Redvers Consulting Ltd

Redvers COBOL XML Interface

User Guide

Superfast Parser

RCFSTCOB Version 2.8

Contents

Preface

This document describes the installation and operation of the Redvers COBOL XML Interface programs RCFSTCMP and RCFSTCOB. 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 RCFSTCOB is a COBOL subroutine that returns the data in XML documents to COBOL applications in COBOL form. Output from RCFSTCMP provides RCFSTCOB with all the information it needs to read XML documents specific to an application RCFSTCOB does not attempt to validate XML documents but will return an error if well-formed XML syntax rules are broken.

RCFSTCOB is the counterpart to RCFSTXML which uses a similar process to generate XML documents from a COBOL record.

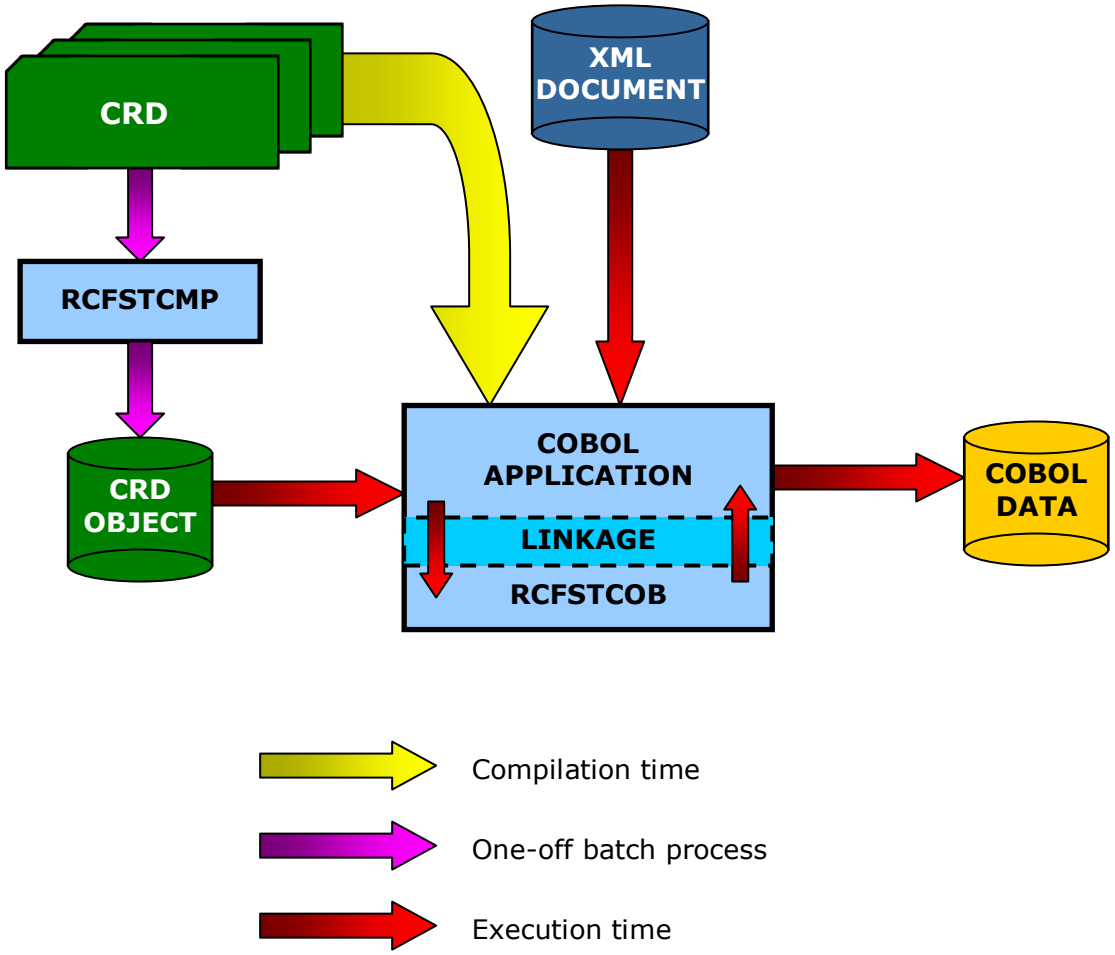
This User Guide can be found on the internet at:
http://www.redversconsulting.com/downloads/user_guides/RCFSTCOB_2.8_user.pdf

Copyright 2012 Redvers Consulting Ltd.

Overview

RCFSTCOB is a COBOL subroutine that is compiled and linked into application programs in order to parse well-formed XML documents. This particular interface level was built for large scale applications requiring fast and efficient XML parsing. 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 format of the COBOL data to be returned. 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, types etc. At execution time, application programs load this CRD object file into working storage and then pass it to RCFSTCOB with the XML document in a CALL statement.



Installation

RCFSTCOB 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 RCFSTCOB 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.

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

When running a free 30 day trial, the sample calling program RCFCCALL 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.

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 returned from the parser subroutine (RCFSTCOB). This CRD is also used to match element and attribute tag names with COBOL data names so that the CRD can be populated with data from the XML input. 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 web site.

Field Names

Fields in the CRD are populated with the data from attributes and elements in the XML document when their field names match XML tag names. When a group level field name on the CRD matches an XML tag, all subordinate fields in the group are initialized. Non-matching fields in the CRD with no matching parents will remain unchanged and elements in the XML document without a match will be ignored.

Default Tag Names

The COBOL field names in the CRD are the default names used to match XML start and end tags.

Eg:

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

Results in:

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

Note: *XML tag names are case sensitive so upper/lower case settings in the field name must be exactly the same as those of the tag name in order to achieve a match.*

Override Tag Names

XML frequently contains long tag names or names containing non-standard COBOL characters. In order to match with these tags, an override name is coded within "<" and ">" in the comment line, or lines, immediately following the field definition.

Eg:

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

Results in:

Field in CRD	Populated with

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

XML Attributes

XML attributes can be matched to COBOL fields by coding an "=" at the end of the override tag name.

Eg:

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

Results in:

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

XML Namespaces and SOAP

The Redvers COBOL XML Interface supports XML namespaces and Simple Object Access Protocol (SOAP) by loading data from XML elements when their tag names (including their prefixes) match override tag names in the CRD.

