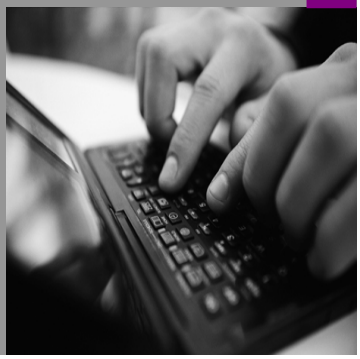


How-to Guide
SAP NetWeaver '04

How To... Convert an IDoc-XML structure to a flat file and vice versa in XI 3.0

Version 1.10 – June 2006

Applicable Releases:
SAP NetWeaver '04

1 Scenario

You want to exchange XI messages with a Third Party System that requires IDoc data in flat file format.

2 Introduction

Numerous Third Party applications can only deal with IDoc data in flat file format. Since IDocs are generally transported inside XI in the so called IDoc-XML format, conversions between these formats might be necessary.
This guide describes on the one hand how to create a flat file out of an IDoc-XML by means of an ABAP mapping program and the J2EE File Adapter. It provides you also with a procedure how to create an XI message in IDoc-XML format starting from a flat file representation of an IDoc.

3 The Step By Step Solution

This chapter is divided into two parts. In section 3.1 the conversion of an IDoc-XML to a flat file in XI 3.0 by means of an ABAP mapping is presented. The relevant coding is listed at the end of section 3.1. In section 3.2 a method to convert a flat file representation to an XI message of IDoc-XML format is described.

3.1 Convert an IDoc-XML representation to a flat file

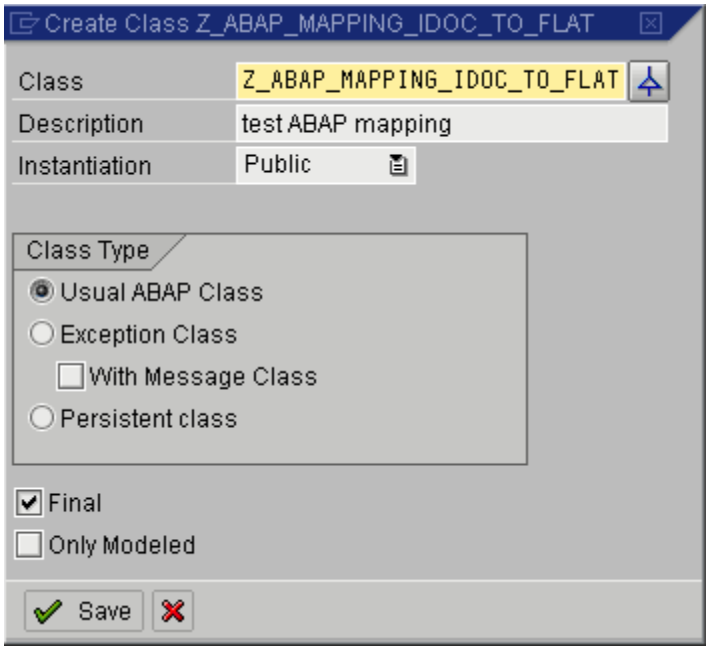
The conversion of an XI message of Doc-XML format to a flat file structure is based on ABAP mapping and makes use of the J2EE File Adapter. Therefore, please make yourself familiar with the usage of ABAP mapping in XI 3.0 before starting the implementation (e.g. see also the How-To guide How to Use ABAP-Mapping in XI 3.0 or the online documentation).
Please note the following points:

- The IDoc segment structure written to the file is based on the structure of the incoming IDoc. The IDocs are written out "as is". The segment definitions will not be converted to a release different from the one of the sender.
- Always the full length of a segment is written out. (I.e. all the fields of one segment result in one line of your file)
- Only the content of one IDoc per XI message can be processed.
- To keep the original IDoc number (DOCNUM) in the flat file, please uncomment the relevant lines as marked in the coding.
- No error handling is implemented.

