001210     MOVE "Westenders"                     TO PROG-NAME (1) .
001220     MOVE "09:00"                          TO PROG-TIME (1) .
001230
001240***   The following (repeated) call will generate XML for all
001250***   changed and populated elements. ie: channel 5.
001260     PERFORM C-CALL-RCFSTXML.
001270
001280     MOVE HIGH-VALUES                       TO TV-LISTINGS.
001290
001300***   This final call completes the document.
001310     PERFORM C-CALL-RCFSTXML.
001320
001330***   Process XML-DOCUMENT (1:XML-DOCUMENT-LENGTH)
001400
001410     STOP RUN.
001420
001430 TOP-EXIT.
001440     EXIT.
001450
001460
001470 A-LOAD-CRD SECTION.
001480*****
001490*   This section reads the CRD object file into CRD-OBJECT-AREA.  *
001500*****
001510 A-ENTER.
001520
001530     OPEN INPUT CRD-FILE.
001540
001550     PERFORM
001560         UNTIL   END-OF-CRD
001570         OR     CRD-RECORD-COUNT = CRD-TABLE-SIZE
001580         READ CRD-FILE

```

```

001590          AT END
001600          SET END-OF-CRD          TO TRUE
001610          NOT AT END
001620          ADD 1                    TO CRD-RECORD-COUNT
001630          MOVE CRD-RECORD         TO CRD-OBJECT-RECORD
001640                                     (CRD-RECORD-COUNT)
001650          END-READ
001660          END-PERFORM.
001670
001680          CLOSE CRD-FILE.
001690
001700          IF NOT END-OF-CRD
001710              DISPLAY "CRD-OBJECT-AREA TABLE IS FULL!"
001720              DISPLAY "CURRENT SIZE IS: " CRD-TABLE-SIZE
001730              STOP RUN
001740          END-IF.
001750
001760          A-EXIT.
001770          EXIT.
001780
001790
001800          C-CALL-RCFSTXML SECTION.
001810*****
001820*   This section executes the CALL to the interface and checks   *
001830*   the feedback code.                                           *
001840*****
001850          C-ENTER.
001860
001870          CALL "RCFSTXML"           USING CRD-OBJECT-AREA
001880                                     CRD-RECORD-COUNT
001890                                     TV-LISTINGS
001900                                     COBOL-RECORD-LENGTH
001910                                     XML-DOCUMENT
001920                                     XML-DOCUMENT-LENGTH
001930                                     FEEDBACK-CODE
001940                                     FEEDBACK-TEXT.
001950
001960          IF FEEDBACK-CODE > ZERO
001970              DISPLAY "BAD RETURN FROM RCFSTXML - FEEDBACK CODE IS "
001980                                     FEEDBACK-CODE
001990              DISPLAY "MESSAGE READS: " FEEDBACK-TEXT
002000              STOP RUN
002010          END-IF.
002020
002030          C-EXIT.
002040          EXIT.

```

## 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).	"Sunday Night Live "
05 prog-cost PIC \$\$\$,\$\$9.99.	2500000

**The second call:**

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

**The third call:**

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

**Generates:**

XML Document
<pre> &lt;TV-program&gt;   &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;   &lt;prog-cost&gt;\$25,000.00&lt;/prog-cost&gt;   &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt; &lt;/TV-program&gt; &lt;TV-program&gt;   &lt;prog-name&gt;News&lt;/prog-name&gt;   &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt; &lt;/TV-program&gt; </pre>

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
<pre> &lt;channel&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;     &lt;prog-cost&gt;\$25,000.00&lt;/prog-cost&gt;     &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt;   &lt;/TV-program&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;News&lt;/prog-name&gt;     &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt;   &lt;/TV-program&gt;   &lt;channel-staff&gt;     &lt;staff-name&gt;John Smith&lt;/staff-name&gt;     &lt;staff-name&gt;Jane Jones&lt;/staff-name&gt;   &lt;/channel-staff&gt; &lt;/channel&gt; </pre>

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
<pre> &lt;channel&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;Sunday Night Live&lt;/prog-name&gt;     &lt;prog-cost&gt;\$25,000.00&lt;/prog-cost&gt;     &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt;   &lt;/TV-program&gt;   &lt;TV-program&gt;     &lt;prog-name&gt;News&lt;/prog-name&gt;     &lt;prog-cost&gt;\$10,000.00&lt;/prog-cost&gt;   &lt;/TV-program&gt;   &lt;total-cost&gt;\$45,000.00&lt;/total-cost&gt; &lt;/channel&gt; </pre>

## Data Integrity

### Character Range

RCFSTXML 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: &#238; = 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 characters 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;

From Release 2.9, application specific entity references may also be defined in the CRD as they would be coded in a DTD – eg: `<!ENTITY auml 'ä'>`. Customers interested in using this capability should contact their account manager for more details.