Eg:

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>

Results in:

Fields in CRD	Populated with
03 TV-program.	
05 SOAP-NAMESP PIC X(34). * <xmlns:SOAP-ENV=>	"http://schemas.xmlsoap.org/soap "
05 US-NAMESPACE PIC X(34). * <xmlns:us=>	"http://www.easternstandardtime.com "
05 prog-name PIC X(20).	"Sunday Night Live "
05 PROG-UK-TIME PIC X(05). * <uk:time>	"10:00"
05 PROG-US-TIME PIC X(05). * <us:time>	"05:00"

Note: *The xmlns:uk namespace declaration was deliberately left out of the CRD to show that the interface does not need namespace declarations to parse namespace tag names.*

From release 2.6, the interface also stores all the namespace declarations at the start of the instance document. If a prefix/tag match cannot be made directly with a field definition in the CRD, the interface will attempt to match the tag using any other prefix declared with the same URI.

From release 2.7, additional namespace declarations, not even defined in the instance document can be stored before parsing commences - See Alternative Namespace Prefixes for further details.

PICTURE Clause

Numeric Fields

Fields defined in the CRD with a display (zoned decimal) numeric picture clause, are populated as if they were the receiving operand of a COBOL **MOVE** statement after any non-numeric characters have been removed and the decimal point aligned.

Eg:

XML Document
<prog-date>31/12/2002</prog-date> <prog-duration>.333</prog-duration> <prog-cost>$2,500,000.00DB</prog-cost>

Results in:

Fields in CRD	Populated with
03 prog-date PIC 9(8).	31122002
03 prog-duration PIC 999V999.	000333
03 prog-cost PIC S9(9)V99.	0025000000}

Alphabetic, Alphanumeric and Numeric Edited Fields

Fields defined in the CRD as alphabetic, alphanumeric or numeric edited, are populated as if they were the receiving operand of a COBOL alphanumeric **MOVE** statement with no prior editing, except that numeric edited fields are moved right justified.

Eg:

XML Document
<prog-name> Sunday Night Live </prog-name> <prog-time>10:00</prog-time> <prog-duration>.333</prog-duration> <prog-cost>$2,500,000.00DB</prog-cost>

Results in:

Fields in CRD	Populated with
03 prog-name PIC X(20).	" Sunday Night Live "
03 prog-time PIC X(5).	"10:00"
03 prog-duration PIC ZZZ.999.	" .333"
03 prog-cost PIC Z(11)9.99-.	" $2,500,000.00DB"

Note: *In the example above,* ***prog-cost*** *is not loaded with formatting consistent with the CRD picture clause. This is because formatting cannot convert from one edited format directly to another. If another edited format is required, the field must be first loaded as a display numeric then moved to the new edited format.*

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 a format known as base64 is used. 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.

Unfortunately there is not yet any standard way to indicate that XML elements contain base64 characters, so the interface assumes that if a field is defined as binary or packed decimal in the CRD then it is to convert the data in the matching element from base64 to binary or packed decimal (depending on the picture **USAGE** clause).

Eg:

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

Results in:

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

Note: *On Little-endian platforms the base64 characters for* ***prog-revenue*** *will actually be "Fc1bBw==" to produce a 1234567.89 binary value.*

Note2: *In order to load a numeric XML element into a binary or packed COBOL field, you must first parse it into a display numeric CRD field, then use a COBOL* ***MOVE*** *into a field defined in the calling program as binary or packed.*

Other Clauses

BLANK WHEN ZERO

If a field is defined in the CRD as numeric but the XML element is empty, the field will be populated with spaces, otherwise it will be populated as a numeric field.

JUSTIFIED RIGHT

As in COBOL, this clause will cause the field to be populated from the rightmost position for alphabetic and alphanumeric cases.

OCCURS

Single dimension arrays can be defined in the CRD using the **OCCURS** clause in order to load multiple occurrences of matching XML elements in a single call. The clause can be used at group or elementary level. If the XML document contains more than one dimension of repeating data, this is handled by issuing multiple calls – see Repeating Groups for details. This approach allows the interface to process an unlimited number of occurrences in unlimited dimensions (which is the case for XML documents) using only a small amount of storage.

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. As stated previously, fields in the CRD are populated from XML elements based on matching CRD field names with XML tag names. When a parent XML tag matches a previously unpopulated CRD group level, the group level area is initialised and subsequent elementary fields are populated from matching child XML elements. XML data cannot be loaded directly into COBOL fields at group level because of the free-format nature of XML. The exception to this rule is when mixed content elements are encountered - see Mixed Content Elements later in this section for details.

Matching can take place in any sequence but group and elementary matches should occur in accordance with their hierarchy within the CRD structure, otherwise the application program will need to make additional calls to the interface in order to return all the XML elements.

Eg:

XML Document
<TV-listings> <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> <broadcast-date>11/12/2002</broadcast-date> </TV-listings>

Results in:

Fields in CRD		Populated with
01 TV-LISTINGS.		
03 broadcast-date	PIC 9(8).	11122002
03 producers-cut	PIC 9(9)V99.	1000000.00
03 channel-number	PIC 99.	5
03 TV-program.		
05 prog-time	PIC X(05).	"10:00"
05 prog-review	PIC X(200).	Spaces
05 prog-name	PIC X(20).	"Sunday Night Live "

In the previous example, no match is made for **<TV-listings>** because XML tags are case sensitive; **<channel>** is ignored because it is not on the CRD; **<channel-number>** is matched and populated; **<TV-program>** is matched at group level and the group area is initialized; **<prog-name>** is matched and populated; **<prog-time>** is matched and populated; **<prog-rating>** is ignored because it is not on the CRD; **<broadcast-date>** is matched and populated; the CRD field **producers-cut** remains unchanged from its original value.

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 translation 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 name in the CRD.

FILLER/null fields are populated according to their position within the matching group. If they are not required by the calling application they need not be coded.

Eg:

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

Results in:

Fields in CRD	Populated with
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 "
* <>	

Advanced Techniques

Override Fill Constants

As stated at the start of the Field Names section, CRD field groups are initialized when their group level field name (or override tag name) matches an XML element tag name. This initialization defaults to the setting of each field in the group as if the group level field was the subject of a COBOL **INITIALIZE** statement. Although this may be the desired effect in most cases, it does produce a problem if a distinction between an empty XML element verses an absent XML element is required.