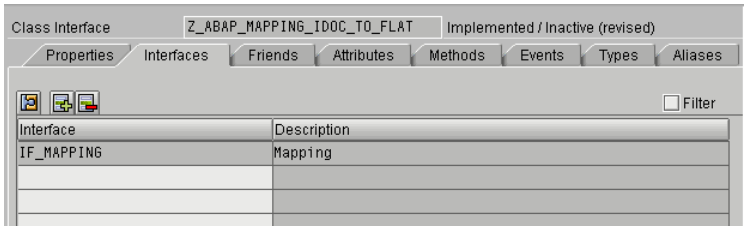
1. Enable ABAP mapping

See HowTo Guide on ABAP mapping (How to Use ABAP-Mapping in XI 3.0).

2. Create a new ABAP-OO class e.g. by using the ABAP Development Workbench (se80). Please note that the method displayed in this How-To Guide is generic. You only have to create the ABAP mapping once per XI system and it will work for any IDoc Type.



3. Your class has to implement the interface `IF_MAPPING` of package `SAI_MAPPING` shipped by SAP.

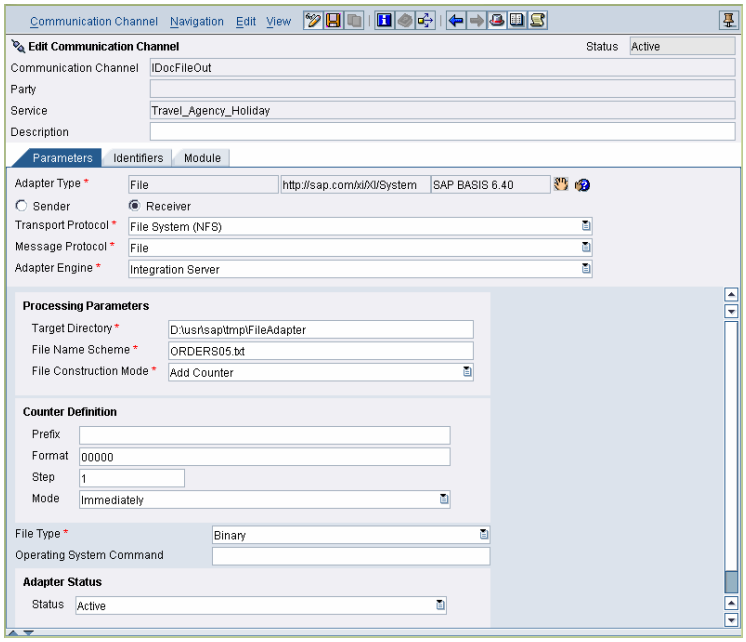


4. Implement the EXECUTE method. Please find the coding for the conversion at the end of this chapter.

The coding can be split up into three parts: First the IDoc-XML document is parsed to get the IDoc header data and the connection parameters to the application system are read in (data maintained by transaction idx1). The selection is based on parameters MANDT and SNDPOR. In a second step the content of IDoc-XML is converted to internal tables of type EDI_DC40 and EDI_DD40. These tables are, in a third step, filled in the export parameter RESULT of type XSTRING using a modification of the shipped function SOTR_SERV_TABLE_TO_STRING.

5. Create an Interface Mapping and your configurations in the Integration Directory as described in the ABAP mapping How-To-Guide.

6. To store the flat file on the file system, use a Communication Channel of type File Receiver and set the parameter Message Protocol to 'File'. Setting the parameter File Type to 'Binary' might help to avoid problems related to codepage conversions.



Implementation of the ABAP mapping

In the following you will find the ABAP Mapping to convert an IDoc-XML representation of your XI message to a flat data structure. Each IDoc segment is represented by a new line. Each line is terminated by a carriage return and line feed.
The mapping consists of an implementation of the ~EXECUTE method of your ABAP-OO class created in step 2. This method calls the function module ZSOTR_SERV_TABLE_TO_STRING which has also to be developed.

