Former Member

March 14, 2012 | 13 minute read

Guide towards a simple conversion of an XML file to ABAP Internal table, using XML parsing.

Follow

Like

23 27 91,088

Assigned Tags

ABAP Development

abap

xml-abap conversion

Similar Blog Posts

Working with JavaScript Object Notation (JSON) format in ABAP (Serialization - Deserialization)

Guide towards a simple conversion of an XML file to ABAP Internal table, using XML parsing.

Applies to:

SAP ECC 6.0. And further.

This article elaborates the conversion process of an XML file residing in the SAP Application server, into an ABAP internal table.

An effortless approach, unlike the other intricate methods available for conversion, has been presented here. In this particular concept, initially, the XML file is converted into a string. Hence, we use the XML parsing technique to convert that string into an 'x' string and finally into an internal table.

A simple conversion idea, along with the supporting code blocks, has been demonstrated in this document.

Author(s): Aastha Mehrotra

Company: Larsen & Toubro Infotech

Created on: 05January, 2012

Author Bio

Aastha Mehrotra has been engaged with Larsen and Toubro Infotech for two years and three months, her acquaintance with the IT industry being the same.

By Shravan Tiwari Oct 16, 2019

A simple and sample program to get the internal table data into XML format

By Former Member Jan 07, 2013

Quick'n'dirty solution for parsing XLSX files on the go

By Pavlo Astashonok Jul 24, 2020

Related Questions

DataSet text file to internal table

By Former Member Jan 14, 2015

Transformation XML -> abap

By eddhie kurnianto Dec 11, 2006

Accessing XML File

By Dominik Klaffke Feb 09, 2006

The author has worked on various projects in SAP with ABAP. She is also trained in a couple of other ABAP technologies like WebDynpro for ABAP and Workflows. However, the milestone in her career by far, has been her distinguished work in the field of Data Extraction from MDM to SAP R/3. Aastha did a major research on retrieving flat as well as lookup data from MDM into SAP R/3 system, using the MDM ABAP APIs. She also applied her know how on MDM ABAP APIs, in two implementation projects. She has also imparted trainings on Setting up connection between MDM and SAP R/3 as well as Extracting data from MDM to SAP R/3.

Introduction

Various practices and procedures are available for XML-ABAP conversion. However, I discovered that, none of those takes us entirely across the process of converting an XML residing in the Application Server into an internal table in ABAP. Certain other methods, which might actually take us across, are difficult to comprehend and involve a number of complicated steps.

I used this method while working on an object which required me to

- Fetch a request XML file from the Application server.
- Convert it into an internal table in ABAP.
- Extract data from ERP corresponding to the data in the internal table (from the XML file).
- Generate a Response file consisting of both XML file data as well as data from ERP.

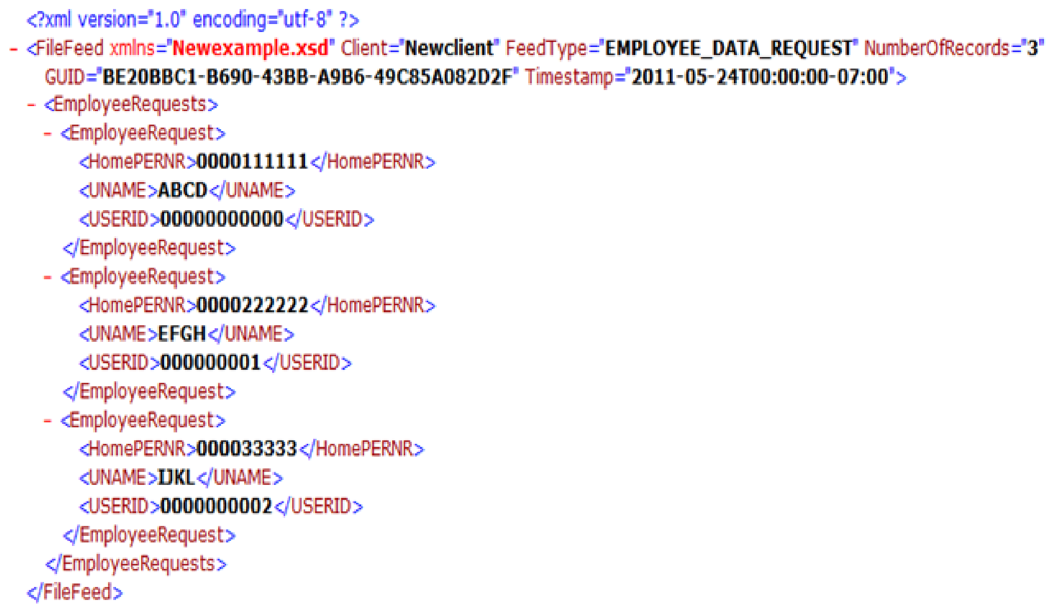
This document shall let you know the entire process involving the evolution of the XML file in the Application server to an Internal Table.

The XML file

Following is an archetype of the XML file which needs to be converted into an internal table in ABAP.

This XML file carries data for three fields- HomePernr, UNAME and USERID. We need to convert the XML file into an internal table comprising of these three fields, and holding the three records from XML.

