

Offline Interactive Forms Using ABAP

Applies to:

Interactive Forms based on Adobe software

Summary

This paper shows the basic steps you need for creating an Interactive Form based on Adobe software for an offline scenario using the SAP delivered function modules and the PDF object for extracting the data. The paper assumes that the reader has a basic understanding of PDF-based form development.

Editor's note:

In general, SAP recommends using Web Dynpro integration (Java or ABAP) of Interactive Forms for interactive scenarios. The Web Dynpro framework handles the required XML transformations automatically in the background so that developers do not need to deal with this aspect manually and on an individual basis.

If you create an interactive scenario in transaction SFP, which was designed to meet printing requirements (i.e. for non-interactive output), you always need to manually code the transformation on the return trip of the PDF (to transfer data entered in the form into the backend).

Author: Vani Krishnamoorthy

Company: SAP America

Created on: 12 May 2006

Author Bio

Vani Krishnamoorthy is an SAP NetWeaver Tools Consultant for SAP America.

Table of Contents

Adobe Interactive Forms: Overview	3
Business Example	3
Designing a Form	3
Form Builder.....	4
Interface	4
Form Context	5
Form Layout.....	5
Generate and Send the Form.....	6
Data Retrieval and Processing	6
Get the Generated Function Module.....	6
Start the Form Processing	6
Call the Generated Function Module	7
End Form Processing.....	8
Send the Form to the Vendor.....	8
Generated Email	11
Payment Form Emailed to Vendor (PDF 82 KB)	11
Filled Form from the Vendor (PDF 82 KB).....	11
Extract Data	11
Upload the Form	11
Instantiate the PDF Object.....	13
Extract the Data	14
Update the Vendor Master.....	15
Related Content.....	17
Copyright.....	18

Adobe Interactive Forms: Overview

Since SAP NetWeaver (Web) Application Server 6.40 (SAP NetWeaver 04), Adobe document services (ADS) have been available. This is a set of runtime services deployed on the Application Server that provide a range of form and document creation and manipulation functions. The key capabilities of the ADS are the creation of documents in PDF and various print formats from XML form templates and current system data, and the extraction of user-entered data from interactive PDF forms for rendering and generating Adobe Forms. SAP has also provided a single programmatic interface called PDF Document Object (or PDF Object) that enables developers to communicate with ADS. PDF Object is available both in ABAP as well as Java.

This paper shows the basic steps you need for creating an Adobe Interactive Form for offline scenario using the SAP delivered function modules and the PDF object for extracting the data. The paper assumes that the reader already has the basics of PDF based form development.

Business Example

The business example in this paper is an offline scenario by which a vendor will be able to fill bank information and send this information back so that this can be updated in the vendor master. The SAP vendor no and vendor name are pre populated in the form. Then this form is emailed to the vendor. The vendor completes the form and sends it back. The data from the PDF form is retrieved and the vendor master is updated. This does not require any Web Dynpro development

Designing a Form

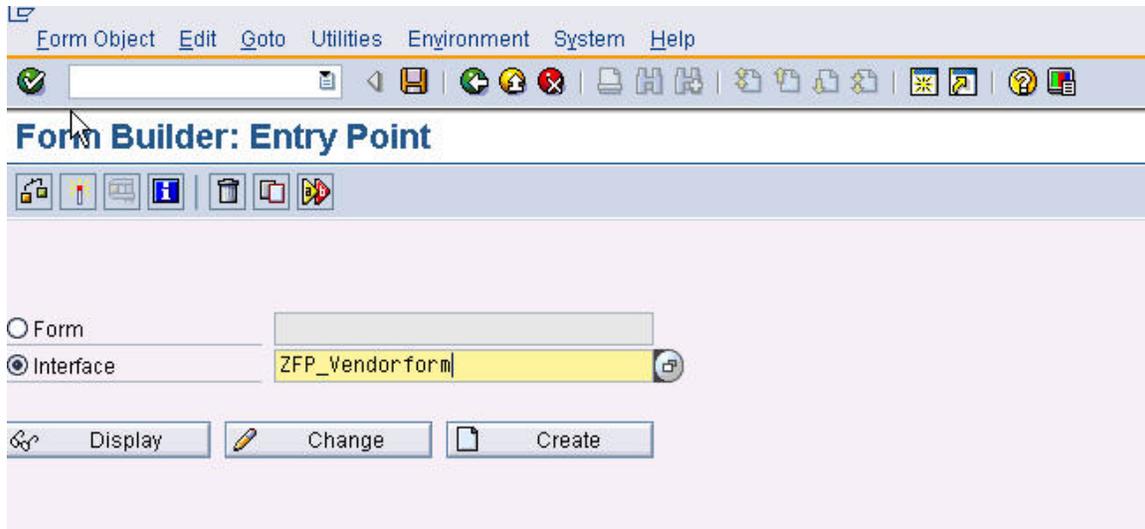
The first step for an offline scenario would be to design a form. This topic has been covered in detail in other How-To documents and is also explained in details in SAP documentation ([Designing PDF Forms](#)). The steps for form design are:

- Start transaction SFP
- Create an interface
- Create a form object
- In the context link the required parameters from the interface
- Finally create the layout of the form and activate the form.

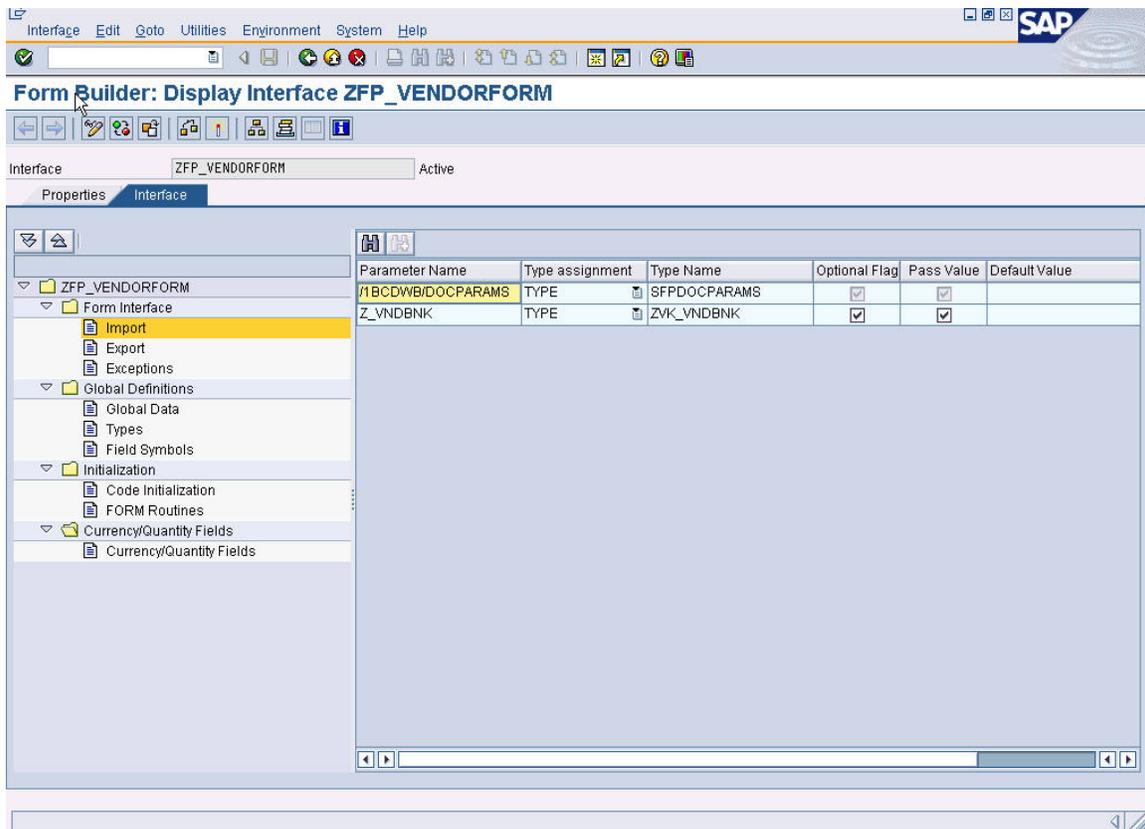
This creates a function module that encapsulates the form description. We will be creating an application program that collects the relevant data, calls this function module so as to generate the fillable PDF form.