## 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[".	"<![CDATA[".
05 prog-name PIC X(16).	"News & Weather "
* <>	
05 FILLER PIC X(3) VALUE "]]>".	"]]>"

**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 **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 RCFSTXML 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 for multiple versions of RCFSTXML and RCFSTCMP with different User Maintained Variables.

## SELECT Statements

In RCFSTCMP, external file names and other information specified in the SELECT statements can be changed to suit site standards and/or to satisfy platform compatibility requirements.

## File Definition Statements

In RCFSTCMP, the input and output `FD` statements may be changed to suit site standards and/or to satisfy platform compatibility requirements. For example: "BLOCK CONTAINS 0 CHARACTERS" is frequently used on IBM platforms but not on HP platforms.



---

## Maximum-COBOL-record-length

The length of this field dictates the maximum **COBOL-RECORD-LENGTH** that can be passed to RCFSTXML 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 in both RCFSTCMP and RCFSTXML (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.

## RCFSTCMP Compile Errors

The following error messages may be displayed by RCFSTCMP in the event that the input CRD could not be interpreted. If an error message is issued, no CRD object file will be produced.

Error Code	Error Text	Reason
+0101	NO FIELDS IDENTIFIED ON RECORD DEFINITION	No fields could be identified on the input CRD.
+0102	TOO MANY FIELDS ON RECORD DEFINITION	The number of CRD fields exceeded the number allowed in the compiler's table. Increase this value.  (See <a href="#">Maximum-number-of-fields</a> in the <a href="#">User Maintained Variables</a> 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.
+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 RCFSTCMP source code and place this activation key in the <b>VALUE</b> 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. The 30 days have now elapsed.  Please contact Redvers Consulting for an additional thirty day trial or to arrange payment.
+0201	TOO MANY CHARACTERS FOR LEVEL NUMBER	RCFSTCMP was expecting a COBOL level number in the CRD but found a string of more than two characters.  <i>This message also points to the offending line number within the CRD.</i>
+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.  <i>This message also points to the offending line number within the CRD.</i>
+0203	ILLEGAL DATA CLAUSE FOUND	An unsupported data clause was found in the field definition.  See <a href="#">Clauses not Supported</a> section.  <i>This message also points to the offending line number within the CRD.</i>

+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. <i>This message also points to the offending line number within the CRD.</i>
+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. <i>This message also points to the offending line number within the CRD.</i>
+0207	EXCEEDED MAXIMUM TAG SIZE	An override tag name of more than 100 characters was encountered. The Superfast level interface is not designed to handle tag names of more than 100 characters. Please use one of the other Redvers COBOL XML Interface levels. <i>This message also points to the offending line number within the CRD.</i>
+0208	INVALID FILL CONSTANT	An invalid override fill constant was found in an override tag name in the CRD. The value found didn't match one of the allowable fill constants: "L" for low-values, "H" for high-values, "S" for spaces, "Z" for zeroes and "Q" for quotes. (See "Override Fill Constants" section in the <a href="#">RCFSTCOB</a> parser manual for details.) <i>This message also points to the offending line number within the CRD file.</i>
