SDN Contribution

Calling services and queries in SAP xMII 11.5 from ABAP

Applies to: SAP xMII 11.5 SR1.

Summary:

How to make use of SAP xMII services and queries from ABAP and shows an example of such a process in the PP/PI sheet scenario, where manufacturing control system data is provided by SAP and integrated within PI sheet.

Author(s):

Bimal Mehta Sr. Solutions Manager Applications Solution Management SAP LABS, LLC 350 Eagleview Blvd. Suite 110 Exton, PA 19341 *O: 610-903-8000 x138* *F: 610-492-9880* *M: 267-307-3730* ***bimal.mehta@sap.com***	Stephan Boecker Solutions Architect Applications Solution Management SAP LABS, LLC Miami, FL *M: +1 941 524-7611* **stephan.boecker@sap.com**

Created on: June15th, 2006

Table of Contents

Introduction

Overview:

One of the core features of xMII 11.5 is the ability to combine data from different sources, including data from SAP R/3 or mySAP ERP systems. In this context xMII acts as a client, e.g. by calling RFC’s in the R/3 or mySAP ERP system. The necessary customizing in xMII to be able to perform these calls is well documented. However, there are scenarios, e.g displaying shop floor data in an SAP PP/PI sheet, where a SAP system needs manufacturing related data from xMII. Here the SAP system acts as client to xMII, pulling data out of xMII and using in an ABAP program. This article shows several possibilities on how to make use of xMII services and queries from ABAP and shows an example of such a process in the PP/PI sheet scenario, where manufacturing control system data is provided by SAP and integrated within PI sheet:

Communication possibilities from ABAP to xMII

IDOC

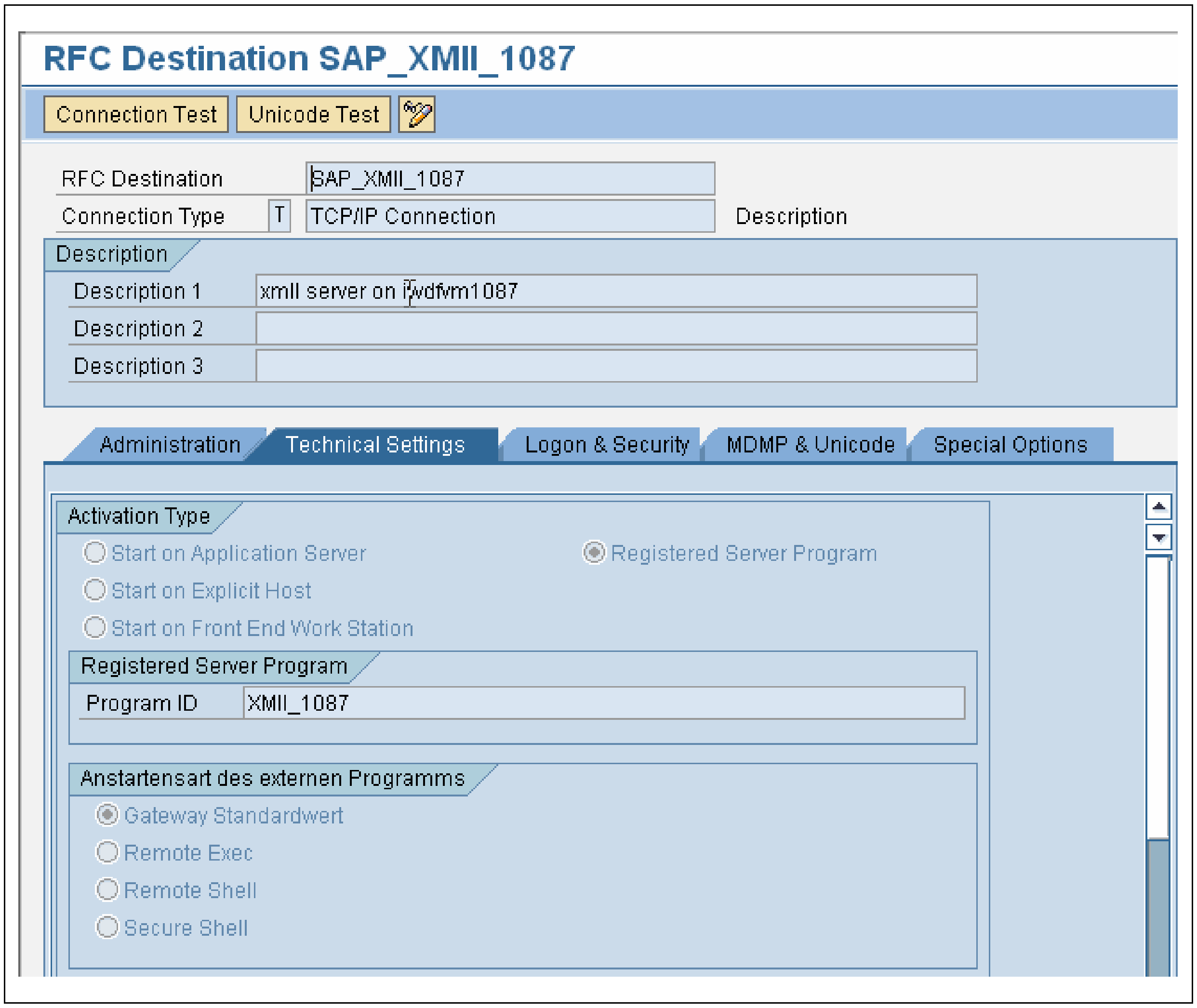
SAP xMII 11.5 can be configured to support the receipt of SAP IDOCS via an ‘Idoc listener’. For details on how to set up such a listener pleas refer to the document by Bimal Mehta (xAppsHowToGuidexMII_iDocListenerSetup.pdf on the SDN)

RFC

The above mentioned Idoc listener is configured as an RFC server in an SAP system. Therefore the corresponding destination can be used in RFC calls. However, synchronous processing is not possible. Only ‘pushing’ of data is supported, so you are limited to the IMPORT and TABLES parameters in your function module. Practically this is identical to using t-RFC and q-RFC calls in your ABAP code. The configuration in xMII is the same as in the IDOC case.