Make sure that the ADS is configured and ready for use (including a valid credential – See SAP Note 736902). The credential is required if, for example, the form is to be saved after filling.

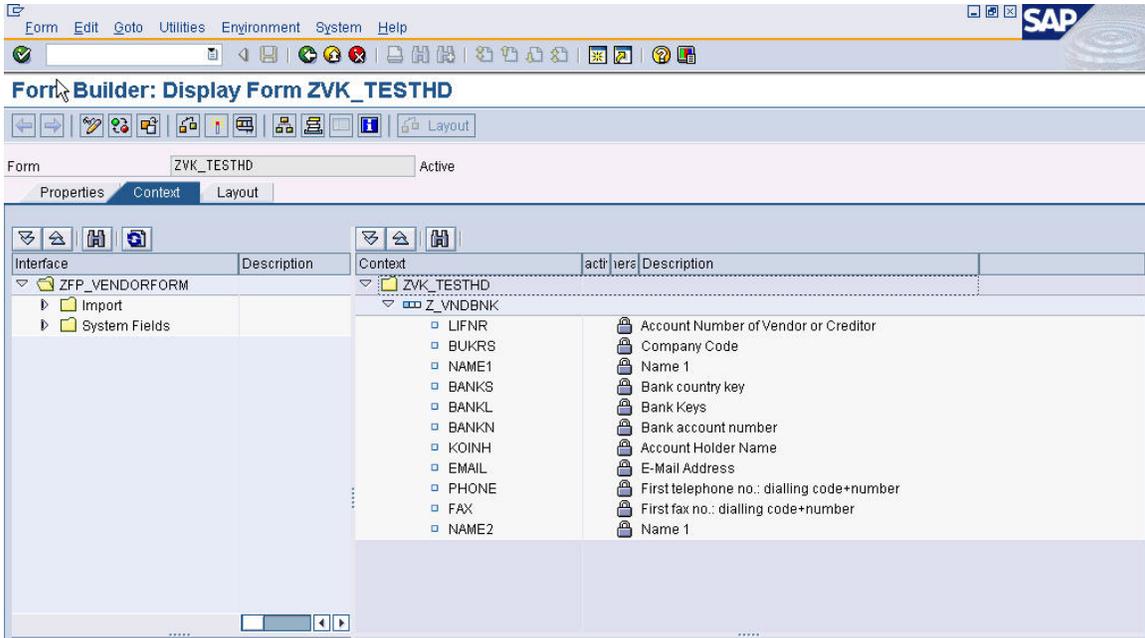
Form Builder



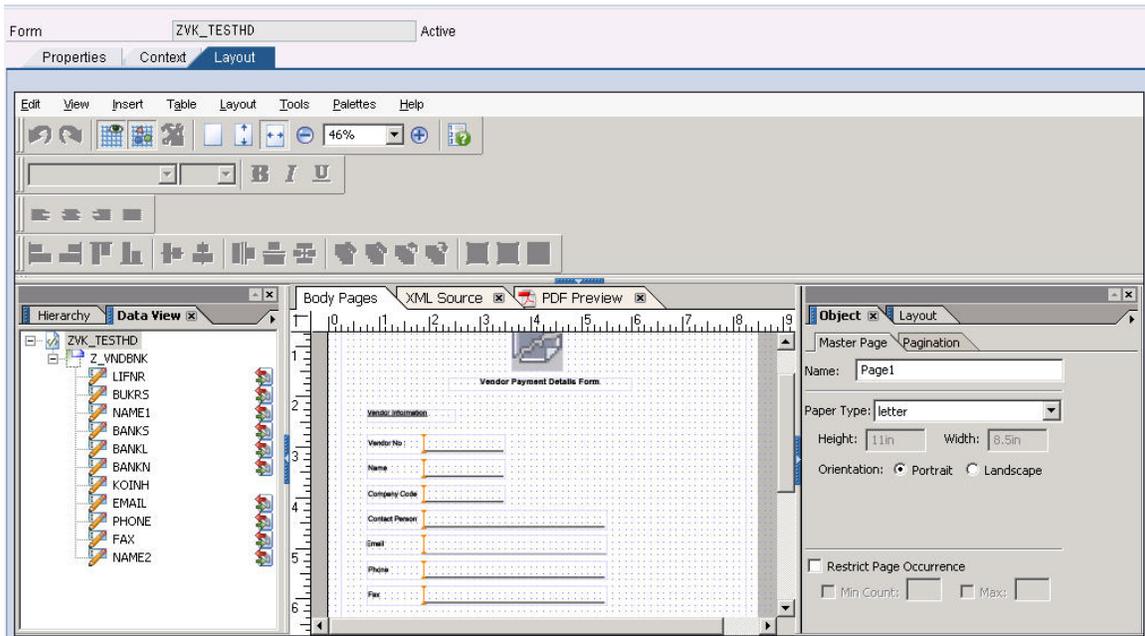
Interface



Form Context



Form Layout



Generate and Send the Form

The next step is to write the ABAP program which will create the form and email it to the vendor so that it can be filled offline.

The program will have the following steps:

- Data retrieval and processing : A select statement for the pre-populated information
- Obtain the name of the Generated Function Module of the form
- Start the form processing
- Call the Generated Function Module
- End form processing
- Send the form to the vendor using Business communication services (BCS)

Data Retrieval and Processing

This can be as simple as a select statement to complex data selection. In this example we select the vendor number, name and company code from the vendor table LFA1 based on the vendor from the selection screen

* Get vendor data

```
select single lifnr name1 bukrs from lfa1 into wa_vndbnk where lifnr = p_lifnr.
```

Get the Generated Function Module

The next step is to get the generated function module. Call function module FP_FUNCTION_MODULE_NAME and pass the form name to it. The parameter e_funcname will contain the name of the generated function module name.

* First get name of the generated function module

```
call function 'FP_FUNCTION_MODULE_NAME'  
  
  exporting  
  
    i_name      = 'ZVK_TESTHD'  
  
  importing  
  
    e_funcname = fm_name.
```

Start the Form Processing