+0209	INCOMPLETE FILL CONSTANT TAG	An override fill constant was found in an override tag name in the CRD but the override was not completed with the ">" character. (See "Override Fill Constants" section in the <a href="#">RCFSTCOB</a> parser manual for details.) <i>This message also points to the offending line number within the CRD file.</i>
+0210	TAG DOES NOT START WITH AN ALPHABETIC CHARACTER	All XML tags must start with an alphabetic character. <i>This message also points to the offending line number within the CRD.</i>
+0211	TAG NAME CONTAINS INVALID XML CHARACTERS	XML tag names are confined to using only alphabetic, numeric, "-", "_", ":" or "." characters. <i>This message also points to the offending line number within the CRD.</i>
+0214	ATTRIBUTES MAY NOT BE GROUP LEVEL ITEMS	Attributes (tag names ending with "=") must be elementary data items. <i>This message also points to the offending line number within the CRD.</i>

+0215	ATTRIBUTE TAGS MAY NOT OCCUR MORE THAN ONCE	Attribute tags (those ending with "=") may occur only once in the parent/group level tag. <i>This message also points to the offending line number within the CRD.</i>
+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. <i>This error would normally have been caught in the compile stage.</i>
+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. <i>This error would normally have been caught in the compile stage.</i>
+0222	MORE THAN ONE ROOT ELEMENT FOUND	The structure of the CRD may not have more than one root level. <i>This message includes the offending field tag name.</i>
+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. <i>This message includes the offending attribute name.</i>
+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. <i>This message includes the offending attribute name.</i>
+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. <i>This message includes the offending field tag name.</i>

# RCFSTXML 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 +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	The <b>CRD-RECORD-COUNT</b> parameter was zero. Probable causes are that the previous recompilation by RCFSTCMP failed in some way or the calling application hasn't loaded the compiled CRD in <b>CRD-OBJECT-AREA</b> .
+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	The total logical length of <b>CRD-OBJECT-AREA</b> , exceeds the size of the maximum <b>COBOL-RECORD</b> permitted. (See <a href="#">Maximum-COBOL-record-length</a> in the <a href="#">User Maintained Variables</a> section.)
+0110	RECORD DEFINITION / LINKAGE MISMATCH	The logical length of <b>CRD-OBJECT-AREA</b> was not the same as <b>COBOL-RECORD-LENGTH</b> passed in linkage. Probable causes are that the CRD needs to be recompiled by RCFSTCMP, 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 <b>CRD-OBJECT-AREA</b> .
+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 RCFSTXML source code and place this activation key in the <b>VALUE</b> clause literal for the last field definition in working storage. Then recompile.

+0190	30 DAY TRIAL PERIOD EXPIRED OR CALL LIMIT REACHED	<p>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 RCFSTXML more than 100 times.</p> <p>Please contact Redvers Consulting for an additional thirty day trial or to arrange payment.</p>
+0301	CALL SEQUENCE ERROR	<p>A call has been made to the interface after processing has completed.</p> <p>Likely explanations are that <b>FEEDBACK-CODE</b> 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 <b>COBOL-RECORD</b>).</p>