An example of such a situation might be a TV listings application that receives a file of updates in the form of an XML document. If a television critic wished to withdraw a review of a TV program, the application would need to distinguish between an intentional update that places spaces in the program review field, compared with no update to the program review field.

In order to solve this dilemma, release 2.5 introduces the facility to override the default initialization so that fields subjected to a field-name/tag-name match at group/parent level can be filled with a figurative constant of **SPACES**, **ZEROES**, **QUOTES**, **LOW-VALUES** or **HIGH-VALUES**. The override is applied using the CRD override tag name facility by placing a forward slash ("/") followed by the first letter of the figurative constant (upper or lower case) to the right of the override tag name. If no override tag name is required, the fill constant can be applied in isolation by coding "**</Z>**", "**</s>**" etc.

Eg:

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

Results in:

Fields in CRD	Populated with
03 TV-program.	
05 prog-name PIC X(20). * </z>	"Sunday Night Live "
05 PROG-TIME PIC X(05). * <time/L>	Low-values
05 PROG-REVIEW PIC X(200). * <review/h>	Spaces

In the previous example, when **<TV-program>** is matched to the **TV-program** CRD group, **prog-name** is initialized to **ZEROES**, **PROG-TIME** is initialized to **LOW-VALUES** and **PROG-REVIEW** is initialized to **HIGH-VALUES**. Then, "Sunday Night Live " is moved to **prog-name** and **SPACES** are moved to **PROG-REVIEW** because of the empty **<review>** element. When processing is complete, the resulting CRD indicates that while there were updates for **prog-name** and **PROG-REVIEW**, no updates were present for **PROG-TIME** because it remains set to its override fill constant of **LOW-VALUES**.

If a CRD containing override fill constants is also used by a Redvers COBOL XML generator module, the "/" character and any subsequent character will be ignored.

Note: *If an override fill constant is coded at group level it will apply to all subordinate fields and field groups unless superseded by a lower level override fill constant.*

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 parser 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 be populated from XML data content.

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 data will be moved to this field. 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 * <(Exclude_Me)>	PIC X(20).

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*.

Repeating Groups

In business applications it would be rare for an XML document to contain only a single set of information details. Elements or element groups are often repeated to reflect multiple sets and subsets of information within the XML document. In order to feed this repeating data to a calling application, single dimension arrays can be defined in the CRD using the COBOL OCCURS clause. Alternatively, if the number of occurrences is unknown or more than one dimension of repeating data is possible, multiple calls can be made to RCFSTCOB which returns the next logical set of repeated data and any related structure.

Using OCCURS

Occurrences of a group or elementary field in the CRD are populated for all matching repeated XML elements. If XML elements repeat more times than will fit in the array, these elements will be returned in subsequent calls. If XML elements repeat fewer times than the number of occurrences in the OCCURS clause, surplus occurrences will remain unchanged, therefore it is important to achieve a match at the group level first to initialize the table.

Eg:

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>
  <TV-program/>
  <TV-program>
    <prog-name>Weather</prog-name>
    <prog-time>11:55</prog-time>
  </TV-program>
</channel>
```

Results in:

Fields in CRD	Occurrence	Populated with
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"
05 TV-program OCCURS 5.	2	
07 prog-name PIC X(17).	2	"News "
07 prog-time PIC X(5).	2	"11:30"
05 TV-program OCCURS 5.	3	
07 prog-name PIC X(17).	3	spaces
07 prog-time PIC X(5).	3	spaces
05 TV-program OCCURS 5.	4	
07 prog-name PIC X(17).	4	"Weather "
07 prog-time PIC X(5).	4	"11:55"
05 TV-program OCCURS 5.	5	
07 prog-name PIC X(17).	5	spaces
07 prog-time PIC X(5).	5	spaces

Note: *As* ***CHANNEL*** *was coded in capitals, the fifth occurrence of* ***prog-name*** *and* ***prog-time*** *was initialized only when the group level <TV-program> matched with* ***TV-program*** *in the CRD. The third occurrence of* ***prog-name*** *and* ***prog-time*** *was initialized when the group level <TV-program> matched with* ***TV-program*** *and again when the empty <TV-program/> element was matched.*

Using Repeated Calls

An unlimited number of occurrences and dimensions (which is often the case for XML documents) can be parsed most efficiently by the use of repeated calls to the interface module. In the first call, a scan is made of the XML document and all CRD fields with matching element names are populated. If the number of matched XML elements exceeds those on the CRD, excess elements overflow into the work area. When subsequent calls are made, the elements in the work area are read and loaded into the CRD in a sequence consistent with their position in the XML document hierarchy.

The example below parses a two dimensional array from XML for multiple **<TV-program>** elements within multiple **<channel>** elements using repeated calls and a smaller CRD.

Eg:

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>
```

After first call:

Fields in CRD		Populated with
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"

After second call:

Fields in CRD		Populated with
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"

After third call:

Fields in CRD		Populated with
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"

Orphan Repeats

In the following example **<prog-name>** and **<prog-time>** are repeated without the presence of a parent element for each occurrence. This configuration can be parsed by the interface but the XML document must be responsible for keeping **<prog-name>** and **<prog-time>** in step with each other by providing empty elements for any unpopulated fields. If this is not done, data from unrelated elements could be returned as a related set.

Eg:

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>

After first call:

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

After second call:

Fields in CRD		Populated with
03 channel.		
05 channel-number	PIC 99.	05
05 prog-name	PIC X(20).	"News "
05 prog-time	PIC X(5).	spaces

After third call:

Fields in CRD		Populated with
03 channel.		
05 channel-number	PIC 99.	05
05 prog-name	PIC X(20).	"Weather "
05 prog-time	PIC X(5).	"11:55"

Optimizing Performance

When RCFSTCOB converts XML to COBOL format, it uses a work area to store repeated occurrences of XML elements. For most applications, the size of the XML document will be relatively small and so the work area size will not be an issue. However, for large XML documents it may be wise to control the amount of data initially read from the XML input. This can be done by manipulating the high level field names in the CRD in order to produce a smaller Unit of Data Transfer.