Implementation of the method IF_MAPPING~EXECUTE of the class created in step 2.

```
method if_mapping~execute .

  data: t_edidc       type table of edi_dc40,
        ls_edidc      type edi_dc40,
        ls_edidc_h    type edi_dc40,
        t_edidd       type table of edi_dd40,
        ls_edidd_h    type edi_dd40,
        ls_idx_xmb    type idx_xmb,
        t_result      type string,
        t_resultc     type string,
        t_resultd     type string.
  data: ls_idx1       type idxporsm59.
  data: t_segtyp      type table of edilsegtyp,
        el_segtyp     type edilsegtyp,
        el_released   type segdefrel,
        el_error_text type string.

* ======================
* 0. parse input document
* ======================
* initialize iXML
  type-pools: ixml.
  class cl_ixml definition load.

* create main factory
  data: ixmlfactory type ref to if_ixml.
  ixmlfactory = cl_ixml=>create( ).

* create stream factory
  data: streamfactory type ref to if_ixml_stream_factory.
  streamfactory = ixmlfactory->create_stream_factory( ).

* create input stream
  data: istream type ref to if_ixml_istream.
  istream = streamfactory->create_istream_xstring( source ).

* initialize input document
  data: idocument type ref to if_ixml_document.
  idocument = ixmlfactory->create_document( ).

* parse input document
  data: iparser type ref to if_ixml_parser.
  iparser = ixmlfactory->create_parser( stream_factory = streamfactory
                                        istream        = istream
                                        document       = idocument ).
  iparser->parse( ).

* ================================================
* 1. get IDoc header data and connection parameters
* ================================================
  data: el_message_id type sxmsguid.
  el_message_id = param->get( if_mapping_param=>message_id ).

  data: el_element type ref to if_ixml_element.
  el_element = idocument->find_from_name( 'TABNAM' ).
  ls_edidc-tabnam = el_element->get_value( ).
  el_element = idocument->find_from_name( 'MANDT' ).
  ls_edidc-mandt =  el_element->get_value( ).
  el_element = idocument->find_from_name( 'DOCNUM' ).
  ls_edidc-docnum = el_element->get_value( ).
  el_element = idocument->find_from_name( 'STATUS' ).
  ls_edidc-status = el_element->get_value( ).
  el_element = idocument->find_from_name( 'OUTMOD' ).
```

```
  ls_edidc-outmod = el_element->get_value( ).
  el_element = idocument->find_from_name( 'TEST' ).
  if not el_element is initial.
    ls_edidc-test = el_element->get_value( ).
  endif.
  el_element = idocument->find_from_name( 'IDOCTYP' ).
  ls_edidc-idoctyp = el_element->get_value( ).
  el_element = idocument->find_from_name( 'CIMTYP' ).
  if not el_element is initial.
    ls_edidc-cimtyp = el_element->get_value( ).
  endif.
  el_element = idocument->find_from_name( 'MESTYP' ).
  ls_edidc-mestyp = el_element->get_value( ).
  el_element = idocument->find_from_name( 'STDVRS' ).
  if not el_element is initial.
    ls_edidc-stdvrs = el_element->get_value( ).
  endif.
  el_element = idocument->find_from_name( 'STD' ).
  if not el_element is initial.
    ls_edidc-std = el_element->get_value( ).
  endif.
  el_element = idocument->find_from_name( 'STDMES' ).
  ls_edidc-stdmes = el_element->get_value( ).
  el_element = idocument->find_from_name( 'SNDPOR' ).
  ls_edidc-sndpor = el_element->get_value( ).
  el_element = idocument->find_from_name( 'SNDPRT' ).
  ls_edidc-sndprt = el_element->get_value( ).
  el_element = idocument->find_from_name( 'SNDPRN' ).
  ls_edidc-sndprn = el_element->get_value( ).
  el_element = idocument->find_from_name( 'RCVPOR' ).
  ls_edidc-rcvpor = el_element->get_value( ).
  el_element = idocument->find_from_name( 'RCVPRT' ).