Example

Our example will use an RFC destination named SAP_XMII_1087 in SAP transaction ‘sm59’ with a registered program id of xMII_1087:



RFC Destination SAP_XMII_1087

RFC Destination: SAP_XMII_1087
Connection Type: T TCP/IP Connection

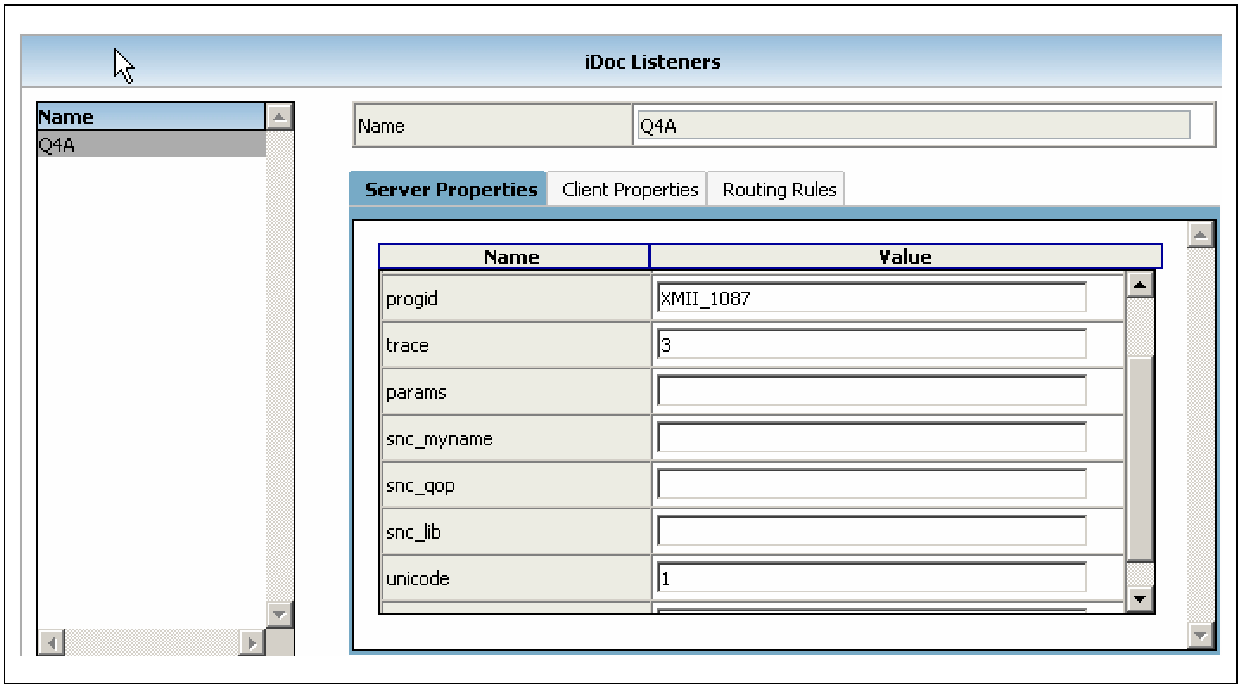
Description
Description 1: xmll server on iwdfvm1087
Description 2:
Description 3:

Activation Type
Start on Application Server [✓ Registered Server Program]
Start on Explicit Host
Start on Front End Work Station

Registered Server Program
Program ID: XMII_1087

Anstartensart des externen Programms
[✓ Gateway Standardwert]
Remote Exec
Remote Shell
Secure Shell

In xMII the same program id is used for the IDOC listener (=RFC server):



Name
Q4A

Name	Q4A

Name	Value
progid	XMII_1087
trace	3
params	
snc_myname	
snc_qop	
snc_lib	
unicode	1

Build function module to ‘push’ data

As described above only the IMPORT and TABLES parameters of the function module are considered. Our example will use an integer number and a table to be pushed to xMII:

```
FUNCTION Z_XMII_SERVER.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     REFERENCE(I_VAL) TYPE  INT4
*"  TABLES
*"      I_TABLE STRUCTURE  DD02L
*"----------------------------------------------------------------------
ENDFUNCTION.
```

Please note that there is no further source code in this function module since its only purpose is to serve as container for the data transport to xMII.

Call function module

Call the function module created above. A simple example would be:

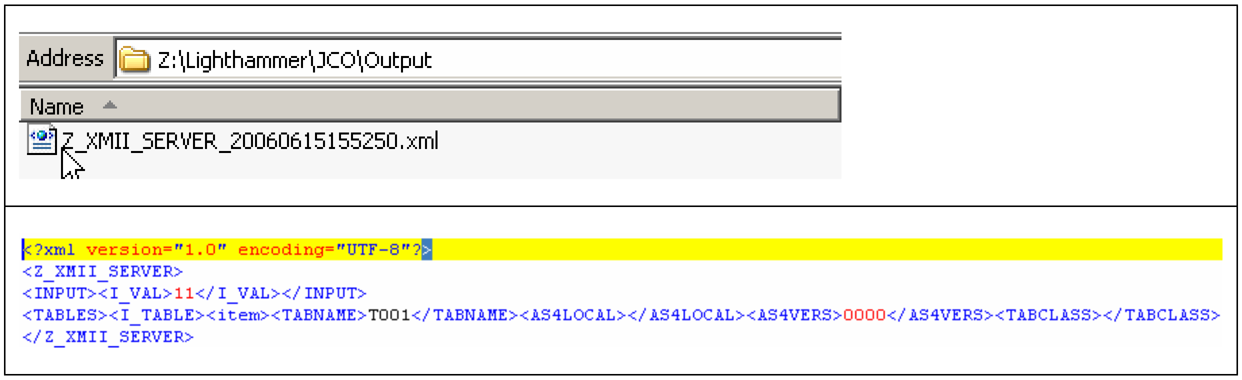
```
data: i_val type int4,
      if_table type dd02l,
      it_table type standard table of dd02l.

* fill data for RFC call
i_val = 11.
if_table-tabname = 'T001'.
append if_table to it_table.

* push data to xMII
CALL FUNCTION 'Z_XMII_SERVER'
IN BACKGROUND TASK
DESTINATION 'SAP_XMII_1087'
  EXPORTING
    I_VAL         = i_val
  TABLES
    I_TABLE       = it_table.
commit work.
```

Receive data as RFC XML

On the xMII side the data can be retrieved from the /Lighthammer/JCO/Output directory. The .xml file is named after the function and contains an additional timestamp. It will contain the data filled in the previous call.