Unit of Data Transfer

When RCFSTCOB begins reading an XML document it searches for the first field/tag name match on the CRD. Once found, the entire content of the matched element becomes the first Unit of Data Transfer (UDT) which is read from the XML document. Occurrences of matched child elements are placed directly into the CRD fields while any repeated elements that won't fit, are written to the work area.

In subsequent calls, the work area is scanned in order to return the repeated data in a logical sequence (starting with the lowest elements in the hierarchy). Once all elements have been loaded from the work area, further calls initiate processing the next UDT.

If a UDT is the result of a field/tag match at document root level, all duplicate elements in the XML document will be loaded into the work area, then repeatedly read, scanned and rewritten. However, if the UDT consists of only one or two elements, many more calls to RCFSTCOB will be required in order to populate all fields in the CRD.

For best results, the CRD should be designed so that the UDT is at CRD record level. That is, the first XML element tag to match a field on the CRD should contain the minimum amount of information needed to populate one entire CRD.

In the following example the top level group field in the CRD (`TV-LISTINGS`) is deliberately coded in capital letters so that it doesn't match with the XML root element. Instead, **<TV-today>** is the first XML element on the input that matches a field on the CRD and so it becomes the UDT. **<TV-today>** contains just enough data to populate the whole CRD with the minimum of repeated elements, making it the optimum UDT.

Eg:

XML Document

```
<TV-listings>
 <TV-today>
  <broadcast-date>11/12/2002</broadcast-date>
  <channel>05
    <prog-name>Sunday Night Live</prog-name>
    <prog-time>10:00</prog-time>
  </channel>
  <channel>06
    <prog-name>Westenders</prog-name>
    <prog-time>08:00</prog-time>
  </channel>
 </TV-today>
 <TV-today>
  <broadcast-date>12/12/2002</broadcast-date>
  <channel>05
    <prog-name>Monday Night Live</prog-name>
    <prog-time>10:00</prog-time>
  </channel>
 </TV-today>
</TV-listings>
```

CRD

```
 01  TV-LISTINGS.
   03  TV-today.
     05  broadcast-date                PIC 9(8).
     05  channel.
       07  channel-number              PIC 99.
*          <>
       07  prog-name                   PIC X(23).
       07  prog-time                   PIC X(5).
```

Calling RCFSTCOB

Parameters

A call to RCFSTCOB 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**.

XML-DOCUMENT (input)

This is the storage area containing the XML document to be parsed.

XML-DOCUMENT-LENGTH (input & output)

This S9(8) binary field must be set to the actual length of the XML document text within **XML-DOCUMENT** and not changed during the parsing process. When RCFSTCOB has fully parsed the document, it will change this value to zero in order to indicate that the document has been fully parsed and to help prevent potential logic loops. The maximum **XML-DOCUMENT-LENGTH** supported by the interface is 99,999,999 characters.

COBOL-RECORD (output)

This is the top level field name in the CRD. It will hold the COBOL data returned from RCFSTCOB.

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**.

FEEDBACK-CODE (output)

This S9(4) binary field is set by RCFSTCOB 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 or 10 (end of processing) 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 RCFSTCOB with diagnostic information on the results of each call. For successful calls it is populated with a message showing the position of the first field updated by the call (as it appears in the CRD) and the number of fields populated so far. 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 RCFSTCOB

The First Call

When the interface is called for the first time, RCFSTCOB will validate the calling parameters and read the first Unit of Data Transfer (UDT) from **XML-DOCUMENT** (see Optimizing Performance for more details on UDT's). RCFSTCOB attempts to match all attribute and element tags in the UDT with field names on the CRD. All available matched fields in the CRD are populated. Any repeated XML elements within the UDT are written to the work area which, if used, becomes the input for the next call.

After successful completion, **FEEDBACK-CODE** is set to zero and **FEEDBACK-TEXT** is populated with a message showing the position of the first field updated (as it appears in the CRD) and the number of fields populated.

Subsequent Calls

Each subsequent call to RCFSTCOB initiates a pass of either the work area or the next UDT from **XML-DOCUMENT**. For each pass, all matching tags and attributes in the CRD are populated while any repetitions that cannot fit in the layout are passed to the work area.

There are three ways in which an application program can find out what CRD fields have been updated in each subsequent call to the interface:

- Use the numeric value in positions 28 through 31 of the returned **FEEDBACK-TEXT.** This value represents the relative field position, within the CRD, of the first field to be updated by the interface. Interrogating this value tells the application program the precise group level or elementary field that has been populated in the call. This technique should be used for more stable CRD layouts as any additions or removals from the CRD will cause the relative field positions below to change, thereby requiring procedure code changes.

- Test key fields for change, starting with the highest level key. In most data structures a repetition of a set of corresponding data is associated with a key field that identifies the group. Using the TV listings example, testing for a change in channel number would initiate new channel processing logic in the application program; if the channel number is the same, testing for a change in program name or time would initiate the new TV program processing, etc.

- Move a figurative constant (**SPACES**, **ZEROES**, **QUOTES**, **LOW-VALUES** or **HIGH-VALUES**) to the entire **COBOL-RECORD** (defined by the CRD) before the **CALL** to RCFSTCOB and test for fields that don't contain the figurative constant after the **CALL**. Once the interface passes back data in the **COBOL-RECORD** to the application, it has no further use for it. This is the simplest way to identify populated fields but any useful data previously held in the **COBOL-RECORD** may need to be stored elsewhere.

After successful completion of each pass, **FEEDBACK-CODE** is set to zero and **FEEDBACK-TEXT** is populated with a message showing the position of the first field updated and the number of fields populated so far.

The Last Call

When there is no more **XML-DOCUMENT** data to be parsed, RCFSTCOB sets **FEEDBACK-CODE** to 10 (indicating "end of processing") and populates **FEEDBACK-TEXT** with a message indicating that processing is complete. No data is transferred to the CRD in this call.