Form printing needs to be explicitly opened and closed. Use the function FP_JOB_OPEN to open the form for printing. The parameter ie_outputparams determines printer settings. This parameter is also where we ask the generated function module to return a PDF file back. Since this is an offline scenario and there is no printing involved we need to suppress the printer dialog popup as well. Optionally there is a parameter connection which can be used to determine the RFC destination for ADS.

* Set output parameters and open spool job

```
fp_outputparams-nodialog = 'X'.    " suppress printer dialog popup
fp_outputparams-GETPDF   = 'X'.    " launch print preview
```

```
call function 'FP_JOB_OPEN'
```

```
  changing
```

```
    ie_outputparams = fp_outputparams
```

```
  exceptions
```

```
    cancel          = 1
```

```
    usage_error     = 2
```

```
    system_error    = 3
```

```
    internal_error  = 4
```

```
    others          = 5.
```

Call the Generated Function Module

This is similar to the generated function module in Smart Forms. Since the parameters of the function module are defined in the interface, this will vary from form to form. However, /1bcdwb/docparams is a standard parameter. This is used to set the forms locale. This is also where we tell the form that it is fillable. Once this parameter is set - if the ADS is configured correctly (including the credential) - a fillable savable form will be returned when the function module is executed.

* Set form language and country (->form locale)

```
fp_docparams-langu    = 'E'.
```

```
fp_docparams-country = 'US'.
```

```
fp_docparams-FILLABLE = 'X'.
```

* Now call the generated function module

```
call function fm_name
```

```
  exporting
```

```
    /1bcdwb/docparams = fp_docparams
```

```
    Z_VNDBNK          = wa_vndbnk
```

```
  importing
```

```
    /1BCDWB/FORMOUTPUT = fp_formoutput
```

```
  exceptions
```

```
usage_error      = 1
system_error     = 2
internal_error   = 3
others           = 4.
```

End Form Processing

Use the function FP_JOB_CLOSE to close the form for printing.