  ls_edidc-rcvprt = el_element->get_value( ).
  el_element = idocument->find_from_name( 'RCVPRN' ).
  ls_edidc-rcvprn = el_element->get_value( ).
  el_element = idocument->find_from_name( 'CREDAT' ).
  ls_edidc-credat = el_element->get_value( ).
  el_element = idocument->find_from_name( 'CRETIM' ).
  ls_edidc-cretim = el_element->get_value( ).
  el_element = idocument->find_from_name( 'SERIAL' ).
  ls_edidc-serial = el_element->get_value( ).

  ls_edidc-direct = '2'.

  move-corresponding ls_edidc to ls_idx_xmb.

* Connection data to application system to get IDoc metadata
  select single * from  idxporsm59 into ls_idx1
         where  port    = ls_edidc-sndpor
         and    client  = ls_edidc-mandt.

  ls_idx_xmb-port    = ls_edidc-sndpor.
  ls_idx_xmb-rfcdest = 'NONE'. "not necessary

* Get DOCREL and SAPREL
  el_element = idocument->find_from_name( 'DOCREL' ).
  if not el_element is initial.
    ls_edidc-docrel = el_element->get_value( ).
  else.
    select segtyp into table t_segtyp from idxidocsyn
           where  port     = ls_edidc-sndpor
           and    idoctyp  = ls_edidc-idoctyp
           and    cimtyp   = ls_edidc-cimtyp.

    loop at t_segtyp into el_segtyp.
      select released into el_released from idxedisdef
             where  port        = ls_edidc-sndpor
```

```
              and    segtyp      = el_segtyp
              and    actrelease  = 'X'.
      endselect.
      if el_released gt ls_edidc-docrel.
        ls_edidc-docrel = el_released.
      endif.

    endloop.
  endif.

* =====================================
* 2. convert XML to IDoc table structure
* =====================================
  call function 'IDX_XML_TO_IDOC'
    exporting
      xml_data          = source
      guid              = el_message_id
      edidc40           = ls_edidc
      idx_xmb           = ls_idx_xmb
      typ_def           = 'X'
    tables
      idoc_control_40   = t_edidc
      idoc_data_40      = t_edidd
    exceptions
      unknown_xml       = 1
      customizing_error = 2
      no_data_found     = 3
      syntax_error      = 4
      others            = 5.
  if sy-subrc <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  endif.

** To keep the original DOCNUM uncomment the following 8 lines of coding
** Otherwise a new DOCNUM will be created
*  LOOP AT t_edidc into ls_edidc_h.
*    ls_edidc_h-DOCNUM = ls_edidc-DOCNUM.
*    modify t_edidc from ls_edidc_h.
*  ENDLOOP.
*
*  LOOP AT t_edidd into ls_edidd_h.
*    ls_edidd_h-DOCNUM = ls_edidc-DOCNUM.
*    modify t_edidd from ls_edidd_h.
*  ENDLOOP.

* =========================================
* 3. convert IDoc table structure to string
* =========================================
  call function 'ZSOTR_SERV_TABLE_TO_STRING'
*    EXPORTING
*     FLAG_NO_LINE_BREAKS       = ' '
*     LINE_LENGTH               = 0
*     LANGU                     = SY-LANGU
    importing
      text                      = t_resultc
    tables
      text_tab                  = t_edidc.

* convert IDoc table structure to string
  call function 'ZSOTR_SERV_TABLE_TO_STRING'
*    EXPORTING
*    FLAG_NO_LINE_BREAKS       = ' '
*    LINE_LENGTH               = 0
*    LANGU                     = SY-LANGU
    importing
      text                      = t_resultd
```

```
    tables
      text_tab                  = t_edidd.

  concatenate t_resultc t_resultd into t_result.

* convert string to xstring
  call function 'SCMS_STRING_TO_XSTRING'
    exporting
      text           = t_result
*   MIMETYPE       = ' '
*   ENCODING       =
    importing
      buffer         = result.
* EXCEPTIONS
*   FAILED         = 1
*   OTHERS         = 2
  .
  if sy-subrc <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  endif.

endmethod.
```