Note: *If RCFSTCOB remains in memory after the last call, it can be reused to parse another XML document for the same, or a different, CRD by repopulating the calling parameters and restarting the call sequence from the first call.*

Sample Program Calling RCFSTCOB

```
…
000040*****************************************************************
000050*  This sample code shows how RCFSTCOB is called to produce      *
000060*  COBOL records of data from an XML document.                   *
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 code 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***  Storage area containing the XML document to be parsed:
000350 01  XML-DOCUMENT                PIC X(16000).
000360
000370***  Start of COBOL Record Definition (CRD)
000380 01  TV-listings.
000390     03  channel.
000400         05  channel-number      PIC 99.
000410*            <number=>
000420         05  channel-name        PIC X(20).
000430*            <>
000440         05  TV-program          OCCURS 10.
000450             07  prog-name       PIC BBX(20)BB.
000460             07  prog-time       PIC X(05).
000470             07  prog-rating     OCCURS 1.
000480*                <rating>
000490                 09  rating-txt1 PIC X(20)B.
000500*                    <>
```

```
000510                09  mark        PIC Z9.
000520                09  rating-txt2 PIC BX(20).
000530*                   <>
000540          05  channel-owner       PIC X(20).
000550*             <>
000560***  End of COBOL Record Definition (CRD)
000570
000580 01  OTHER-PARAMETER-FIELDS.
000590     03  CRD-RECORD-COUNT        PIC S9(8) BINARY VALUE ZERO.
000600     03  XML-DOCUMENT-LENGTH     PIC S9(8) BINARY.
000610     03  COBOL-RECORD-LENGTH     PIC S9(8) BINARY VALUE ZERO.
000620     03  FEEDBACK-CODE           PIC S9(4) BINARY VALUE ZERO.
000630     03  FEEDBACK-TEXT.
000640         05  FILLER              PIC X(27).
000650         05  FIRST-FIELD-LOADED  PIC 9(04).
000660         05  FILLER              PIC X(49).
000670
000680 01  MISCELLANEOUS-FIELDS.
000690     03  CRD-TABLE-SIZE          PIC 9(04)        VALUE ZERO.
000700     03  CRD-FLAG                PIC X            VALUE SPACE.
000710         88  START-OF-CRD                         VALUE SPACE.
000720         88  END-OF-CRD                           VALUE "E".
000730
000740 PROCEDURE DIVISION.
000750
000760 TOP-LEVEL SECTION.
000770*****************************************************************
000780*  This section loads the CRD object file into CRD-OBJECT-AREA   *
000810*  and calls RCFSTCOB to parse the content of XML-DOCUMENT into  *
000820*  the COBOL Record Definition area until RCFSTCOB returns an    *
000830*  end of processing status (FEEDBACK-CODE of 10).               *
000832*  XML-DOCUMENT will already contain the XML document to be      *
000834*  parsed and XML-DOCUMENT-LENGTH will already contain the       *
000836*  length of the XML document text.                              *
000840*****************************************************************
000850 TOP-ENTER.
000860
000870     COMPUTE CRD-TABLE-SIZE = LENGTH OF CRD-OBJECT-AREA
000880                            / LENGTH OF CRD-OBJECT-RECORD (1).
000890     MOVE LENGTH OF TV-LISTINGS  TO COBOL-RECORD-LENGTH.
000960
000970     INITIALIZE                     TV-LISTINGS
000972                                    FEEDBACK-TEXT.
000980
001010     PERFORM B-LOAD-CRD.
001020
001030     PERFORM C-CALL-RCFSTCOB.
001040
001050     PERFORM UNTIL FEEDBACK-CODE > ZERO
001060         PERFORM D-PROCESS-XML-INPUT
001070         PERFORM C-CALL-RCFSTCOB
001080     END-PERFORM.
```

```
001090
001100***  Display the processing complete message:
001110     DISPLAY FEEDBACK-TEXT.
001120
001130     STOP RUN.
001140
001150 TOP-EXIT.
001160     EXIT.
001170
001180
001610 B-LOAD-CRD SECTION.
001620*****************************************************************
001630*  This section reads the CRD object file into CRD-OBJECT-AREA.  *
001640*****************************************************************
001650 B-ENTER.
001660
001670     OPEN INPUT CRD-FILE.
001680
001690     PERFORM
001700       UNTIL   END-OF-CRD
001710       OR      CRD-RECORD-COUNT = CRD-TABLE-SIZE
001720         READ CRD-FILE
001730           AT END
001740             SET END-OF-CRD      TO TRUE
001750           NOT AT END
001760             ADD 1               TO CRD-RECORD-COUNT
001770             MOVE CRD-RECORD     TO CRD-OBJECT-RECORD
001780                                    (CRD-RECORD-COUNT)
001790         END-READ
001800     END-PERFORM.
001810
001820     CLOSE CRD-FILE.
001830
001840     IF  NOT END-OF-CRD
001850         DISPLAY "CRD-OBJECT-AREA TABLE IS FULL!"
001860         DISPLAY "CURRENT SIZE IS: " CRD-TABLE-SIZE
001870         STOP RUN
001880     END-IF.
001890
001900 B-EXIT.
001910     EXIT.
001920
001930
001940 C-CALL-RCFSTCOB SECTION.
001950*****************************************************************
001960*  This section executes the CALL to the interface and checks    *
001970*  the feedback code.                                            *
001980*****************************************************************
001990 C-ENTER.
002000
002010     CALL "RCFSTCOB"          USING CRD-OBJECT-AREA
002020                                    CRD-RECORD-COUNT
```

```
002030                                    XML-DOCUMENT
002040                                    XML-DOCUMENT-LENGTH
002050                                    TV-LISTINGS
002060                                    COBOL-RECORD-LENGTH
002070                                    FEEDBACK-CODE
002080                                    FEEDBACK-TEXT.
002090
002100     IF  FEEDBACK-CODE > 10
002110         DISPLAY "BAD RETURN FROM RCFSTCOB - FEEDBACK CODE IS "
002120                                    FEEDBACK-CODE
002130         DISPLAY "MESSAGE READS: "  FEEDBACK-TEXT
002140         STOP RUN
002150     END-IF.
002160
002170 C-EXIT.
002180     EXIT.
002190
002200
002210 D-PROCESS-XML-INPUT SECTION.
002220*****************************************************************
002230*  The following code uses FIRST-FIELD-LOADED in FEEDBACK-TEXT   *
002240*  to indicate the field number in the CRD that was first        *
002250*  populated in order to decide on what processing is required.  *
002260*  An alternative and more flexible approach would be to check   *
002270*  which key fields have changed as a result of the call.        *
002280*****************************************************************
002290 D-ENTER.
002300
002310     EVALUATE FIRST-FIELD-LOADED
002320       WHEN 1
002330***      process new document
002340         CONTINUE
002350       WHEN 2
002360***      process next channel
002370         CONTINUE
002380       WHEN 5
002390***      process next program
002400***      (for this to happen there must have been more than 10
002410***       programs for the channel)
002420         CONTINUE
002430       WHEN OTHER
002440***      process other field group changes
002450         CONTINUE
002460     END-EVALUATE.
002470
002540 D-EXIT.
002550     EXIT.
```