* Close spool job

```
call function 'FP_JOB_CLOSE'
exceptions
  usage_error      = 1
  system_error     = 2
  internal_error   = 3
  others           = 4.
```

Send the Form to the Vendor

The PDF file generated is available in the parameter fp_result which is returned by the generated function module. The next step would be to extract this PDF and send it to the vendor using BCS.

```
CALL FUNCTION 'SCMS_XSTRING_TO_BINARY'
EXPORTING
  buffer          = fp_formoutput-PDF    "PDF file from function module
TABLES
  binary_tab     = lt_att_content_hex.
```

```
CLASS c1_bcs DEFINITION LOAD.
```

```
DATA:
```

```
  lo_send_request TYPE REF TO c1_bcs VALUE IS INITIAL.
```

```
lo_send_request = c1_bcs=>create_persistent( ).
```

* Message body and subject

```
DATA:
```

```
  lt_message_body TYPE bcsy_text VALUE IS INITIAL,
```

```

    lo_document TYPE REF TO cl_document_bcs VALUE IS INITIAL.
APPEND 'Dear Vendor,' TO lt_message_body.
append ' ' to lt_message_body.
APPEND 'Please fill the attached form and send it back to us.'
                TO lt_message_body.

append ' ' to lt_message_body.
APPEND 'Thank You,'      TO lt_message_body.

lo_document = cl_document_bcs=>create_document(
                i_type = 'RAW'
                i_text = lt_message_body
                i_subject = 'Vendor Payment Form' ).

DATA: lx_document_bcs TYPE REF TO cx_document_bcs VALUE IS INITIAL.
TRY.
    lo_document->add_attachment(
        EXPORTING
            i_attachment_type      = 'PDF'
            i_attachment_subject   = 'Vendor Payment Form'
*           I_ATTACHMENT_SIZE     =
*           I_ATTACHMENT_LANGUAGE = SPACE
*           I_ATT_CONTENT_TEXT    =
*           I_ATTACHMENT_HEADER   =
            i_att_content_hex     = lt_att_content_hex ).

    CATCH cx_document_bcs INTO lx_document_bcs.
ENDTRY.

* Add attachment
* Pass the document to send request
lo_send_request->set_document( lo_document ).

* Create sender

DATA:
    lo_sender TYPE REF TO if_sender_bcs VALUE IS INITIAL,
    l_send    type ADR6-SMTP_ADDR value 'Vendappr@HD.com',

```

```
lo_sender = cl_cam_address_bcs=>create_internet_address( l_send ).
```

```
* Set sender
```

```
lo_send_request->set_sender(
```

```
    EXPORTING
```

```
        i_sender = lo_sender ).
```

```
* Create recipient
```

```
*DATA:
```

```
    lo_recipient TYPE REF TO if_recipient_bcs VALUE IS INITIAL.
```

```
lo_recipient = cl_sapuser_bcs=>create( sy-uname ).
```

```
** Set recipient
```

```
lo_send_request->add_recipient(
```

```
    EXPORTING
```

```
        i_recipient = lo_recipient
```

```
        i_express   = 'X' ).
```

```
lo_send_request->add_recipient(
```

```
    EXPORTING
```

```
        i_recipient = lo_recipient
```

```
        i_express   = 'X' ).
```

```
* Send email
```

```
DATA: lv_sent_to_all(1) TYPE c VALUE IS INITIAL.
```

```
lo_send_request->send(
```

```
    EXPORTING
```

```
        i_with_error_screen = 'X'
```

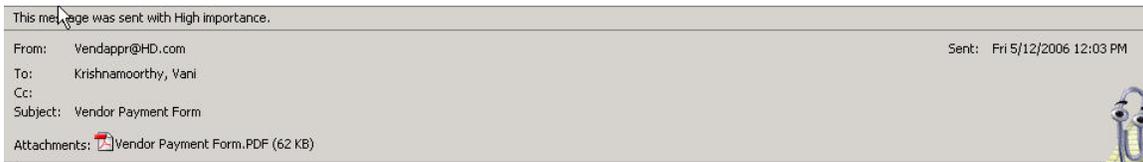
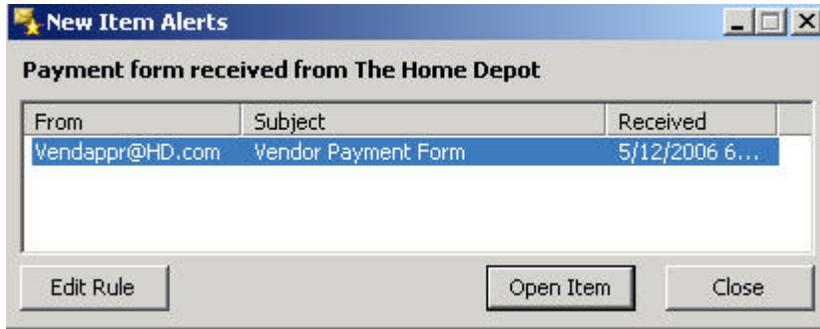
```
    RECEIVING
```

```
        result = lv_sent_to_all ).
```

```
COMMIT WORK.
```

```
message 'The payment form has been emailed to the Vendor' type 'I'.
```