We would be considering the example of the following file through the course of this document. This would assist us convert any XML containing any number of fields and records into an internal table, in an uncomplicated manner.



```
  <?xml version="1.0" encoding="utf-8" ?>
- <FileFeed xmlns="Newexample.xsd" Client="Newclient" FeedType="EMPLOYEE_DATA_REQUEST" NumberOfRecords="3"
    GUID="BE20BBC1-B690-43BB-A9B6-49C85A082D2F" Timestamp="2011-05-24T00:00:00-07:00">
  - <EmployeeRequests>
    - <EmployeeRequest>
        <HomePERNR>0000111111</HomePERNR>
        <UNAME>ABCD</UNAME>
        <USERID>00000000000</USERID>
      </EmployeeRequest>
    - <EmployeeRequest>
        <HomePERNR>0000222222</HomePERNR>
        <UNAME>EFGH</UNAME>
        <USERID>000000001</USERID>
      </EmployeeRequest>
    - <EmployeeRequest>
        <HomePERNR>000033333</HomePERNR>
        <UNAME>IJKL</UNAME>
        <USERID>0000000002</USERID>
      </EmployeeRequest>
    </EmployeeRequests>
  </FileFeed>
```

Note: FileFeed happens to be the root node for the XML.

EmployeeRequests is the element node containing the separate Employee records within it.

Employeerequest is the element node for each record.

HomePERNR, UNAME and USERID are the value nodes containing the values for the records.

Step wise conversion of the XML to Internal Table

The entire mechanism would be elaborated by this document in several steps, supported with the appropriate code snippets.

The series of execution would be as follows:

- Open the XML in order to read the data.
- Transfer the contents into an internal table.
- Concatenate the lines of the internal table into a string.
- Convert the string into a hexadecimal string – The 'X' string.
- Parse the 'x' string to convert the data into an XML table.
- Read the XML table to transfer the data into the table with the required structure.

Note: The code snippets are actually parts of a single code, in a sequence. Hence, these can be conjoined together to achieve the entire code for XML_ABAP conversion.

We shall advance with the specification of the step wise conversion process. However, we must be aware of the variables to be declared, in advance. This would enhance our understanding of the steps and would provide clarity to the context.

Type declarations for the variables

Following are the declarations for all the variables to be used in the code snippets throughout the document. This would help us avoid all the confusion pertaining to the type of variables used.

```
*      Declaring the file type
```

```
DATA:  g_unixfilename TYPE zpathfile.                 “UNIX file path.

*      Declaring the structure for the XML internal table

TYPES: BEGIN OF ty_xml,

         raw(2000) TYPE c,

       END OF ty_xml.

*      Declaring the XML internal table

DATA:  g_t_xml_tab TYPE TABLE OF ty_xml INITIAL SIZE 0.

*      Declaring the work area for the XML internal table

DATA:  wa_xml_tab TYPE ty_xml.

*      Declaring the string to contain the data for the XML internal
table

DATA:  g_str TYPE string.

*      Declaring the string to contain x string

DATA:  g_xmldata TYPE xstring.

*      Declaring the table to contain the parsed data

DATA:  g_t_xml_info TYPE TABLE OF smum_xmltb INITIAL SIZE 0.

*      Declaring the work area for the internal table containing the
parsed data.

DATA:  g_s_xml_info LIKE LINE OF g_t_xml_info.
```

```
*       Declaring the table to contain the returned messages from the
parsing FM

DATA:   g_t_return TYPE STANDARD TABLE OF bapiret2.

*       Declaring the work area for the return table

DATA:   wa_return LIKE LINE OF g_t_return.

*       Declaring the structure for the table containing fields in the
XML file

TYPES: BEGIN OF struc_people,

         homepernr(8),

         Uname(4) TYPE c,

         Userid(32),

       END OF struc_people.

*       Declaring the internal table containing the fields in the XML
file

DATA:   g_t_employeerequest TYPE TABLE OF struc_people.

*       Declaring the work area for the internal table containing the
fields in the      *      XML file

DATA:   g_s_employeerequest LIKE LINE OF g_t_employeerequest.
```

- ## Open the XML to read data

The first breakthrough would be to open the XML file and read the data using the OPEN DATASET statement.

The addition IN TEXT MODE, to the OPEN DATASET opens the file as a text file. The addition ENCODING defines how the characters are represented in the text file. While writing the data into a text file, the content of a data object is converted to the representation entered after ENCODING, and transferred to the file.

```
* Open the XML file for reading data

OPEN DATASET g_unixfilename FOR INPUT IN TEXT MODE ENCODING DEFAULT.

  IF sy-subrc NE 0.

    MESSAGE ‘Error opening the XML file’ TYPE ‘E’.

  ELSE.
```

- ## Transfer the contents into an internal table of a specific type.

The next furtherance would be to move the contents of the file to an internal table. Following is the piece of code which would help us Read the data from the file into an internal table.

We use the READ DATASET statement to read the data from the file.

```
continued…….

   DO.

* Transfer the contents from the file to the work area of the internal
table
```

```
      READ DATASET g_unixfilename INTO wa_xml_tab.

      IF sy-subrc EQ 0.

        CONDENSE wa_xml_tab.

*       Append the contents of the work area to the internal table

        APPEND wa_xml_tab TO g_t_xml_tab.

      ELSE.

        EXIT.

      ENDIF.

   ENDDO.

ENDIF.

* Close the file after reading the data

  CLOSE DATASET g_unixfilename.
```

- ## Concatenate the lines of the internal table into a string.

Next, we move the lines of the internal table into a string. Hence, we get a string containing the data of the entire XML file.

```
continued…….

*Transfer the contents from the internal table to a string
```

```
IF NOT g_t_xml_tab IS INITIAL.

    CONCATENATE LINES OF g_t_xml_tab INTO g_str SEPARATED BY space.

ENDIF.
```

- ## Converting the normal string into an 'X' string.