Selecting XML elements

From release 2.7, the Redvers COBOL XML Interface parser routine is capable of selective XML parsing, based on a simple **SELECT** statement, passed through the **FEEDBACK-TEXT** parameter. If the application sets the first 6 character positions of **FEEDBACK-TEXT** to "**SELECT**", selection logic will be invoked and the **SELECT** statement will be validated and processed.

SELECT Statement Format

Standard:

```
SELECT |<TagName>|* |<TagName AttributeName>|? |EQ| 'literal'
                                               |GT|
                                               |LT|
                                               |GE|
                                               |LE|
                                               |NE|
```

Or:

```
SELECT NEXT
```

|…|: *Option.*
***:** *Zero, one or many occurrences.*
?: *Zero or one occurrence.*

SELECT Statement Syntax

- **SELECT** must start in position 1.
- There must be at least one **<TagName>** or **<TagName AttributeName>** option present.
- There must be one **EQ**, **GT**, **LT**, **GE**, **LE** or **NE** option present.
- **'literal'** can be delimited by single or double quotes but not both.
- Single or multiple spaces within the statement are optional.
- **SELECT NEXT** can only be used after a previous valid **SELECT** statement.

SELECT Statement Use

To use the **SELECT** statement successfully, the application must identify the XML element or attribute to be tested using the **<TagName>** and/or **<TagName AttributeName>** options:

- **<TagName>** identifies an element within the root or previous **<TagName>** (if present).
- **<TagName AttributeName>** identifies an element and attribute within the root or previous **<TagName>** (if present).

The application program therefore places as many **<TagName>** and/or **<TagName AttributeName>** options in the **FEEDBACK-TEXT** parameter as are necessary to uniquely identify the XML element or attribute within the XML instance document.

The order of the **<TagName>** and/or **<TagName AttributeName>** options in the **SELECT** statement must be the same as their order in the XML hierarchy.

Currently only the **EQ** (equal), **GT** (greater than), **LT** (less than), **GE** (greater than or equal), **LE** (less than or equal) or **NE** (not equal) operators (in capitals) are supported, although this list may be extended in a future release.

The literal value must be coded as it would appear in the XML document – NOT as it would appear after it has been loaded into the CRD. If the length of the literal is less than the length of the element content, zeroes or spaces are assumed to the right or left of the literal depending on the picture clause in the CRD.

If the total length of the **SELECT** statement is longer than the length of the **FEEDBACK-TEXT** parameter (80 bytes), additional options may be passed to the parser routine in subsequent calls. When this occurs, no XML data is parsed and **FEEDBACK-TEXT** is returned with "**STORING SELECT DETAILS...**". An option or literal cannot be split across multiple calls.

If the **FEEDBACK-TEXT** parameter does not contain "**SELECT**" in the first 6 positions, the parser will return to normal mode and return XML data from the next Unit of Data Transfer (UDT).

The **SELECT NEXT** statement repeats the previous successful **SELECT** statement regardless of whether any normal mode calls were executed since the previous **SELECT** statement.

If the selection is successful, the CRD will be returned containing XML data from the selected element or attribute AND data from elements and attributes, both above and below the selection, in the XML hierarchy.

If no XML data content could be found matching the **SELECT** statement condition, **FEEDBACK-TEXT** is set to "**RUN COMPLETED – NO XML ELEMENTS SELECTED**" and **FEEDBACK-CODE** is set to +0010.

Note: *More sophisticated **SELECT** features may be added in future releases depending on customer demand.*

SELECT Statement Example

XML Document

```
<TV-listings>
  <TV-today>
    <broadcast-date>11/12/2010</broadcast-date>
    <channel number=”5”>
      <TV-program>
        <prog-name>Sunday Night Live</prog-name>
        <prog-time>10:00</prog-time>
      </TV-program>
    </channel>
    <channel number=”6”>
      <TV-program>
        <prog-name>Westenders</prog-name>
        <prog-time>08:00</prog-time>
      </TV-program>
    </channel>
  </TV-today>
</TV-listings>
```

FEEDBACK-TEXT

```
SELECT <TV-listings> <channel number> GT ”5”
```

Fields in CRD			Populated with
01	TV-LISTINGS.		
03	TV-today.		
05	broadcast-date	PIC 9(8).	11122010
05	channel.		
07	channel-nbr <number=>	PIC 99.	06
07	TV-program.		
09	prog-name	PIC X(20).	“Westenders ”
09	prog-time	PIC X(5).	“08:00”

In the previous example **<TV-today>** was omitted from the **SELECT** statement in **FEEDBACK-TEXT** because **<channel>** is a unique element name within **<TV-listings>**. In fact **<TV-listings>** could also have been omitted for the same reason.

Note: *After the **SELECT** on <**channel number**> completes, fields higher in the CRD hierarchy (“**broadcast-date**”) and lower in the hierarchy (the first “**prog-name**” and “**prog-time**”) are also populated.*

Alternative Namespace Prefixes

In response to the increased use of XML namespaces, release 2.7 gave applications the capability to parse instance documents containing namespace prefixes, not previously known to the application.

In order to parse elements and attributes with namespace prefixes different from those coded in the CRD, RCFSTCOB stores namespace declaration prefixes and URI's in the "**`Entity table`**", near the start of **`WORKING-STORAGE`**. If an input XML tag fails to match with a field name in the CRD, the entity table is checked to see if an alternative namespace prefix exists for the same URI. If it does, the alternative prefix is assumed for the remainder of the parse and the elements/attributes are loaded into the CRD.