Z_XMII_SERVER_20060615155250.xml

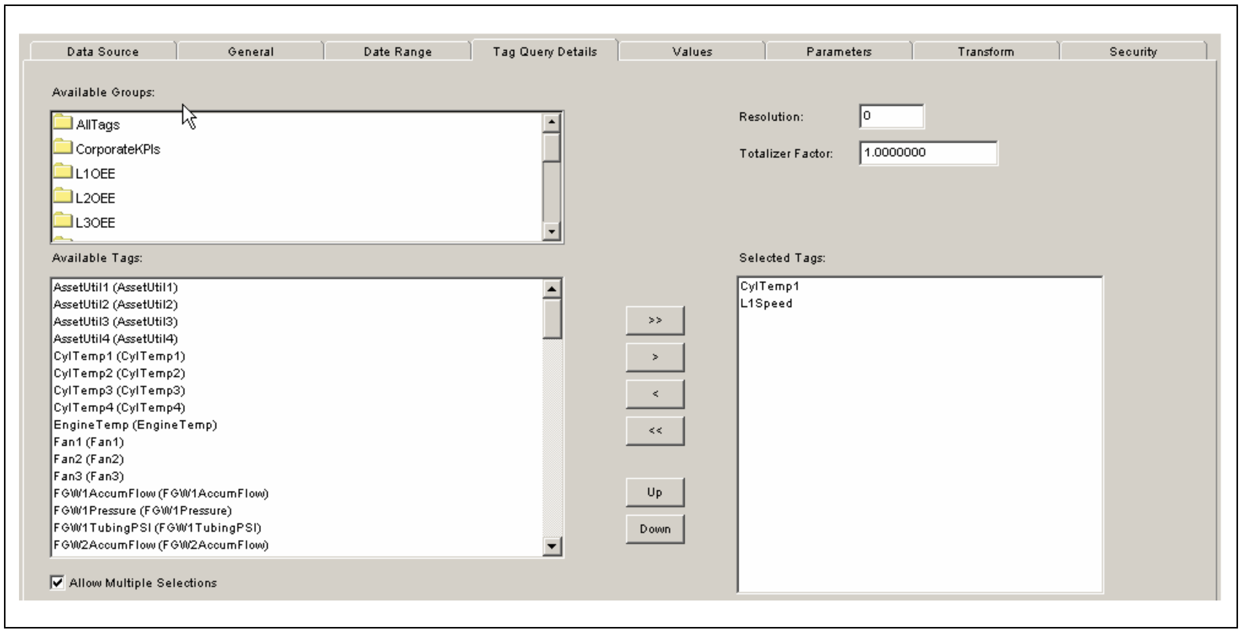
```
<?xml version="1.0" encoding="UTF-8"?>
<Z_XMII_SERVER>
<INPUT><I_VAL>11</I_VAL></INPUT>
<TABLES><I_TABLE><item><TABNAME>T001</TABNAME><AS4LOCAL></AS4LOCAL><AS4VERS>0000</AS4VERS><TABCLASS></TABCLASS>
</Z_XMII_SERVER>
```

HTTP

Queries and services from xMII are available by constructing a URL request containing the necessary query parameters. The URL response will contain the query/service data.

Queries

Query data can be received programmatically via a URL specifying the query template to be used. Lets assume you created a tag query named ‘TagTest’ in the default directory called ‘UserTemplates’:



Available Groups:
AllTags
CorporateKPIs
L1OEE
L2OEE
L3OEE

Resolution: 0
Totalizer Factor: 1.0000000

Available Tags:
AssetUtil1 (AssetUtil1)
AssetUtil2 (AssetUtil2)
AssetUtil3 (AssetUtil3)
AssetUtil4 (AssetUtil4)
CylTemp1 (CylTemp1)
CylTemp2 (CylTemp2)
CylTemp3 (CylTemp3)
CylTemp4 (CylTemp4)
EngineTemp (EngineTemp)
Fan1 (Fan1)
Fan2 (Fan2)
Fan3 (Fan3)
FGW1AccumFlow (FGW1AccumFlow)
FGW1Pressure (FGW1Pressure)
FGW1TubingPSI (FGW1TubingPSI)
FGW2AccumFlow (FGW2AccumFlow)

Selected Tags:
CylTemp1
L1Speed

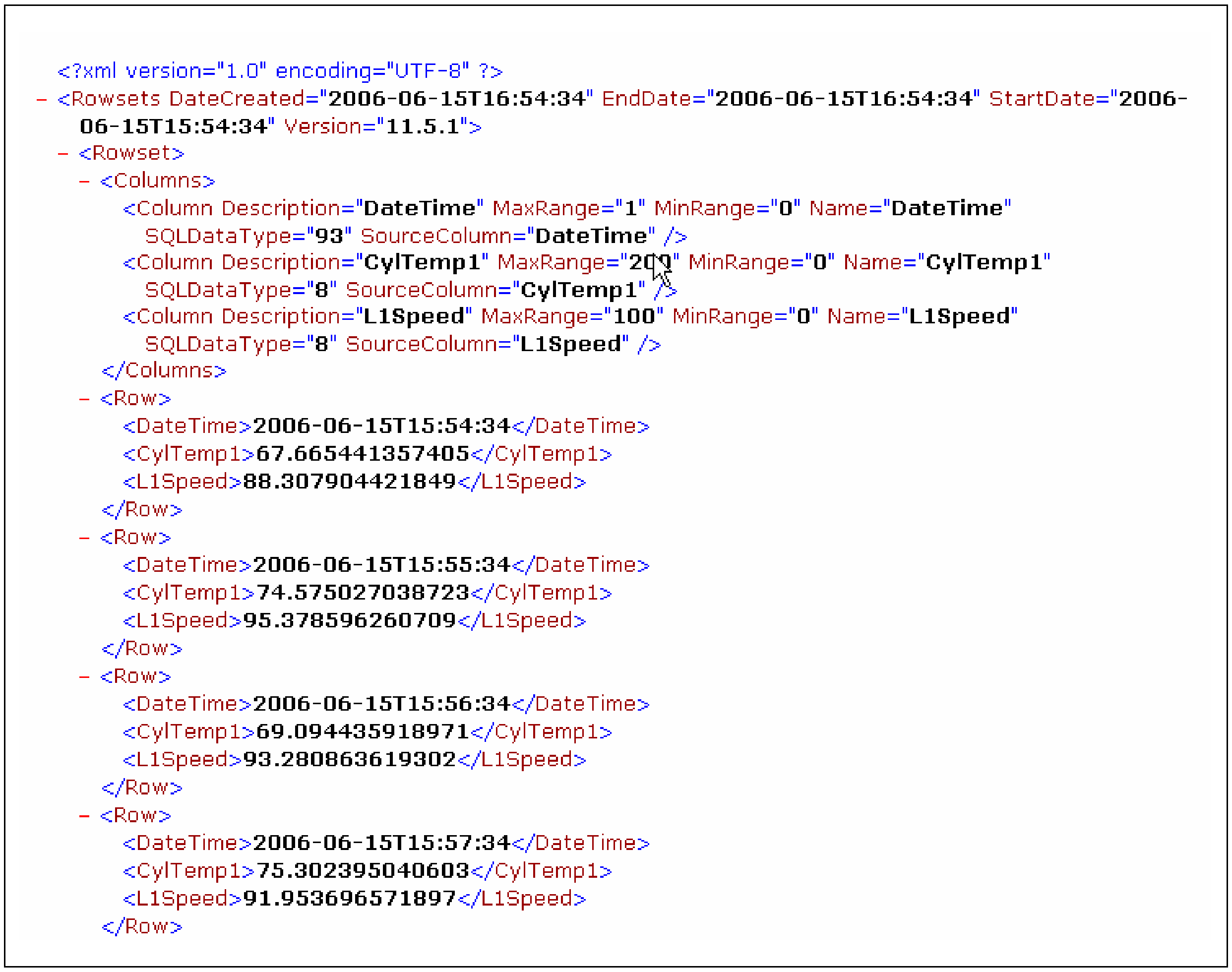
☑ Allow Multiple Selections

This tag query will read the history of two tags called ‘CyclTemp’1 and ‘L1Speed’. In order to receive the tag values programmatically from an xMII instance, which runs on a server called ‘iwdfvm1087’’ port 1080 you would use the following URL:

http://iwdfvm1087:1080/Lighthammer/Illuminator?QueryTemplate=UserTemplates/TagTest&Content-Type=text/xml&IllumLoginName=admin&IllumLoginPassword=XXXX

where the parameter ‘Content-Type’ specifies that you want to receive XML as output and the parameters ‘IllumLoginName’ and ‘IllumLoginPassword’ specify the user and password to log on programmatically to xMII. The set of parameters allowed in a URL is large, so please refer to the ‘Query parameter Reference’ in the xMII Help menu for more details.