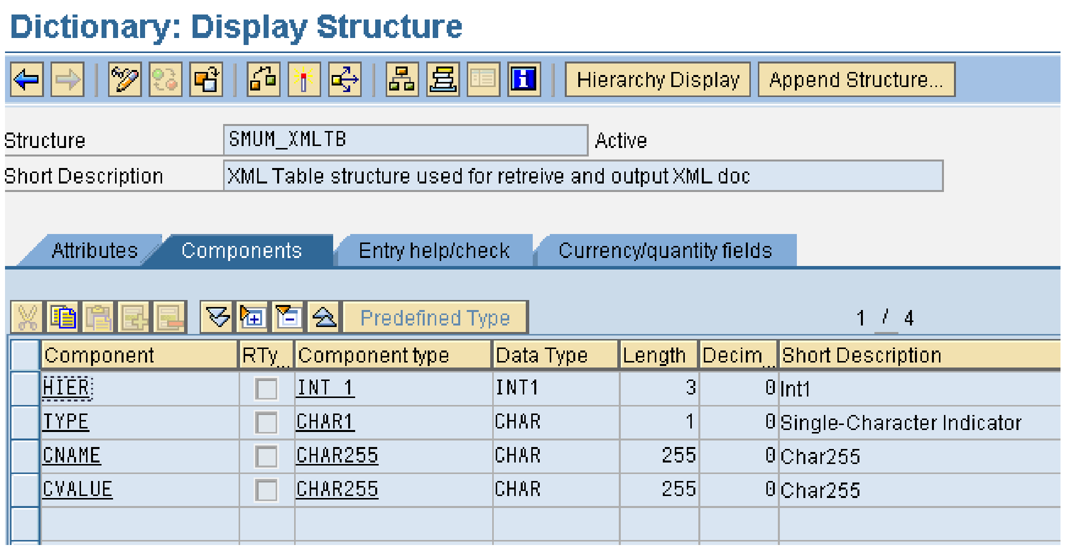
We need to convert the string thus formed into a 'x' string or a hexadecimal string using the function Module `'SCMS_STRING_TO_XSTRING'`. The type xstring allows a hexadecimal display of byte chains instead of the presentation to the base of 64.

```
continued…….

* The function module is used to convert string to xstring

  CALL FUNCTION 'SCMS_STRING_TO_XSTRING'

   EXPORTING

    text   = g_str

   IMPORTING

    buffer = g_xmldata

   EXCEPTIONS

    failed = 1

    OTHERS = 2.

  IF sy-subrc<> 0.
```

```
        MESSAGE ‘Error in the XML file’ TYPE ‘E’.

    ENDIF.
```

• Parsing the x string in order to convert the data into an XML table.

In case we have an XML string that needs to be converted into an object, then the XML string needs to be parsed. The parsing would convert the string into a table. The internal table that would be returned is g_t_xml_info. G_t_xml_info is of type SMUM_XML_TB. The structure for SMUM_XML_TB is in the screen-shot below:



Structure	SMUM_XMLTB Active
Short Description	XML Table structure used for retreive and output XML doc

Component	RTy...	Component type	Data Type	Length	Decim...	Short Description
HIER	☐	INT_1	INT1	3	0	Int1
TYPE	☐	CHAR1	CHAR	1	0	Single-Character Indicator
CNAME	☐	CHAR255	CHAR	255	0	Char255
CVALUE	☐	CHAR255	CHAR	255	0	Char255

The table SMUM_XML_TB has four fields: HIER, TYPE, CNAME and CVALUE. The XML data would be transferred into the four fields.

Moving forward, we would come to know that how exactly is the data allocated to the fields of this table?

```
continued……..

* This function module is used to parse the XML and get the

* data in the form of a table

  CALL FUNCTION ‘SMUM_XML_PARSE’

    EXPORTING

     xml_input = g_xmldata

    TABLES

     xml_table = g_t_xml_info

     return    = g_t_return

    EXCEPTIONS

      OTHERS    = 0.

*"If XML parsing is not successful, return table will contain error
messages

  READ TABLE g_t_return INTO wa_return WITH KEY type = ‘E’.

  IF sy-subrc EQ 0.

    MESSAGE ‘Error converting the input XML file’ TYPE ‘E’.

  ELSE.

    REFRESH g_t_return.

  ENDIF.
```

- ## Read the XML table to transfer the data into the required table.

We are acquainted with the fields of table g_t_xml_info HIER, TYPE, CNAME and CVALUE.

We now, need to move the data from this table into the required, final table. For instance in the scenario we have been considering, the final table would consist of the three fields HomePernr, UNAME and USERID. The transfer of data from the table g_t_xml_info to our final table would be done in accordance with the values held by the four fields of the g_t_xml_info table.

While parsing, the values are assigned to the table g_t_xml_info. The values are moved depending upon the node being converted to the table.

Let me elaborate about the values held by the table g_t_xml_info.

1. Values held by the field HIER in XML table

The field ‘Hier’ would hold the value ‘1’ for the element in the root node. Since, Root node is the first level of the XML hierarchy.

‘Hier’ would hold the value ‘2’ for the fields in the Element node, which is at the second level of hierarchy. In our example EmployeeRequests is at the second level of hierarchy.

It would hold ‘3’ for Element nodes at the third level of hierarchy in the XML file that is EmployeeRequest here.

The value would be ‘4’ for all the Value nodes. Here the value nodes would be HomePernr, UNAME and USERID.

Hence, we conclude that the value of ‘Hier’ depends upon the level of hierarchy that particular node holds in the XML file.

2. Values held by the field TYPE in XML table

The value that the field 'Type' holds for all the elements in the Header of the XML file would be 'A'. It would be 'V' for the value nodes. For all the other nodes, it would hold a blank.

3. Values held by the field CNAME in XML table

Cname contains the names of the nodes. Hence, considering our example here, if the Hier is '2' the Type would be blank the Cname would be 'EmployeeRequests' and the Cvalue would be blank since EmployeeRequests holds no value.