The maximum number of entries in the entity table and the maximum length of the prefixes and URI's can be adjusted using the Maximum-number-of-entities, Maximum-entity-argument and Maximum-entity-length User Maintained Variables. See User Maintained Variables for details.

Namespace declaration information can be stored from any of three sources:

- The instance document root.
- Performing a pre-parse of the instance document, collecting all namespace declarations.
- A <**xmlnamespaces**> element held in the calling application.

Each option is discussed below:

The Instance Document Root

Namespace declaration details within the instance document root element are loaded into the entity table by default. For the majority of applications, no other namespace processing is necessary.

Note: *This default process does not load namespace declarations in the body of the instance document. If additional namespace declaration details are required, see The xmlnamespaces Element or Performing a Pre-parse sections.*

Performing a Pre-parse

To pre-load all namespace declaration details from within the body the XML instance document, a pre-parse of all attributes in the document is required. This processing can be switched on by the use of a special processing flag set in your copy of RCFSTCOB. For more information, please contact your account manager or use our "Contact" facility at: http://www.redversconsulting.com/contact.php.

The xmlnamespaces Element

Additional namespace declaration information can be passed to RCFSTCOB using a dummy <**xmlnamespaces**> element, inserted before the XML instance document by the application. This element can be written from scratch using an editor or the text can be copied from one or more **<schema>** root elements. Any non-namespace declarations are ignored. See xmlnamespaces Example.

The benefit from using the <**xmlnamespaces**> element is that any number of namespace declarations can be passed to RCFSTCOB, providing a complete list of all possible namespace prefixes and their URI's. This information provides the parser with a cross-reference of all possible alternative prefixes for each URI without the need for a pre-parse.

Note: *When the <**xmlnamespaces**> element is passed to RCFSTCOB, namespace declarations in the instance document root element are also loaded into the entity table. However, namespace declarations in the body of the instance document are not loaded. If namespace declaration details in the body of the instance document are also required, see Performing a Pre-parse section.*

xmlnamespaces Example

<xmlnamespaces> in the Application
<xmlnamespaces xmlns="http://www.redversconsulting.com/schema.xsd" xmlns:fred=”http://www.redversconsulting.com/schema.xsd” xmlns:dave="http://www.redversconsulting.com/schema.xsd" xmlns:bob="http://www.redversconsulting.com/schema.xsd" targetNamespace="http://www.redversconsulting.com/schema.xsd" attributeFormDefault="unqualified"/>
XML Document
<fred:TV-listings> <fred:broadcast-date>11/12/2010</fred:broadcast-date> <channel number=”5”> <bob:TV-program> <bob:prog-name>Sunday Night Live</bob:prog-name> <bob:prog-time>10:00</bob:prog-time> </bob:TV-program> </channel> </fred:TV-listings>

Fields in CRD		Populated with
01 TV-listings.		
03 broadcast-date	PIC 9(8).	11122010
03 channel.		
* <dave:channel>		
05 channel-nbr	PIC 99.	05
* <dave:number=>		
05 TV-program.		
07 prog-time	PIC X(5).	“10:00”
07 prog-name	PIC X(20).	“Sunday Night Live ”

In the example above, namespace declarations in <**xmlnamespaces**> mean that prefixes “fred”, “dave”, “bob” and no-prefix are all associated with the same namespace URI (http://www.redversconsulting.com/schema.xsd) and are therefore alternative definitions of each other. When **<TV-listings>** is parsed, tag-name/field-name matches occur for all CRD fields because all namespace prefixes are effectively the same.

Note: *The <**xmlnamespaces**> element must be added by the receiving application just before the call to the Redvers parser routine. It cannot be added by the sending application because all tag names starting with “xml” are illegal under XML syntax rules (this is why the name “xmlnamespaces” was chosen – it cannot be part of a valid XML instance document).*

Data Integrity

Character Range

RCFSTCOB accepts single byte characters in the hexadecimal range "00" through "FF". However, the end-of-text character (hex "03" in EBCDIC and ASCII) is used internally by the interface and is not loaded into the CRD. The end-of-text character is not within the XML character range defined by the W3C Extensible Markup Language (XML) 1.0 (Second Edition) definition.

Character References

Unicode character references (eg: `&#238;` = î) may passed to the CRD unchanged. No attempt is made to interpret their character form.

Entity References

Data is usually transferred from XML elements to CRD fields without alteration. However, if an XML entity is encountered, the entity name (along with the preceding "&" and trailing ";") are automatically substituted for the entity replacement text. The interface does this for the standard XML predefined entities (listed below) as well as general, internal, parsed entities defined in the incoming DTD.

Entity Reference	Description	Character
>	greater than	>
<	less than	<
&	ampersand	&
'	apostrophe	‘
"	double quote	“

If general internal parsed entity declarations are encountered in the input DTD their details are stored in a **WORKING-STORAGE** table so that substitution can take place as they are encountered on the input data stream. The maximum number of entries in the entity table and the maximum length of the entity values can be adjusted using the Maximum-number-of-entities and Maximum-entity-length User Maintained Variables. See User Maintained Variables for details.

CDATA

Character data strings, within the **<![CDATA[** and **]]>** delimiters, are loaded into CRD fields entirely unchanged. This includes entity references and other XML markup characters. CDATA can occupy the entire content of an element or it may just form part of it.

The "**<![CDATA[**" and "**]]>**" literals themselves are NOT passed to the CRD fields.

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

Due to the fact that processing instructions do not actually contain data that can be moved into a COBOL field these are ignored by the interface.

Comments

Due to the fact that comments do not actually contain data that can be moved into a COBOL field these are ignored by the interface.

White Space

Due to the nature of COBOL applications RCFSTCOB assumes all XML elements contain the attribute: xml:space="preserve". In most circumstances this will have the desired

effect, as spaces within the content of XML elements are preserved and spaces between elements are ignored.

If settings other than those described above are required, contact your account manager or use our "Contact" facility at: http://www.redversconsulting.com/contact.php.

User Maintained Variables

One of the features of RCFSTCOB 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 RCFSTCOB 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.