Implementation of the function ZSOTR_SERV_TABLE_TO_STRING

The following function module is a modification of the function module SOTR_SERV_TABLE_TO_STRING delivered by SAP. The function is called inside the ~EXECUTE method of your ABAP mapping class. It converts each line of an internal table to a string and terminates the end of the line with an operating system dependent carriage return and line feed.

```
FUNCTION ZSOTR_SERV_TABLE_TO_STRING.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(FLAG_NO_LINE_BREAKS) TYPE  AS4FLAG DEFAULT 'X'
*"     VALUE(LINE_LENGTH) TYPE  I OPTIONAL
*"     VALUE(LANGU) TYPE  SYLANGU DEFAULT SY-LANGU
*"  EXPORTING
*"     VALUE(TEXT) TYPE  STRING
*"  TABLES
*"      TEXT_TAB
*"----------------------------------------------------------------------

* concatenate text from table into string
* - with separator SPACE, unless
*   (a) line before ends with sth. not equal SPACE
*       and current line begins with sth. not equal SPACE
*   (b) line before ends with '<' or '-' (no matter how current
*       line begins)
*   (c) current line begins with '>' or SPACE (since space will
*       stay) (no matter how line before ends)

* local data ------------------------------------------------------

*** start modification ***
*  data: l_line               type sotr_txt,
*        l_line_before        type sotr_txt.
  data: l_line               type string, "sotr_txt,
        l_line_before        type string. "sotr_txt.
*** end modification ***


  data: l_strlen_line        type i,
```

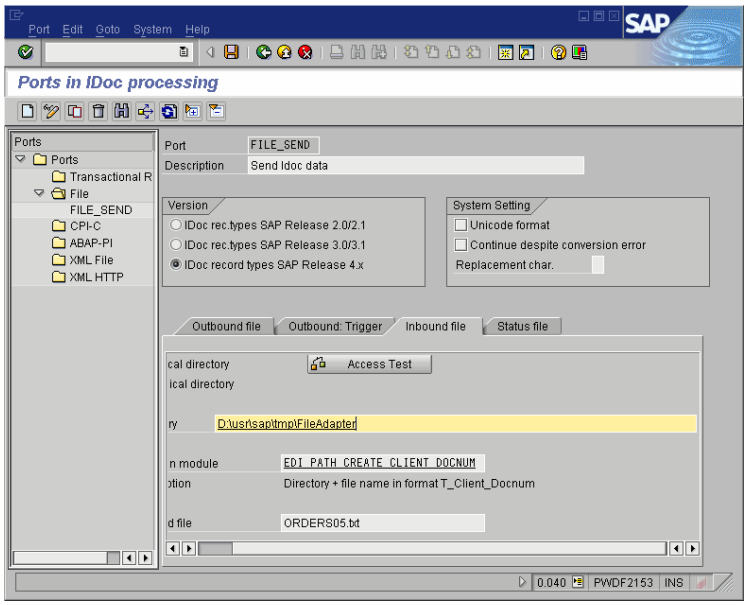
```
        l_strlen_line_before type i.
  data: l_length_tab_line    type i.
  data: l_last_char(1),
        l_last_char_tmp(1),
        l_first_char(1).

*** start modification ***
  class cl_abap_char_utilities definition load.
  data: l_cr_lf(2) type c.
* note: replace here not possible since space is ignored
  l_cr_lf = cl_abap_char_utilities=>cr_lf.
*** end modification ***


* constants ------------------------------------------------------------
  constants: lc_last_chars(2)  value '<-',
             lc_first_chars(2) value '> '.

* get number of lines in text_tab --------------------------------------
  describe table text_tab lines sy-tfill.
* no text --> exit -----------------------------------------------------
  if sy-tfill = 0.
    exit.
  endif.
* only one line of text --> directly into text-string ------------------
  if sy-tfill = 1.
    read table text_tab index 1.

*** start modification ***
*    text = text_tab.
    concatenate text_tab
                l_cr_lf
                into text.
*** end modification ***

* more than one line of text --> concatenate into text-string ---------
  else.
*   determine width of one line in text_tab
    if line_length is initial.
      describe field  text_tab
               length l_length_tab_line in character mode.
    else.
      l_length_tab_line = line_length.
    endif.
*   text_tab --> string
    loop at text_tab into l_line.
*     determine last character of line before
      l_strlen_line_before = strlen( l_line_before ) - 1.
      if l_strlen_line_before > 0.
        l_last_char_tmp = l_line_before+l_strlen_line_before(1).
*       consider blanks at end of line before
        if  strlen( l_line_before ) < l_length_tab_line
        and l_last_char_tmp        na lc_last_chars.
          l_last_char_tmp = space.
        endif.
      else.
        l_last_char_tmp = space.
      endif.
      l_last_char = l_last_char_tmp.
*     determine first character of current line
      if not l_line is initial.
        l_first_char = l_line(1).
      else.
        l_first_char = space.
      endif.
*     concatenate line into text:
*     (a1) without separator, if
```

```
*          - last  character before is < or - OR
*          - first character behind is > or SPACE
      if l_last_char  ca lc_last_chars
      or l_first_char ca lc_first_chars.

*** start modification ***
        concatenate text
                    l_line
                    l_cr_lf
                    into text.
*** end modification ***

*     (a2) without separator, if
*          - last  character before is not SPACE AND
*          - first character behind is not SPACE
      elseif not  l_last_char is initial
      and    not l_first_char is initial.

*** start modification ***
        concatenate text
                    l_line
                    l_cr_lf
                    into text.

**     (b) with separator space
*       else.
*         if langu ca c_no_space_langus.
*           concatenate text
*                       l_line
*                       l_cr_lf
*                       into text.
*** end modification ***

         else.

*** start modification ***
          concatenate text
                      l_line
                      l_cr_lf
                      into text
*                      separated by space
                      .
*         endif.
*** end modification ***

       endif.
       l_line_before = l_line.
    endloop.
  endif.
** delete line breaks ------------------------------------------------
*  if flag_no_line_breaks = 'X'.
*    perform delete_line_breaks
*            changing text.
*  endif.

endfunction.
```