4. Values held by the field CVALUE in XML table

Cvalue contains the values held by the elements of the various nodes of an XML file. Taking an example of our case here-

For HIER '4' the Type would be 'V', Cname can be 'HomePernr', 'UNAME' or 'USERID' and the Cvalue would be the values held by these records.

Here is a screen-shot of the values contained in the table g_t_xml_info, considering our example.

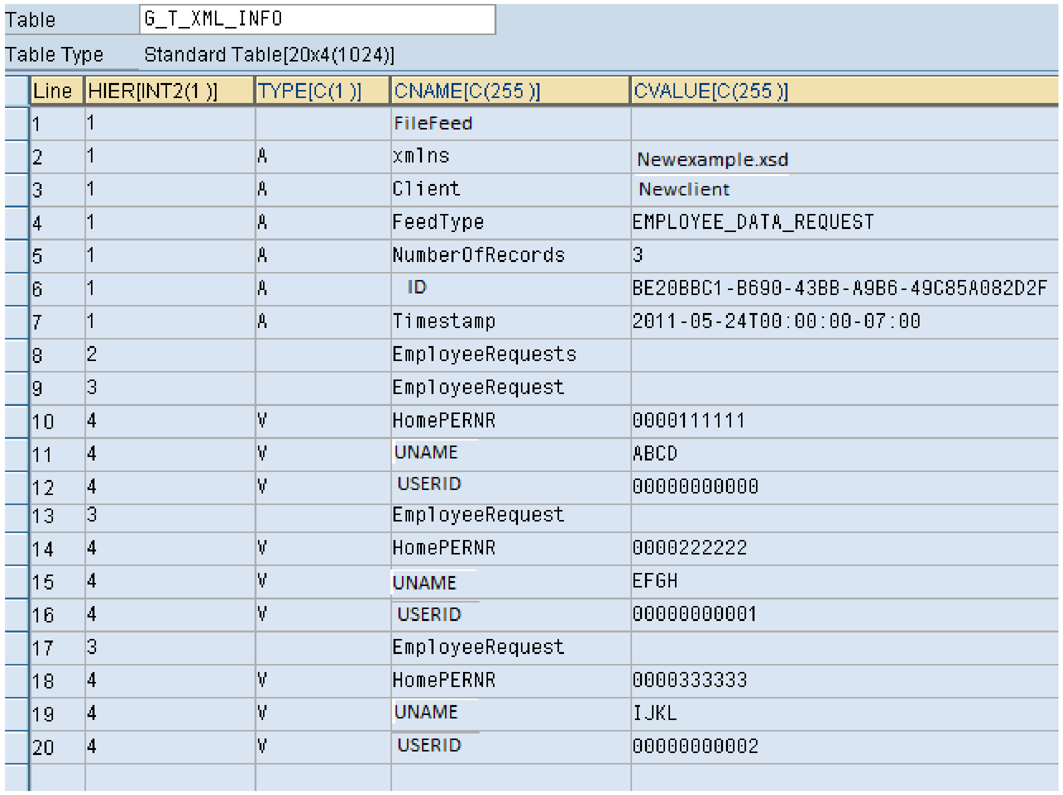


Table G_T_XML_INFO

Table Type Standard Table[20x4(1024)]

Line	HIER[INT2(1)]	TYPE[C(1)]	CNAME[C(255)]	CVALUE[C(255)]
1	1		FileFeed	
2	1	A	xmlns	Newexample.xsd
3	1	A	Client	Newclient
4	1	A	FeedType	EMPLOYEE_DATA_REQUEST
5	1	A	NumberOfRecords	3
6	1	A	ID	BE20BBC1-B690-43BB-A9B6-49C85A082D2F
7	1	A	Timestamp	2011-05-24T00:00:00-07:00
8	2		EmployeeRequests	
9	3		EmployeeRequest	
10	4	V	HomePERNR	0000111111
11	4	V	UNAME	ABCD
12	4	V	USERID	00000000000
13	3		EmployeeRequest	
14	4	V	HomePERNR	0000222222
15	4	V	UNAME	EFGH
16	4	V	USERID	00000000001
17	3		EmployeeRequest	
18	4	V	HomePERNR	0000333333
19	4	V	UNAME	IJKL
20	4	V	USERID	00000000002

Now that we are acquainted with the pattern of values contained in g_t_xml_info, we can move these values to the required structure.

We require only the values contained in the value nodes and we have to transfer these to their respective fields i.e. HomePernr, UNAME and USERID. Since, we need the values only for the value nodes we can directly move the values in the final table for a value of HIER equal to '3'.

Note: The value of HIER for which we need the data, can be manipulated in accordance with the position of the value nodes in the hierarchy of XML.

We use the following code snippet to move the data:

```
continued…….

* Moving the data from the g_t_xml_info table to the required table
```

```
IF NOT g_t_xml_info IS INITIAL.

  LOOP AT g_t_xml_info INTO    g_s_xml_info WHERE hier EQ 3.

      tabix = sy-tabix + 1.

      READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

      g_s_employeerequest-homepernr = g_s_xml_info-cvalue.

      tabix = tabix + 1.

      READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

      g_s_employeerequest-Uname = g_s_xml_info-cvalue.

      tabix = tabix + 1.

      READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

      g_s_employeerequest-Userid = g_s_xml_info-cvalue.

      APPEND g_s_employeerequest TO g_t_employeerequest.

  ENDLOOP.

ENDIF.
```

Here, the value ‘3’ for the field ‘Hier’ marks the beginning of a new record. Hence, we move the values to the g_t_employeerequest table whenever the value for ‘Hier’ is ‘3’.

This way we get the required values in the internal table g_t_employeerequest.