The HTTP response body of this call would look like:

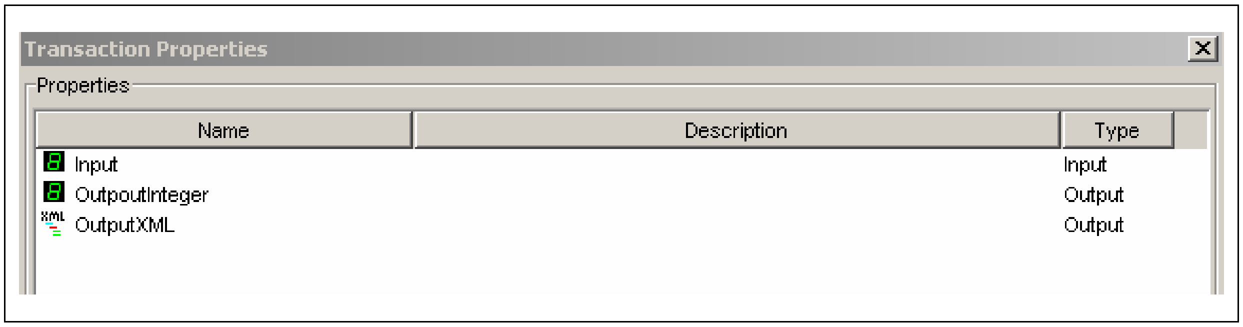


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Rowsets DateCreated="2006-06-15T16:54:34" EndDate="2006-06-15T16:54:34" StartDate="2006-
    06-15T15:54:34" Version="11.5.1">
  - <Rowset>
    - <Columns>
        <Column Description="DateTime" MaxRange="1" MinRange="0" Name="DateTime"
          SQLDataType="93" SourceColumn="DateTime" />
        <Column Description="CylTemp1" MaxRange="200" MinRange="0" Name="CylTemp1"
          SQLDataType="8" SourceColumn="CylTemp1" />
        <Column Description="L1Speed" MaxRange="100" MinRange="0" Name="L1Speed"
          SQLDataType="8" SourceColumn="L1Speed" />
      </Columns>
    - <Row>
        <DateTime>2006-06-15T15:54:34</DateTime>
        <CylTemp1>67.665441357405</CylTemp1>
        <L1Speed>88.307904421849</L1Speed>
      </Row>
    - <Row>
        <DateTime>2006-06-15T15:55:34</DateTime>
        <CylTemp1>74.575027038723</CylTemp1>
        <L1Speed>95.378596260709</L1Speed>
      </Row>
    - <Row>
        <DateTime>2006-06-15T15:56:34</DateTime>
        <CylTemp1>69.094435918971</CylTemp1>
        <L1Speed>93.280863619302</L1Speed>
      </Row>
    - <Row>
        <DateTime>2006-06-15T15:57:34</DateTime>
        <CylTemp1>75.302395040603</CylTemp1>
        <L1Speed>91.953696571897</L1Speed>
      </Row>
```

This structure is worth to be understood, since this internal 'xMII XML' format will be used for all query communication in xMII. Data is represented as a Row set, similar to a record set, where the first element contains the column metadata followed by a set of rows containing a timestamp and the actual data. There can be several Row sets in one document.

Xacute transactions

Xacute transactions can be triggered via a URL request as well. Assume you created a transaction with the name of 'HttpTest', which contains an Integer input parameter, an integer output parameter and an XML output:


Transaction Properties

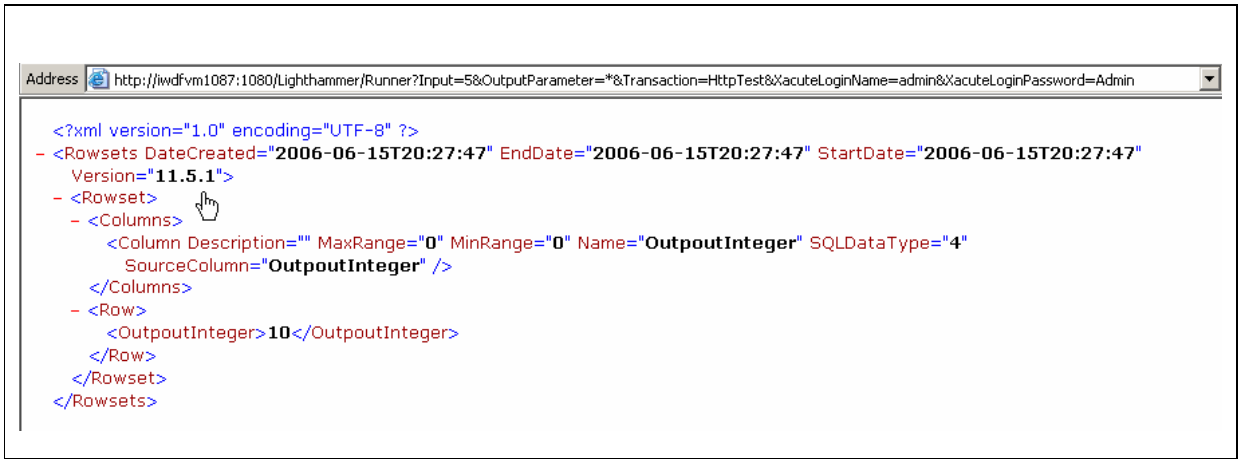
Properties

Name	Description	Type
Input		Input
OutpoutInteger		Output
OutputXML		Output

The transaction itself will take the input integer, multiply by a factor of 2 and assign the result to the 'OutputInteger' parameter. The XML output parameter is filled from a BAPI call to SAP (BAPI_BANK_GETLIST, returning up to 3 rows). The output of this BAPI call will be in RFC-XML format. In order to trigger this example transaction form a URL, use the following request:

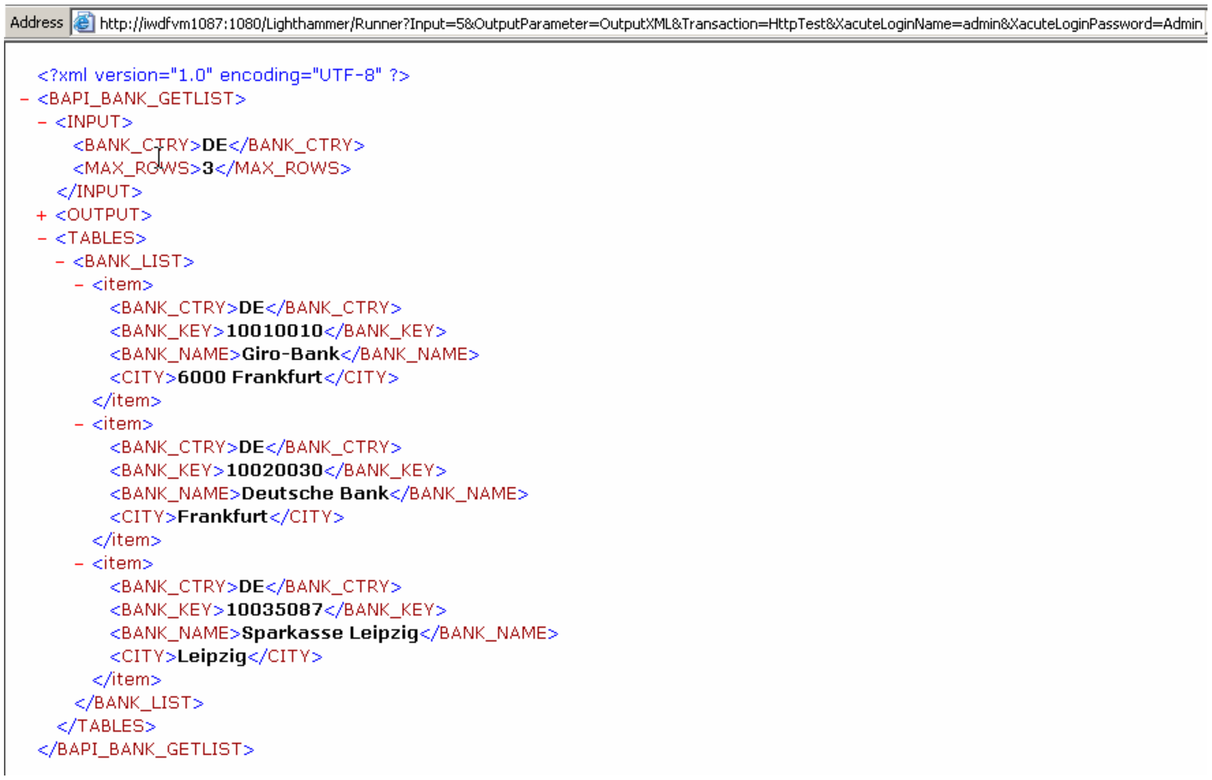
http://iwdfvm1087:1080/Lighthammer/Runner?Input=5&OutputParameter=*&Transaction=HttpTest&XacuteLoginName=yyyy&XacuteLoginPassword=xxxx

where 'yyy' and 'xxxx' stand for your login name and credentials. We are again using this name and the password to log on to xMII programmatically. We as well set the input parameter to be 5. Since we specified with the URL parameter 'OutputParamter=*", that we want to receive all primitive (non-XML transaction output parameters), our result will contain the output in the familiar xMII-XML format:


Address http://iwdfvm1087:1080/Lighthammer/Runner?Input=5&OutputParameter=*&Transaction=HttpTest&XacuteLoginName=admin&XacuteLoginPassword=Admin

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Rowsets DateCreated="2006-06-15T20:27:47" EndDate="2006-06-15T20:27:47" StartDate="2006-06-15T20:27:47"
    Version="11.5.1">
  - <Rowset>
    - <Columns>
        <Column Description="" MaxRange="0" MinRange="0" Name="OutpoutInteger" SQLDataType="4"
          SourceColumn="OutpoutInteger" />
      </Columns>
    - <Row>
        <OutpoutInteger>10</OutpoutInteger>
      </Row>
    </Rowset>
  </Rowsets>
```