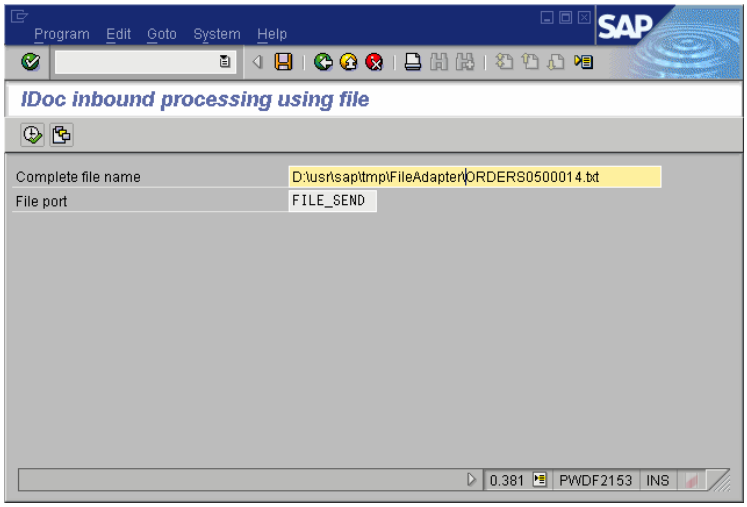
3.2 Convert a flat file representation of an IDoc to IDoc-XML

The following sequence provides a technique to hand over a flat file representation of an IDoc to the IDoc-adapter of your Integration Engine. Please note that the IDoc control record data of your flat file is used to derive the related Sender Service.

7. Create an ALE Port of type 'File' using transaction we21 on your XI Integration Engine.



8. Run Report RSEINB00 on your XI Integration Engine. The Report will upload the file, filter the IDoc header data, perform the conversion to IDoc-XML and put the data into the pipeline of the XI Integration Server. Therefore it is crucial, that the IDoc control record data is filled correctly and corresponds to the adapter specific identifiers of your sender service. Once the message is put successfully into the pipeline of the XI Integration Server, the file is deleted.



www.sdn.sap.com/irj/sdn/howtoguides