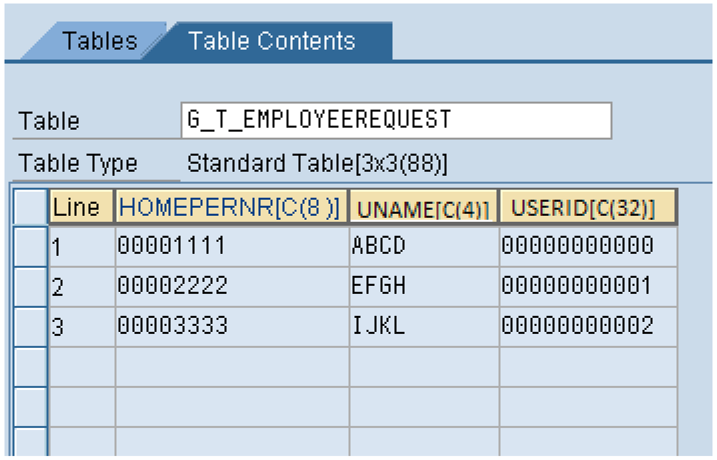


Table G_T_EMPLOYEEREQUEST
Table Type Standard Table[3x3(88)]

Line	HOMEPERNR[C(8)]	UNAME[C(4)]	USERID[C(32)]
1	00001111	ABCD	00000000000
2	00002222	EFGH	00000000001
3	00003333	IJKL	00000000002

Thus, we get the values in our desired structure. We populated the final table in accordance with the location

This marks the end of our journey from the XML residing in the application server to the internal table in SAP ABAP.

Advantages/Disadvantages of using the method SMUM_XML_PARSE over XSLT (call transformation id)

- As apparent from the description of the entire process, the Function Module SMUM_XML_PARSE is beyond question an undemanding approach towards the XML-ABAP conversion. SMUM_XML_PARSE is an uncomplicated, unreleased, effortless and undocumented version of the powerful, released and documented iXML.

- This function module can be used for complex XML structures by deciding upon suitable ways to segregate the data from the XML table in to the required internal table. This is entirely in our hands since; we are the ones to decide upon the structure of the final internal table required and to decide upon the HIER values of the XML table for which we need the data in the required table..

- To turn the XML into an internal table or any other series of ABAP objects we can make use of ABAP's CALL TRANSFORMATION keyword. Though, this unlike SMUM_XML_PARSE is very well documented in the help documentations and makes use of a XSLT template to transform XML into ABAP objects and vice versa. However, while making use of CALL TRANSFORMATION we need to create the XML schema or the XSLT program in the transaction SE80 which has to be in a definite format, failing which there can be a number of anomalies. Besides, this is a time consuming process. This adds to the complexity quotient of the CALL TRANSFORMATION method.

- Quite often, a peculiar problem that occurs while reading the xml file using the CALL TRANSFORMATION method is- 'format not compatible with the internal table'. Hence, in order to get rid of this particular issue we further need to apply another transformation to convert the data from the internal table into the xml file. Then only, do we get the format of XML which might be utilized for conversion and is compatible with the given internal table.

Related Content

SAP Community Network Wiki – Code Gallery – XML TO ABAP INTERNAL TABL CONVERSION

Parsing an XML and Inserting to ABAP

XML XSLT with ABAP

Alert Moderator

23 Comments

You must be Logged on to comment or reply to a post.

Former Member
March 14, 2012 at 9:26 am

This is a very step by step document which is easy to understand and very helpful.

Like 0 | Share

Former Member
March 27, 2012 at 9:04 am

Excellent step by step document .

Like 0 | Share

Gaurab Banerji
March 27, 2012 at 9:32 am

the function module does all the work… anyways… nice post

Like 0 | Share

Former Member

April 9, 2012 at 8:39 am

Comprehensive and elaborate! Good for understanding the concept! Thanks!

Like 0 | Share

Former Member

April 19, 2012 at 12:35 pm

Great work!

Like 0 | Share

Vinoth Kumar

April 20, 2012 at 3:35 am

Nice to know of other way apart from XLST conversion

Thanks for the details

Vinoth

Like 0 | Share

Former Member
April 23, 2012 at 5:26 am

This document is a good ready reference and an excellent exlearning..Thanks ☐

Like 0 | Share

Former Member
April 23, 2012 at 9:16 am

Hello Aastha,

A couple of points:

- Why are you opening the file in TEXT mode & then converting it into XSTRING? You can open the file in BINARY MODE directly.
- With all due respect to your effort, did you search SCN thoroughly? I somehow don't advocate use of unreleased FMs unless there are no other available options. You can refer to the blog by Uwe - Visualizing Any Kind of XML Data Using Class CL_XML_DOCUMENT.

Let me know your thoughts.

BR,

Suhas

Like 0 | Share

Former Member | Blog Post Author
April 23, 2012 at 11:31 am

Hi Suhas,

I am opening the file in text and ghence converting it to Xstring because I was not sure if I open the file in Binary then I would not require a conversion. Since, xstring is like hexadecimal string and presume that Binary and xstring would be different. However, in case you have any inputs about that I shall be more then happy to widen my horizon □

Regarding your second point, trust me when I came across this requirement I tried searching everywhere and could not find out an article or piece of code that could take me through my desired phenomenon. I needed to pick up an XML residing at the Application server into an ABAP internal table. This was the easiest way i could adopt....

Thanks a lot about your inputs and I found Uwe's post convincing too.

Regards,

Aastha

Like 0 | Share

Former Member
July 9, 2012 at 6:09 pm

Hi Suhas,

I've been trying to comment at your blog (http://wiki.sdn.sap.com/wiki/display/ABAP/Upload+XML+file+to+internal+table) but something seems to be wrong with SDN...Your method of approach in your blog works very nicely...but I have couple of questions that I hope you can answers

1. Is there an approach to handle app server files?

2. How can we handle multiple worksheets using your approach?

If there is any sort of info you could provide...would be a great help...