If we want to receive the XML output from the RFC call, we would specify the URL parameter 'OutpoutParameter=OutputXML', to retrieve the XML in the http response:



```
http://iwdfvm1087:1080/Lighthammer/Runner?Input=5&OutputParameter=OutputXML&Transaction=HttpTest&XacuteLoginName=admin&XacuteLoginPassword=Admin
```

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <BAPI_BANK_GETLIST>
  - <INPUT>
      <BANK_CTRY>DE</BANK_CTRY>
      <MAX_ROWS>3</MAX_ROWS>
    </INPUT>
  + <OUTPUT>
  - <TABLES>
    - <BANK_LIST>
      - <item>
          <BANK_CTRY>DE</BANK_CTRY>
          <BANK_KEY>10010010</BANK_KEY>
          <BANK_NAME>Giro-Bank</BANK_NAME>
          <CITY>6000 Frankfurt</CITY>
        </item>
      - <item>
          <BANK_CTRY>DE</BANK_CTRY>
          <BANK_KEY>10020030</BANK_KEY>
          <BANK_NAME>Deutsche Bank</BANK_NAME>
          <CITY>Frankfurt</CITY>
        </item>
      - <item>
          <BANK_CTRY>DE</BANK_CTRY>
          <BANK_KEY>10035087</BANK_KEY>
          <BANK_NAME>Sparkasse Leipzig</BANK_NAME>
          <CITY>Leipzig</CITY>
        </item>
      </BANK_LIST>
    </TABLES>
  </BAPI_BANK_GETLIST>
```

For further details on the possible parameters for a Xacute transaction please refer to the SAP xMII Business Logic Services Integration Guide in the xMII help.

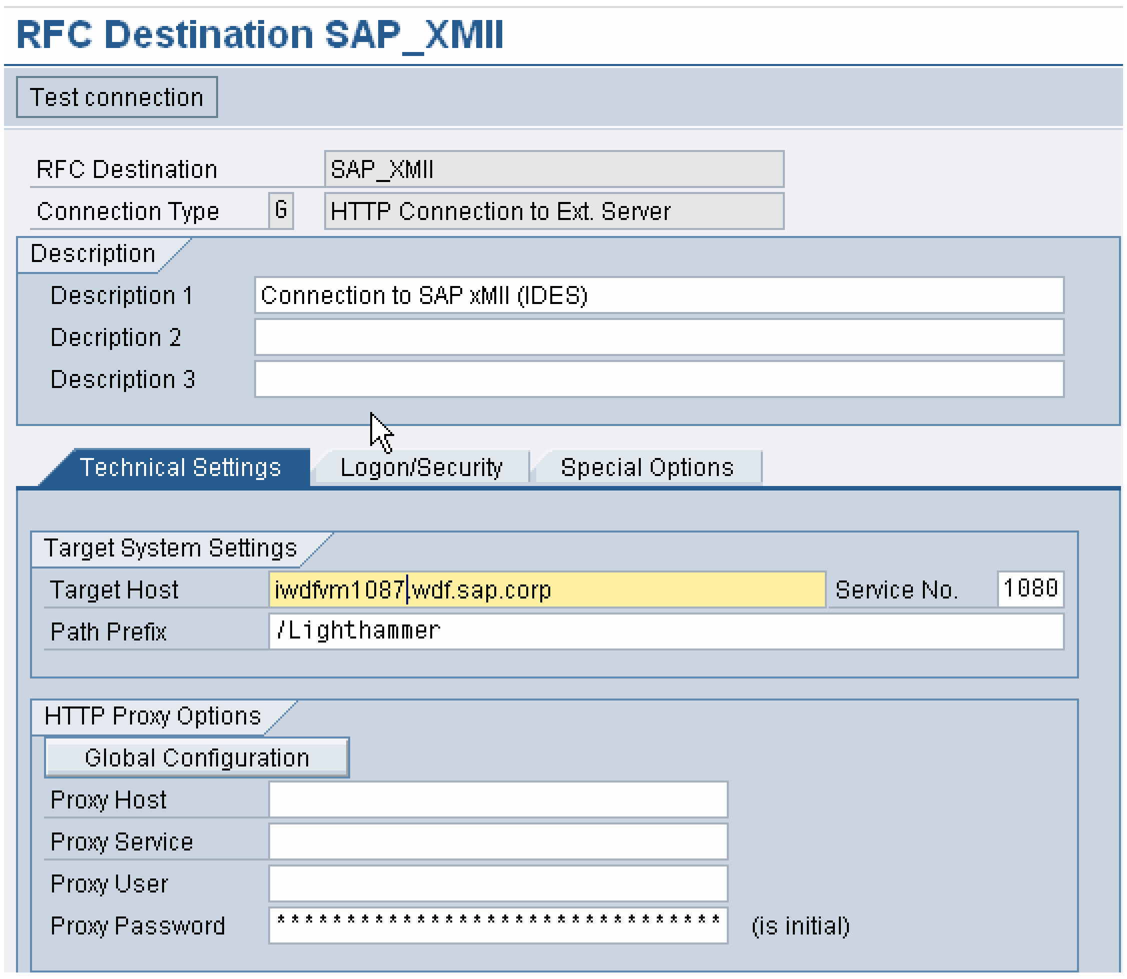
Xacute queries.

Since you can create queries based on Xacute transactions (so called Xacute queries) you can use the parameters specified under the section named queries to pass them to the underlying Xacute transaction. This scenario is only mentioned here since it provides no additional functionality. Please check out the query parameter reference in the SAP xMII help section for details on Xacute queries.

Usage in ABAP

In order to use queries and transactions triggered via URL in ABAP you have to perform the following steps:

Set up HTTP connection to xMII. Using transaction ‘sm59’ set up a connection of type ‘HTTP connection to external server’:



RFC Destination SAP_XMII

RFC Destination	SAP_XMII
Connection Type	G HTTP Connection to Ext. Server

Description

Description 1	Connection to SAP xMII (IDES)
Decription 2	
Description 3	

Target System Settings

Target Host	iwdfvm1087.wdf.sap.corp	Service No.	1080
Path Prefix	/Lighthammer		

HTTP Proxy Options

Proxy Host	
Proxy Service	
Proxy User	
Proxy Password	******************************** (is initial)

Please note that xMII does not support Basic authentication. You can either use SSO or – as in this example – use the user and password as URL parameters to achieve a programmatic logon.

Build request object in your ABAP code using the destination specified above (which is the parameter i_rfc_destination).