Generated Email



Dear Vendor,

Please fill the attached form and send it back to us.

Thank You,

[Payment Form Emailed to Vendor \(PDF 82 KB\)](#)

[Filled Form from the Vendor \(PDF 82 KB\)](#)

Extract Data

Once the vendor fills the form and sends it back the data needs to be extracted from the PDF file. In this example we are assuming that the vendor sends back the whole PDF file. But we can also make it easier and send only the data as an XML file when the vendor hits the SUBMIT button. For this we will use the PDF document object. SAP provides us with the interfaces IF_FP (Form) and IF_FP_PDF_OBJECT (PDF object). These two are the main interfaces which we will be using. The following are the steps to extract the data from the PDF file.

- Upload the form to the system
- Instantiate a PDF object and assign the PDF file to the object
- Extract the data from the PDF object
- Update the vendor master

Upload the Form

To keep things simple in this example the filled form is saved in the C drive and uploaded using CL_GUI_FRONTEND_SERVICES. But there are many other options like sending the email directly to SAP, Receiving the data using http post etc. but this would be beyond the scope of this paper

```
CALL METHOD cl_gui_frontend_services=>file_open_dialog
```

```
  CHANGING
```

```
    file_table          = lt_file_table
```

```
    rc                  = lv_rc
```

```
*    USER_ACTION        =
```

```
*    FILE_ENCODING      =
```

```
  EXCEPTIONS
```

```
    file_open_dialog_failed = 1
```

```
    cntl_error              = 2
```

```
    error_no_gui            = 3
```

```
    not_supported_by_gui    = 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.
```

```
READ TABLE lt_file_table
```

```
  INTO lv_filename
```

```
  INDEX 1.
```

```
*lv_filename = p_pdf.
```

```
cl_gui_frontend_services=>gui_upload(
```

```
  EXPORTING
```

```
    filename            = lv_filename
```

```
    filetype            = 'BIN'          "Binary
```

```
  IMPORTING
```

```
    filelength          = lv_filelength
```

```
  CHANGING
```

```
    data_tab            = lt_rawtab
```

```
  EXCEPTIONS
```

```
    file_open_error     = 1
```

```

file_read_error      = 2
no_batch             = 3
gui_refuse_filetransfer = 4
invalid_type        = 5
no_authority        = 6
unknown_error       = 7
bad_data_format     = 8
header_not_allowed  = 9
separator_not_allowed = 10
header_too_long     = 11
unknown_dp_error    = 12
access_denied       = 13
dp_out_of_memory    = 14
disk_full           = 15
dp_timeout          = 16
not_supported_by_gui = 17
error_no_gui        = 18
OTHERS              = 19 ).

```

Instantiate the PDF Object

The uploaded file is just a stream of raw data. We need to extract just the data from this file. For this we feed the data to the PDF object and use the methods to extract data. The first step would be to create a form object. Once a form object is created we can create a PDF object and assign the file to this object. The PDF object also needs to be informed that the mode would be to extract data. We can then generate a form by connecting to the assigned ADS.

* Get FP reference

```

DATA: lo_fp TYPE REF TO if_fp VALUE IS INITIAL,
lo_fp = cl_fp=>get_reference( ).

```

* For handling exceptions

```

DATA: lo_fpex TYPE REF TO cx_fp_runtime VALUE IS INITIAL.
TRY.