Thanks & Regards,

Divaker

Like 0 | Share

Former Member

July 10, 2012 at 6:04 am

Hi Divaker,

Sorry for the late reply!

1. Is there an approach to handle app server files? - If you check the method *IMPORT_FROM_FILE( )*, you can see the method *CL_GUI_FRONTEND_SERVICES=>GUI_UPLOAD( )* is used to upload the XML file as 'BIN'(ary). Imo you can open the AS file in *BINARY MODE* & then use the method *CREATE_WITH_TABLE( )* or *PARSE_TABLE( )* to proceed as usual.

2. *How can we handle multiple worksheets using your approach?* - No idea about that. You gotto try it out on your own

Cheers,

Suhas

Like 0 | Share

Former Member

July 10, 2012 at 11:03 am

Hi Suhas,

Thanks for your reply…But there is an issue that I'm encountering when I try your approach, there is a value which contains an ampersand '&' and the program flow is being stopped when it reads this particular value hence interupting the data read!

Did you face anything like this, or do you have any idea regarding this?

Please do let me know…

Thanks & Regards,

Divaker

Like 0 | Share

Former Member

July 9, 2012 at 11:41 am

Hi Aastha,

I have an excel sheet with multiple sheets saving as an XML file, would the method suggested by you here be possible to read this multiple sheet data into an internal table again?

Please do let me know...

Thanks,

Divaker

Like 0 | Share

Former Member | Blog Post Author
July 9, 2012 at 12:44 pm

Hi Divaker,

In that case you can first download each file into separate internal tables and finally combine those into a final table that you require.

Please let me know in case you require any further information on this one.

Regards,

Aastha

Like 0 | Share

Former Member
July 9, 2012 at 6:03 pm

Hi Aastha,

Thank You for your reply, but my issue is that does this method work for multiple worksheets too? How can we segregate data based on each worksheet? That is where I'm getting caught up. Also I tried your method and the function call to 'SMUM_XML_PARSE' is failing.

If there is anything you think could help me, please do let me know.

Thanks & Regards,

Divaker

Like 0 | Share

Dipak Khatavkar

May 28, 2013 at 5:31 am

Hi Aastha,

Very nice document. Its very easy to understand.

Thanks,

Dipak.

Like 0 | Share

Former Member

April 8, 2015 at 9:05 am

Hi Aastha,

Thanks a lot for this tutorial.

I have a problem with the "zpathfile" though.

As it's not a predefined type in abap, I replaced it with the type "localfile" but I had the message error "Error opening the xml file".

My question is: How can I define manually the "zpathfile"?

Thank you in advance for your answer.

Like 1 | Share

Naveen Kumar

August 3, 2015 at 12:08 pm

Hi Aastha,

Very useful document and thanks I got the solution

Like 0 | Share

Steven Ludmon

November 24, 2016 at 4:10 am

Hi all

Firstly, this is a very useful posting, thanks to the author Former Member.

There is similar function module in the SOAP_UTIL package that I have used recently:

SRTUTIL_CONVERT_XML_TO_TABLE

Regards, Steve

Like 1 | Share

Pawel Kurpinski

October 9, 2019 at 2:24 pm

Thanks a lot for this tutorial Aastha!

I have the same problem with the "zpathfile" 🙁

I also replaced it with the type "localfile" but I had the message error "Error opening the xml file".

My question is: How can I define manually the "zpathfile"?

Thank you in advance for your answer!

Regards

Paweł

Aastha,

Like 0 | Share

Ipsita Kundu

January 28, 2022 at 12:08 pm

Hello Aastha,

I have used your code as mentioned below :

I have uploaded XML(This file is generated from a composite provider) file in AL11 and want to convert it to tabular form.

T-Code: XSLT_TOOL.

when I am giving the file path ,it is showing "Invalid XML Source".(please check the screenshot)

Please help.

Code which I written:

```
*     Declaring the file type

DATA:  g_unixfilename TYPE zpathfile.              “UNIX file path.

*     Declaring the structure for the XML internal table
```

```
TYPES: BEGIN OF ty_xml,

raw(2000) TYPE c,

END OF ty_xml.

*      Declaring the XML internal table

DATA:  g_t_xml_tab TYPE TABLE OF ty_xml INITIAL SIZE 0.

*      Declaring the work area for the XML internal table

DATA:  wa_xml_tab TYPE ty_xml.

*      Declaring the string to contain the data for the XML internal table

DATA:  g_str TYPE string.

*      Declaring the string to contain x string

DATA:  g_xmldata TYPE xstring.

*      Declaring the table to contain the parsed data

DATA:  g_t_xml_info TYPE TABLE OF smum_xmltb INITIAL SIZE 0.

*      Declaring the work area for the internal table containing the parsed data.

DATA:  g_s_xml_info LIKE LINE OF g_t_xml_info.

*      Declaring the table to contain the returned messages from the parsing FM

DATA:  g_t_return TYPE STANDARD TABLE OF bapiret2.

*      Declaring the work area for the return table
```

DATA: wa_return LIKE LINE OF g_t_return.

*   Declaring the structure for the table containing fields in the XML file

TYPES: BEGIN OF struc_people,

homepernr(8),

Uname(4) TYPE c,

Userid(32),

END OF struc_people.

*   Declaring the internal table containing the fields in the XML file

DATA: g_t_employeerequest TYPE TABLE OF struc_people.

*   Declaring the work area for the internal table containing the fields in the   *   XML file

DATA: g_s_employeerequest LIKE LINE OF g_t_employeerequest.