```
  * create http client object

  call method cl_http_client=>create_by_destination

    exporting destination   = i_rfc_destination
```

```
      importing client        = client
       EXCEPTIONS
         argument_not_found       = 1
         destination_not_found    = 2
         destination_no_authority = 3
         plugin_not_active        = 4
         internal_error           = 5
         OTHERS                   = 6.
      if sy-subrc <> 0.
       raise HTTP_DEST_NOT_FOUND.
      endif.

    * set request method
      CALL METHOD client->request->set_header_field
           EXPORTING
             name  = '~request_method'
             value = 'GET'.
```

Build complete URL depending on your scenario. In our example we want to access xMII query data via the URL request, so we will use the path beginning with 'Illuminator'. Please note that the variables used here starting with an 'i_' are all input variables. The full URL path is kept in the variable 'path'.

```
    build request path to Illuminator service
     path = '/Illuminator'.

    * build query parameters
     refresh: it_query_field.
     clear if_query_field.
     if_query_field-name = 'QueryTemplate'.
     concatenate i_template_folder '/' i_template_name into
    if_query_field-value.
     append if_query_field to it_query_field.


     clear if_query_field.
     if_query_field-name = 'Content-Type'.
     if_query_field-value = 'text/xml'.
     append if_query_field to it_query_field.


      if not i_user_name is initial.
      if_query_field-name = 'IllumLoginName'.
      if_query_field-value = i_user_name.
      append if_query_field to it_query_field.


      if_query_field-name = 'IllumLoginPassword'.
      perform get_password
                              using i_user_password
                              changing if_str.
      if_query_field-value = if_str.
      append if_query_field to it_query_field.
      endif.



    * build query string
      if_query = cl_http_utility=>fields_to_string( fields = it_query_field
    encode = 0 ).


    * build path
     concatenate path '?' if_query into path.
```

```
CALL METHOD client->request->set_header_field

       EXPORTING

         name  = '~request_uri'

         value = path.
```

Please note as well that there is a form named ‘get_password’ which takes the base64 encoded password from the input and converts to unencoded string. This does not provide any additional security since the clear text password is transmitted in the URL itself but is more included here to be consistent with the overall concept of xMII to store passwords as base64-encoded strings

```
*&---------------------------------------------------------------------
*

*&      Form  get_password

*&---------------------------------------------------------------------
*

*       get password string from base64 encoded value

*----------------------------------------------------------------------
*

*      -->i_password_encoded: encoded password

*      <--e_password: decoded password

*----------------------------------------------------------------------
*

FORM get_password     USING    i_password

                      CHANGING e_password.


data: i_pwd type string,

      e_pwd type string.

i_pwd = i_password.

e_pwd = cl_http_utility=>decode_base64( encoded = i_pwd ).

e_password = e_pwd.

ENDFORM.                    " get_password
```

Send the request and parse the returned XML stream. This piece is fully dependent on your chosen scenario. In our example we receive an xMII-XML result set containing a list of tag values and fill the data into a simple return table which contains a timestamp, the user and the value of the tag. Please note that your need to include the type-pool ‘ixml’ for this.

```
* send request
    call method client->send
        exceptions
         http_communication_failure = 1
         others = 4.
  if sy-subrc <> 0.
   call method client->close( ).
   RAISE HTTP_COMMUNICATION_FAILURE.
  endif.


* get response
    CALL METHOD client->receive
      EXCEPTIONS
        http_communication_failure = 1
        http_invalid_state         = 2
        http_processing_failed     = 3
        OTHERS                     = 4.
  if sy-subrc <> 0.
     call method client->close( ).
     RAISE HTTP_COMMUNICATION_FAILURE.
  endif.


* check that return is an xml document
   if client->response->get_header_field( 'content-type' ) <>
'text/xml'
```

```
.
    raise NO_XML_DOCUMENT.
   endif.

   if_response = client->response->get_data( ).
      CALL FUNCTION 'SDIXML_XML_TO_DOM'
           EXPORTING
              xml                   = if_response
            IMPORTING
              document              = ref_document
           EXCEPTIONS
             invalid_input        = 1
             OTHERS               = 2.



        call method client->close( ).
       if sy-subrc <> 0.
        raise NO_XML_DOCUMENT.
       endif.
*  look for error messages in xml document
   row_items = ref_document->get_elements_by_tag_name( name =
'FatalError' depth = 2 ).
   ilen = row_items->get_length( ).
   if ilen > 0.