+0302	PARAMETERS MUST NOT CHANGE AFTER INITIAL CALL	<p>One or more of the input calling parameters was found to have changed after the first call to the interface.</p> <p>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.</p>
+0310	ATTEMPTED TO GENERATE MULTIPLE ROOT DOCUMENT	<p>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.</p> <p>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 <b>FILLER</b>.</p>
+0330	XML DOCUMENT LENGTH EXCEEDS DOCUMENT AREA	<p>When building the XML document RCFSTXML has attempted to address beyond the length of the <b>XML-DOCUMENT</b> area in the calling program.</p> <p>Ensure that <b>XML-DOCUMENT-LENGTH</b> contains the full length of the <b>XML-DOCUMENT</b> area in the calling program. Otherwise the size of <b>XML-DOCUMENT</b> must be increased in the calling program.</p>

# Index

## A

account manager, 8, 20, 45  
 activation key, 7  
 apostrophes, 7, 49  
 arrays, 14, 25, 26  
 ASCII, 13  
 attributes, 10, 18

## B

base64, 13  
 Binary Fields, 13  
 binary zeroes, 40  
 BLANK WHEN ZERO, 14  
 BLOCK CONTAINS, 48

## C

Calling, 32  
 Calls, 34  
 carriage return, 50  
 CDATA, 46  
 Character Range, 45  
 Character References, 45  
 Cloaking Device, 7  
 COBOL Record, 32  
 COBOL Record Definition, 9  
 COBOL record length, 49  
 comment, 23  
 Comments, 47  
 compile, 7  
 Compile Errors, 51  
 Contact, 8  
 copybook, 21  
 CRD Object Area, 32  
 CRD Record Count, 32

## D

dimensions, 26

Document, 32  
 Document Length, 33, 34  
 Document Type Definition, 24, 34  
 double quotes, 7  
 DTD, 9, 19, 20, 24, 34, 49  
 DTD-headers, 47, 49  
 dummy fields, 28, 29  
 duplicate records, 29

## E

EBCDIC, 13  
 empty elements, 9, 14  
 empty fields, 9, 19, 20, 46  
 end tags, 9  
 End-of-line-chtrs, 50  
 entity declarations, 45  
 execution time, 6  
 Extensible Markup Language, 5, 45  
 external file names, 48

## F

fatal, 54  
 FD, 48  
 Feedback Code, 33, 34, 54  
 Feedback Messages, 54  
 Feedback Text, 33, 34, 54  
 field editing, 18  
 field names, 9  
 formatting, 18

## H

hash total, 43  
 hierarchy, 15, 28  
 HIGH-VALUES, 34

## I

*imbedded sign*, 12, 13



include, 21

install, 7

## J

JUSTIFIED RIGHT, 14

## L

line feed, 50

low-values, 28, 40, 45

## M

many-to-many, 41

Maximum Document Size, 47

Maximum-COBOL-record-length, 49

Maximum-number-of-fields, 50

minOccurs, 20, 46

mixed content, 16

## N

namespaces, 11

new line, 50

non-tagged, 17

non-tagged fields, 16, 18

non-tagged groups, 30

null, 45

Number-of-DTD-headers, 50

Number-of-XML-headers, 49

## O

OCCURS, 14, 20, 25

OCCURS DEPENDING ON, 14

one-to-many, 41

optional, 14

optional elements, 20

Orphan Repeats, 30

override tag names, 10, 11, 16, 18, 19, 22

## P

Packed Fields, 13

Parameters, 32

parent, 30

picture, 12, 19

Point of Change, 26, 28

Processing Instructions, 47

program-id, 35, 48

## Q

quote, 49

## R

RCCRDIN, 7

RCCRDOBJ, 7

RCFSTCMP, 5, 6, 7, 32, 48, 51

RCFSTCOB, 5

RCFXCALL, 7

Record Length, 32, 34

REDEFINES, 14

repeating groups, 25

required, 46

required attributes, 19

required elements, 19

reserved words, 10

root element, 15

## S

Sample Program, 35

schema, 9, 19, 20

SELECT, 48

single quotes, 7

SOAP, 11

speech marks, 7

start tags, 9

STRING, 46

structure, 15

Structure Break, 39

SYNC, 14

SYNCHRONIZED, 14

## **T**

tools, 9

totals, 39, 43

## **U**

Unicode, 45

User Maintained Variables, 7, 23, 24, 48

## **W**

*warranty*, 48

WORKING-STORAGE, 7, 23

World Wide Web Consortium (W3C), 5, 45

## **X**

XML declaration, 7, 23, 49

XML-headers, 24, 47, 49



***Redvers Consulting Ltd***

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