* Open the XML file for reading data

OPEN DATASET g_unixfilename FOR INPUT IN TEXT MODE ENCODING DEFAULT.

IF sy-subrc NE 0.

MESSAGE ‘Error opening the XML file’ TYPE ‘E’.

ELSE.

DO.

* Transfer the contents from the file to the work area of the internal table

```
READ DATASET g_unixfilename INTO wa_xml_tab.

IF sy-subrc EQ 0.

CONDENSE wa_xml_tab.

*       Append the contents of the work area to the internal table

APPEND wa_xml_tab TO g_t_xml_tab.

ELSE.

EXIT.

ENDIF.

ENDDO.

ENDIF.

* Close the file after reading the data

CLOSE DATASET g_unixfilename.

*Transfer the contents from the internal table to a string

IF NOT g_t_xml_tab IS INITIAL.

CONCATENATE LINES OF g_t_xml_tab INTO g_str SEPARATED BY space.

ENDIF.

* The function module is used to convert string to xstring

CALL FUNCTION ‘SCMS_STRING_TO_XSTRING’
```

```
EXPORTING

text   = g_str

IMPORTING

buffer = g_xmldata

EXCEPTIONS

failed = 1

OTHERS = 2.

IF sy-subrc NE 0.

MESSAGE ‘Error in the XML file’ TYPE ‘E’.

ENDIF.

* This function module is used to parse the XML and get the

* data in the form of a table

CALL FUNCTION ‘SMUM_XML_PARSE’

EXPORTING

xml_input = g_xmldata

TABLES

xml_table = g_t_xml_info

return    = g_t_return
```

```
EXCEPTIONS

OTHERS    = 0.

*"If XML parsing is not successful, return table will contain error messages

READ TABLE g_t_return INTO wa_return WITH KEY type = 'E'.

IF sy-subrc EQ 0.

MESSAGE 'Error converting the input XML file' TYPE 'E'.

ELSE.

REFRESH g_t_return.

ENDIF.

* Moving the data from the g_t_xml_info table to the required table

IF NOT g_t_xml_info IS INITIAL.

LOOP AT g_t_xml_info INTO    g_s_xml_info WHERE hier EQ 3.

tabix = sy-tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-homepernr = g_s_xml_info-cvalue.

tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Uname = g_s_xml_info-cvalue.
```

tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Userid = g_s_xml_info-cvalue.

APPEND g_s_employeerequest TO g_t_employeerequest.

ENDLOOP.

ENDIF.

Like 0 | Share

Ipsita Kundu
January 28, 2022 at 2:54 pm

Hello Aastha,

It is not working for me , I have pasted the code which I have used.

It is also not working if I hardcode the file path.

Please Help.

Code:

*   Declaring the file type

```
DATA:  g_unixfilename TYPE zpathfile.              "UNIX file path.

*      Declaring the structure for the XML internal table

TYPES: BEGIN OF ty_xml,

raw(2000) TYPE c,

END OF ty_xml.

*      Declaring the XML internal table

DATA:  g_t_xml_tab TYPE TABLE OF ty_xml INITIAL SIZE 0.

*      Declaring the work area for the XML internal table

DATA:  wa_xml_tab TYPE ty_xml.

*      Declaring the string to contain the data for the XML internal table

DATA:  g_str TYPE string.

*      Declaring the string to contain x string

DATA:  g_xmldata TYPE xstring.

*      Declaring the table to contain the parsed data

DATA:  g_t_xml_info TYPE TABLE OF smum_xmltb INITIAL SIZE 0.

*      Declaring the work area for the internal table containing the parsed data.

DATA:  g_s_xml_info LIKE LINE OF g_t_xml_info.

*      Declaring the table to contain the returned messages from the parsing FM
```

```
DATA:  g_t_return TYPE STANDARD TABLE OF bapiret2.

*      Declaring the work area for the return table

DATA:  wa_return LIKE LINE OF g_t_return.

*      Declaring the structure for the table containing fields in the XML file

TYPES: BEGIN OF struc_people,

homepernr(8),

Uname(4) TYPE c,

Userid(32),

END OF struc_people.

*      Declaring the internal table containing the fields in the XML file

DATA:  g_t_employeerequest TYPE TABLE OF struc_people.

*      Declaring the work area for the internal table containing the fields in the      *      XML file

DATA:  g_s_employeerequest LIKE LINE OF g_t_employeerequest.

* Open the XML file for reading data

OPEN DATASET g_unixfilename FOR INPUT IN TEXT MODE ENCODING DEFAULT.

IF sy-subrc NE 0.

MESSAGE ‘Error opening the XML file’ TYPE ‘E’.

ELSE.
DO.
```

```
g_unixfilename =  /Filepath/.txt



* Transfer the contents from the file to the work area of the internal table

READ DATASET g_unixfilename INTO wa_xml_tab.

IF sy-subrc EQ 0.

CONDENSE wa_xml_tab.

*       Append the contents of the work area to the internal table

APPEND wa_xml_tab TO g_t_xml_tab.

ELSE.

EXIT.

ENDIF.

ENDDO.

ENDIF.

* Close the file after reading the data

CLOSE DATASET g_unixfilename.

*Transfer the contents from the internal table to a string

IF NOT g_t_xml_tab IS INITIAL.

CONCATENATE LINES OF g_t_xml_tab INTO g_str SEPARATED BY space.
```

```
ENDIF.

* The function module is used to convert string to xstring

CALL FUNCTION ‘SCMS_STRING_TO_XSTRING’

EXPORTING

text   = g_str

IMPORTING

buffer = g_xmldata

EXCEPTIONS

failed = 1

OTHERS = 2.

IF sy-subrc NE 0.

MESSAGE ‘Error in the XML file’ TYPE ‘E’.

ENDIF.

* This function module is used to parse the XML and get the

* data in the form of a table

CALL FUNCTION ‘SMUM_XML_PARSE’

EXPORTING

xml_input = g_xmldata
```

```
TABLES

xml_table = g_t_xml_info

return    = g_t_return

EXCEPTIONS

OTHERS    = 0.

*"If XML parsing is not successful, return table will contain error messages

READ TABLE g_t_return INTO wa_return WITH KEY type = 'E'.

IF sy-subrc EQ 0.

MESSAGE 'Error converting the input XML file' TYPE 'E'.

ELSE.

REFRESH g_t_return.

ENDIF.

* Moving the data from the g_t_xml_info table to the required table

IF NOT g_t_xml_info IS INITIAL.

LOOP AT g_t_xml_info INTO    g_s_xml_info WHERE hier EQ 3.

tabix = sy-tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-homepernr = g_s_xml_info-cvalue.
```

```
tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Uname = g_s_xml_info-cvalue.

tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Userid = g_s_xml_info-cvalue.

APPEND g_s_employeerequest TO g_t_employeerequest.

ENDLOOP.

ENDIF.
```