*    get actual Error element
     row_node = row_items->get_item( 0 ).
     row_node_element ?= row_node->query_interface( ixml_iid_element ).
     e_bapiret-message = row_node_element->get_value( ).
   endif.


* get list of <Row> elements

  row_items = ref_document->get_elements_by_tag_name( name = 'Row'
depth

 = 2 ).

* iterate over list

  ilen = 0.

  while ilen < row_items->get_length( ).

* get actual <Row> element

   row_node = row_items->get_item( ilen ).

   row_node_element ?= row_node->query_interface( ixml_iid_element ).


*  read <DateTime> child element and assign to if_timestamp

   dateTimeElement = row_node_element->find_from_name( name =
'DateTime'

 ).

   if_str = dateTimeElement->get_value( ).

   perform adjust_timestamp

                          using if_str

                          changing if_timestamp.

*  loop over all children of this <Row> element

   childelements = row_node_element->get_children( ).

   jlen = 0.

   while jlen < childelements->get_length( ).

     clear: if_tag_data.

*     get actual child element

      childnode = childelements->get_item( jlen ).

      childnode_element ?= childnode->query_interface(
ixml_iid_element).
```

```
*     process only those elements which are not of time time stamp

      if ( childnode_element->get_name( ) <> dateTimeElement-
>get_name() ).

*     fill return structure  and append

         if_tag_data-kpi_id = childnode_element->get_name( ).

         if_str = childnode_element->get_value( ).

         move if_str to if_tag_data-value.

         if_tag_data-creationdate = if_timestamp.

         if_tag_data-uname = sy-uname.

         append if_tag_data to e_tag_values.

      endif.

     jlen = jlen + 1.

   endwhile. " end loop over jlen

   ilen = ilen + 1.