Work-area-size

The maximum number of bytes to be used for storing duplicated elements. This is the storage area mentioned in the Optimizing Performance section when duplicate elements exist in a Unit of Data Transfer (UDT). The requirement for work area space can be minimised by designing the CRD so that the first XML element tag to match a field on the CRD contains the minimum amount of information needed to populate one entire CRD – see Optimizing Performance**.** This UMV is defaulted to 4,096 but if more space is required it can be increased. Similarly, if storage is limited, this value can be decreased to the total length of the maximum duplicated elements possible in a UDT.

Maximum-number-of-entities

The entity table is used to store general, internal, parsed entity declarations and namespace declarations. The maximum number of these declarations is defaulted to 20. If an input XML document contains more than 20 declarations this UMV can be increased. Similarly, if storage is limited, this value can be decreased to the largest number of expected declarations.

Maximum-entity-argument

For each declaration stored in the entity table, the maximum entity argument length or namespace prefix length is defaulted to 30. If an input XML document contains entity arguments or namespace prefixes longer than 30 characters in length, this UMV can be increased.

Maximum-entity-length

For each declaration stored in the entity table, the maximum entity value length or namespace URL length is defaulted to 100. If an input XML document contains entity values or namespace URLs longer than 100 characters in length, this UMV can be increased.

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.

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 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.
+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 **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. 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. *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 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. *This message also points to the offending line number within the CRD.*
+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 for details.) *This message also points to the offending line number within the CRD file.*
+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 for details.) *This message also points to the offending line number within the CRD file.*
+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.*

RCFSTCOB Feedback Messages

A **FEEDBACK-CODE** less than +100, indicates processing has completed successfully and that **FEEDBACK-TEXT** contains diagnostic information.

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

The +300 series indicate fatal errors caused by inconsistencies in the XML document.

FEEDBACK-CODE	FEEDBACK-TEXT
+0000	CRD LOADED FROM FIELD NBR: 9999 FIELDS LOADED SO FAR: 99999999

FEEDBACK-CODE	FEEDBACK-TEXT	Reason
+0010	RUN COMPLETED - ALL XML ELEMENTS LOADED	RCFSTCOB has successfully loaded all matched elements into the COBOL Record Definition (CRD). No fields were populated in this call. *This is not an error feedback*.
+0010	RUN COMPLETED - NO XML ELEMENTS LOADED	RCFSTCOB did not find any data on the input XML document that could be loaded into the CRD. Either the document was empty or no XML tag names matched with CRD field names. *This is not an error feedback*.
+0010	RUN COMPLETED - NO XML ELEMENTS SELECTED	RCFSTCOB did not find any XML content that satisfied the SELECT statement provided. SELECT logic proceeds from the point the last call completed, down the XML document. Matching data content could exist earlier in the document. *This is not an error feedback*.
+0101	NO FIELDS IDENTIFIED ON RECORD DEFINITION	The **CRD-RECORD-COUNT** 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 **CRD-OBJECT-AREA**.
+0103	PROCESSING EXCEPTION. PLEASE CONTACT REDVERS CONSULTING	There has been an internal logic error within the program. Please contact your Redvers Consulting account manager.

+0110	**RECORD DEFINITION / LINKAGE MISMATCH**	The logical length of **CRD-OBJECT-AREA** was not the same as **COBOL-RECORD-LENGTH** 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 **CRD-OBJECT-AREA**.
+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 RCFSTCOB 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 RCFSTCOB more than 100 times. Please contact Redvers Consulting for an additional thirty day trial or to arrange payment.
+0301	**CALL SEQUENCE ERROR**	A call to RCFSTCOB has been made at an illogical processing phase. Likely explanations are that **FEEDBACK-CODE** was not checked after a previous unsuccessful call or that all XML data has already been loaded and the previous call returned a +0010 **FEEDBACK-CODE**.
+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 parse a new XML document before completing the previous one.
+0305	**ENTITY TABLE IS FULL**	Too many entity definitions were found at the start of the input DTD for the table. (See Maximum-number-of-entities in the User Maintained Variables section.) *This message also identifies the offending entity name in the DTD.*
+0306	**ENTITY VALUE IS TOO LONG FOR ENTITY TABLE**	An entity value defined in the DTD is too long to fit in the entity table. (See Maximum-entity-length in the User Maintained Variables section.) *This message also identifies the offending entity name in the DTD.*

+0308	WORK AREA IS FULL	The length of all duplicated elements in the Unit of Data Transfer (UDT) has exceeded the size of the work area. (See Work-area-size in the User Maintained Variables section.)
+0310	TOO MANY NAMESPACES FOR ENTITY TABLE	There were more namespace definitions (xmlns) at the start of the XML document than would fit into the entity table. The entity table OCCURS must be increased. (See Maximum-number-of-entities in the User Maintained Variables section.)
+0311	PREFIX IS TOO LONG FOR ENTITY TABLE:	A namespace definition (xmlns:) at the start of the XML document used a prefix that was too long to fit in the entity table. The length of the entity argument field must be increased. (See Maximum-entity-argument in the User Maintained Variables section.) *This message also identifies the offending prefix.*
+0312	URL IS TOO LONG FOR ENTITY TABLE:	A namespace definition (xmlns:) at the start of the XML document used a URL that was too long to fit in the entity table. The length of the entity value field must be increased (See Maximum-entity-length in the User Maintained Variables section.) *This message also identifies the offending URL.*
+0320	XML SYNTAX ERROR	RCFSTCOB has encountered a syntactical error in the XML document. *This message also points to the offending character position in the XML document.* The PHASE number in the message should always be zero - indicating that the error was found in the first pass of the XML document - if this number is not zero contact Redvers Consulting.
+0330	XML CONTENT IS NOT BASE64 FORMAT	RCFSTCOB has attempted to convert non-base64 data to binary format. This is likely to be caused by a field, defined as binary or packed decimal in the CRD, that is not base64 format in the XML document. (See Binary / Packed Fields in the Picture Clause section). *This message also points to the offending character position in the XML document.* The PHASE number in the message should always be zero - indicating that the error was found in the first pass of the XML document - if this number is not zero contact Redvers Consulting.

Index

X

Z

Redvers Consulting Ltd

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