```

```

* Create PDF Object using destination 'ADS' (<-- this is how it is
* defined in SM59)

```

```

DATA: lo_pdfobj TYPE REF TO if_fp_pdf_object VALUE IS INITIAL.
lo_pdfobj = lo_fp->create_pdf_object( connection = 'ADS' ).

```

* Set document

```
lo_pdfobj->set_document(  
    EXPORTING  
        pdfdata = pdf_data ).
```

* Tell PDF object to extract data

```
lo_pdfobj->set_extractdata( ).
```

* Execute the call to ADS

```
lo_pdfobj->execute( ).
```

Extract the Data

Now that we have a PDF object we can extract the data by the simple call of a method. The extracted data is in XML format. We can do a transformation to convert the data to ABAP internal table. In this example the standard identity transformation has been used which needs a few additional steps of replacing the XML namespace. But a custom transformation can be used instead and these additional steps can be avoided.

```
DATA: xml_data TYPE xstring,  
      lt_xml_data TYPE STANDARD TABLE OF xstring.  
APPEND xml_data TO lt_xml_data.  
lo_pdfobj->get_data(  
    IMPORTING  
        formdata = xml_data ).
```

* Convert XML data from XSTRING format to STRING format

```
DATA: lv_xml_data_string TYPE string.  
CALL FUNCTION 'ECATT_CONV_XSTRING_TO_STRING'  
    EXPORTING  
        im_xstring = xml_data  
    IMPORTING  
        ex_string = lv_xml_data_string.
```

* Remove NEW-LINE character from XML data in STRING format

```
CLASS c1_abap_char_utilities DEFINITION LOAD.  
REPLACE ALL OCCURENCES OF c1_abap_char_utilities=>newline IN
```

```
lv_xml_data_string WITH ''.
```

** Make the XML envelope compliant with identity transform*

```
REPLACE '<?xml version="1.0" encoding="UTF-8"?><data>'
      IN lv_xml_data_string
      WITH '<?xml version="1.0" encoding="iso-8859-1"?><asx:abap xmlns
:asx="http://www.sap.com/abapxml" version="1.0"><asx:values>'
```

```
REPLACE '</data>'
      IN lv_xml_data_string
      WITH '</asx:values></asx:abap>'
```