Like 0 | Share

Ipsita Kundu
January 28, 2022 at 2:57 pm

Hello Aastha,

The code is not working for me .I have copied your code and pasted but not working.

If hardcode the , that is also not working.

Please help.

Code:

```
*     Declaring the file type

DATA:  g_unixfilename TYPE zpathfile.               "UNIX file path.

*     Declaring the structure for the XML internal table

TYPES: BEGIN OF ty_xml,

raw(2000) TYPE c,

END OF ty_xml.

*     Declaring the XML internal table

DATA:  g_t_xml_tab TYPE TABLE OF ty_xml INITIAL SIZE 0.

*     Declaring the work area for the XML internal table

DATA:  wa_xml_tab TYPE ty_xml.

*     Declaring the string to contain the data for the XML internal table

DATA:  g_str TYPE string.

*     Declaring the string to contain x string

DATA:  g_xmldata TYPE xstring.

*     Declaring the table to contain the parsed data
```

DATA: g_t_xml_info TYPE TABLE OF smum_xmltb INITIAL SIZE 0.

*   Declaring the work area for the internal table containing the parsed data.

DATA: g_s_xml_info LIKE LINE OF g_t_xml_info.

*   Declaring the table to contain the returned messages from the parsing FM

DATA: g_t_return TYPE STANDARD TABLE OF bapiret2.

*   Declaring the work area for the return table

DATA: wa_return LIKE LINE OF g_t_return.

*   Declaring the structure for the table containing fields in the XML file

TYPES: BEGIN OF struc_people,

homepernr(8),

Uname(4) TYPE c,

Userid(32),

END OF struc_people.

*   Declaring the internal table containing the fields in the XML file

DATA: g_t_employeerequest TYPE TABLE OF struc_people.

*   Declaring the work area for the internal table containing the fields in the   *   XML file

DATA: g_s_employeerequest LIKE LINE OF g_t_employeerequest.

* Open the XML file for reading data

```
OPEN DATASET g_unixfilename FOR INPUT IN TEXT MODE ENCODING DEFAULT.

IF sy-subrc NE 0.

MESSAGE ‘Error opening the XML file’ TYPE ‘E’.

ELSE.
DO.
g_unixfilename =  /filepath/file.txt



* Transfer the contents from the file to the work area of the internal table

READ DATASET g_unixfilename INTO wa_xml_tab.

IF sy-subrc EQ 0.

CONDENSE wa_xml_tab.

*       Append the contents of the work area to the internal table

APPEND wa_xml_tab TO g_t_xml_tab.

ELSE.

EXIT.

ENDIF.

ENDDO.

ENDIF.

* Close the file after reading the data
```

```
CLOSE DATASET g_unixfilename.

*Transfer the contents from the internal table to a string

IF NOT g_t_xml_tab IS INITIAL.

CONCATENATE LINES OF g_t_xml_tab INTO g_str SEPARATED BY space.

ENDIF.

* The function module is used to convert string to xstring

CALL FUNCTION ‘SCMS_STRING_TO_XSTRING’

EXPORTING

text   = g_str

IMPORTING

buffer = g_xmldata

EXCEPTIONS

failed = 1

OTHERS = 2.

IF sy-subrc NE 0.

MESSAGE ‘Error in the XML file’ TYPE ‘E’.

ENDIF.

* This function module is used to parse the XML and get the
* data in the form of a table

CALL FUNCTION 'SMUM_XML_PARSE'

EXPORTING

xml_input = g_xmldata

TABLES

xml_table = g_t_xml_info

return    = g_t_return

EXCEPTIONS

OTHERS    = 0.

*"If XML parsing is not successful, return table will contain error messages

READ TABLE g_t_return INTO wa_return WITH KEY type = 'E'.

IF sy-subrc EQ 0.

MESSAGE 'Error converting the input XML file' TYPE 'E'.

ELSE.

REFRESH g_t_return.

ENDIF.

* Moving the data from the g_t_xml_info table to the required table

IF NOT g_t_xml_info IS INITIAL.
```

```
LOOP AT g_t_xml_info INTO    g_s_xml_info WHERE hier EQ 3.

tabix = sy-tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-homepernr = g_s_xml_info-cvalue.

tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Uname = g_s_xml_info-cvalue.

tabix = tabix + 1.

READ TABLE g_t_xml_info INTO g_s_xml_info INDEX tabix.

g_s_employeerequest-Userid = g_s_xml_info-cvalue.

APPEND g_s_employeerequest TO g_t_employeerequest.

ENDLOOP.

ENDIF.
```

Like 0 | Share