** Apply the identity transform and convert XML into ABAP in one step*

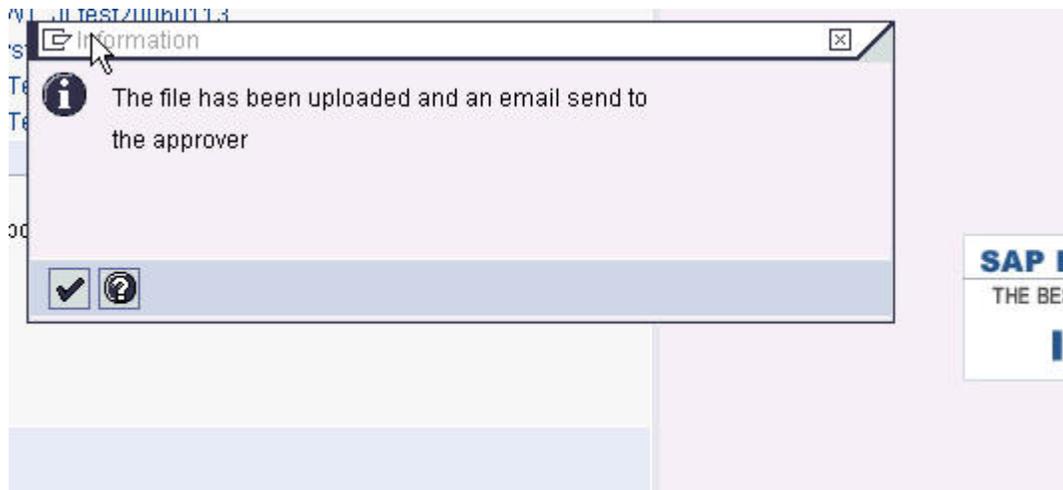
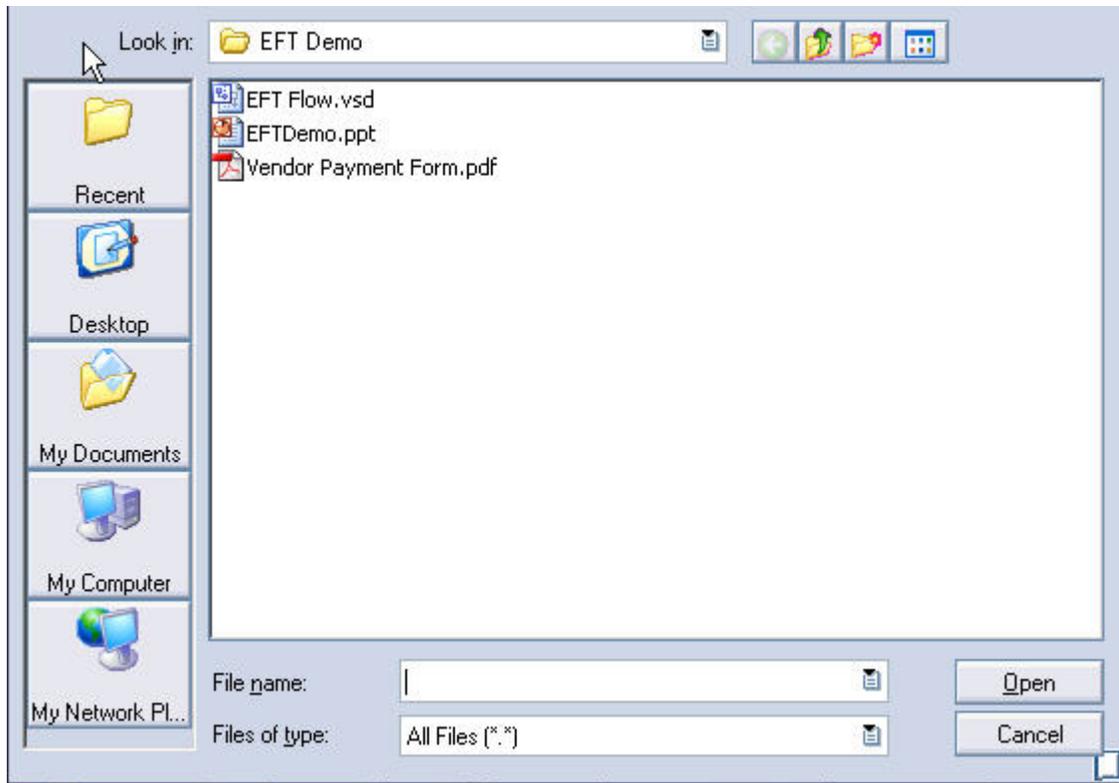
```
DATA: wa_VNDBNK          type ZVK_VNDBNK  VALUE IS INITIAL,
      wa_VENDOR          type ZHD_VENDOR  value is initial,
      lv_subrc TYPE sysubrc VALUE IS INITIAL,
      lt_messtab TYPE STANDARD TABLE OF bdcmsgcoll,
      l_key type SWR_STRUCT-OBJECT_KEY,
      l_pack type zhd_vendor-lifnr.
```

```
CALL TRANSFORMATION id
      SOURCE XML lv_xml_data_string
      RESULT Z_VNDBNK = wa_vndbnk.
```

Update the Vendor Master

Now that the data is available in the internal table the vendor master is updated using standard SAP function calls.

Upload the form



Update Vendor Master

Navigation icons: [Home] [Back] [Forward] [Print] [Refresh] [Cancel] [Save] [Apply] [OK] [Cancel]

CIN Details Test Screen Group

Vendor: 100203 Black and Decker

Bank details

Ctry	Bank Key	Bank Account	Acct holder	C..	IBAN	BnkT	Reference details	C..	Name of bank
US	238100235	3456789			→				<input type="checkbox"/> Mellon Bank

Bank data...

Payment transactions

Alternative payee

DME indicator

Instruction key

PBC/POR number

Alternative payee in document

Individual spec.

Spec. per reference

Related Content

1. [Interactive Forms Based on Adobe Software](#)
2. [Creating Print Forms](#)
3. [Creating Interactive Forms](#)

Copyright

© Copyright 2006 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

Any software coding and/or code lines/strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.