  endwhile. " end loop over ilen
```

Web service

You can consume any xMII Business transaction as a web service. Since it is possible to create client proxies in ABAP you can trigger an xMII Business transaction using that proxy. The wsdl can be found by specifying the URL using the path ‘/WSDLGen/<transactionName>. In our example – using the transaction ‘HttpTest’ -this would be the URL

http://iwdfvm1087:1080/Lighthammer/WSDLGen/HttpTest.

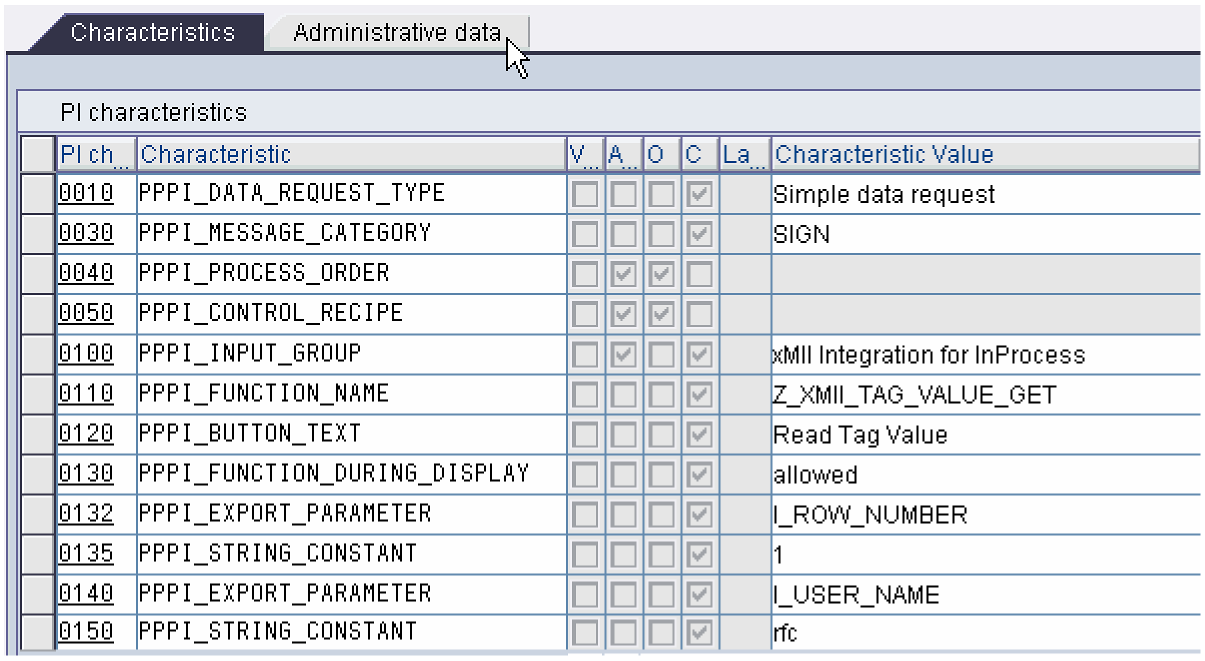
Please note that output XML variables will be handled as String elements, so you will have to parse in your own code. Please note as well that for xMII 11.5SP2 and lower you should prefer declaring XML transaction output parameters as parameters of type String and map your XML accordingly.

Image servlet

An easy way to launch the visualization of an xMII query to produce a simple html page or an image which can be displayed in a browser, is to use the /Lighthammer/ChartServlet. The process is very similar to the one described in the section on retrieving query data via HTTP. The set of URL parameters (like image size, html or image output) is described in the xMII query help. After construction ghe URL you can launch the browser window by calling e.g the function module ‘CALL_BROWSER’ using this URL string.

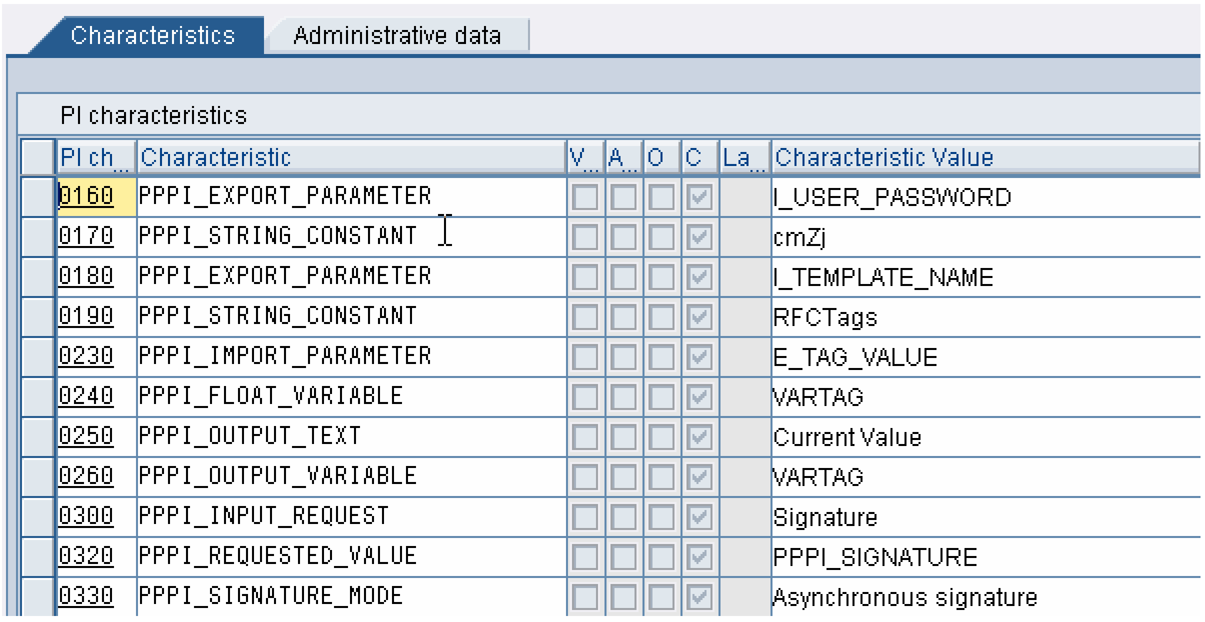
Usage in PP/PI sheets

In order to use xMII query data in a PP/PI sheet we construct a function module, which will be called via PP/PI characteristics in a process instruction. The input of the function module consists of al the data needed to retrieve data from xMII in one of the before mentioned ways. The output will contain the query results needed for display. A typical sequence to call a function module would look like:



PI characteristics

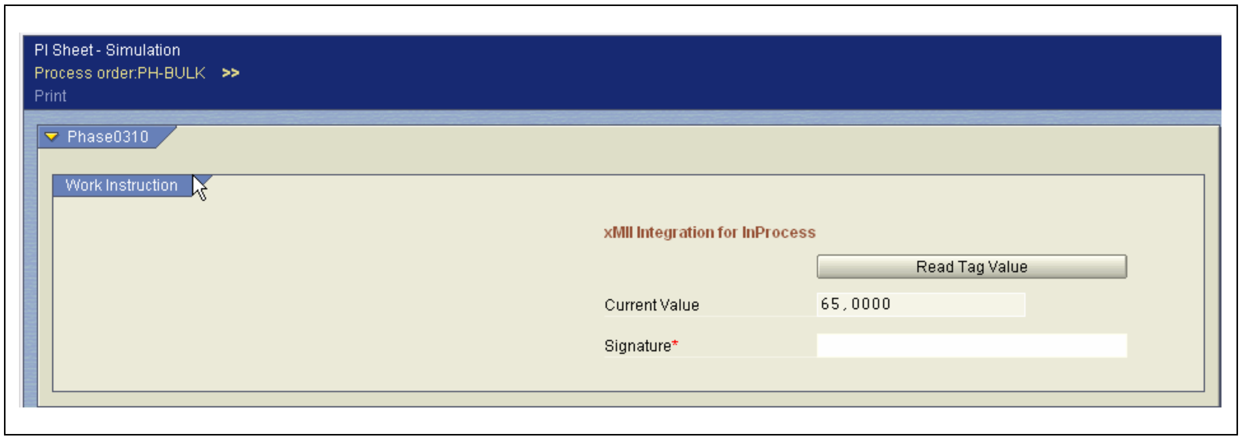
PI ch...	Characteristic	V...	A...	O	C	La...	Characteristic Value
0010	PPPI_DATA_REQUEST_TYPE				✓		Simple data request
0030	PPPI_MESSAGE_CATEGORY				✓		SIGN
0040	PPPI_PROCESS_ORDER		✓	✓			
0050	PPPI_CONTROL_RECIPE		✓	✓			
0100	PPPI_INPUT_GROUP		✓		✓		xMII Integration for InProcess
0110	PPPI_FUNCTION_NAME				✓		Z_XMII_TAG_VALUE_GET
0120	PPPI_BUTTON_TEXT				✓		Read Tag Value
0130	PPPI_FUNCTION_DURING_DISPLAY				✓		allowed
0132	PPPI_EXPORT_PARAMETER				✓		I_ROW_NUMBER
0135	PPPI_STRING_CONSTANT				✓		1
0140	PPPI_EXPORT_PARAMETER				✓		I_USER_NAME
0150	PPPI_STRING_CONSTANT				✓		rfc



PI characteristics

PI ch...	Characteristic	V...	A...	O	C	La...	Characteristic Value
0160	PPPI_EXPORT_PARAMETER				✓		I_USER_PASSWORD
0170	PPPI_STRING_CONSTANT				✓		cmZj
0180	PPPI_EXPORT_PARAMETER				✓		I_TEMPLATE_NAME
0190	PPPI_STRING_CONSTANT				✓		RFCTags
0230	PPPI_IMPORT_PARAMETER				✓		E_TAG_VALUE
0240	PPPI_FLOAT_VARIABLE				✓		VARTAG
0250	PPPI_OUTPUT_TEXT				✓		Current Value
0260	PPPI_OUTPUT_VARIABLE				✓		VARTAG
0300	PPPI_INPUT_REQUEST				✓		Signature
0320	PPPI_REQUESTED_VALUE				✓		PPPI_SIGNATURE
0330	PPPI_SIGNATURE_MODE				✓		Asynchronous signature

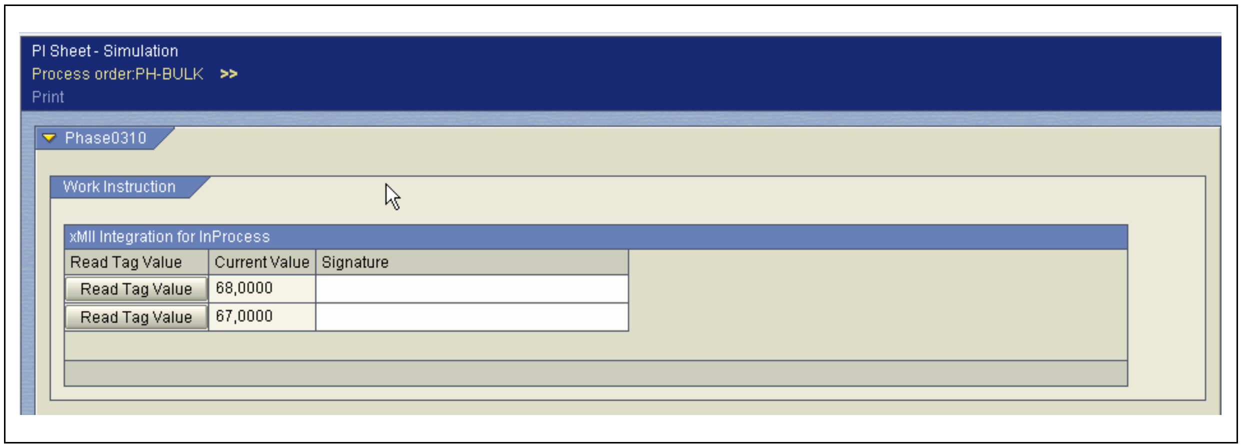
This example calls a function named 'Z_XMII_TAG_VALUE_GET' and provides logon data, the name of the query template and expects a tag value in the return variable E_TAG_VALUE which is mapped to the internal variable 'VARTAG'. The value of this variable is displayed in the PI sheet in a flied named 'Current Value':



Current Value 65,0000

Signature*

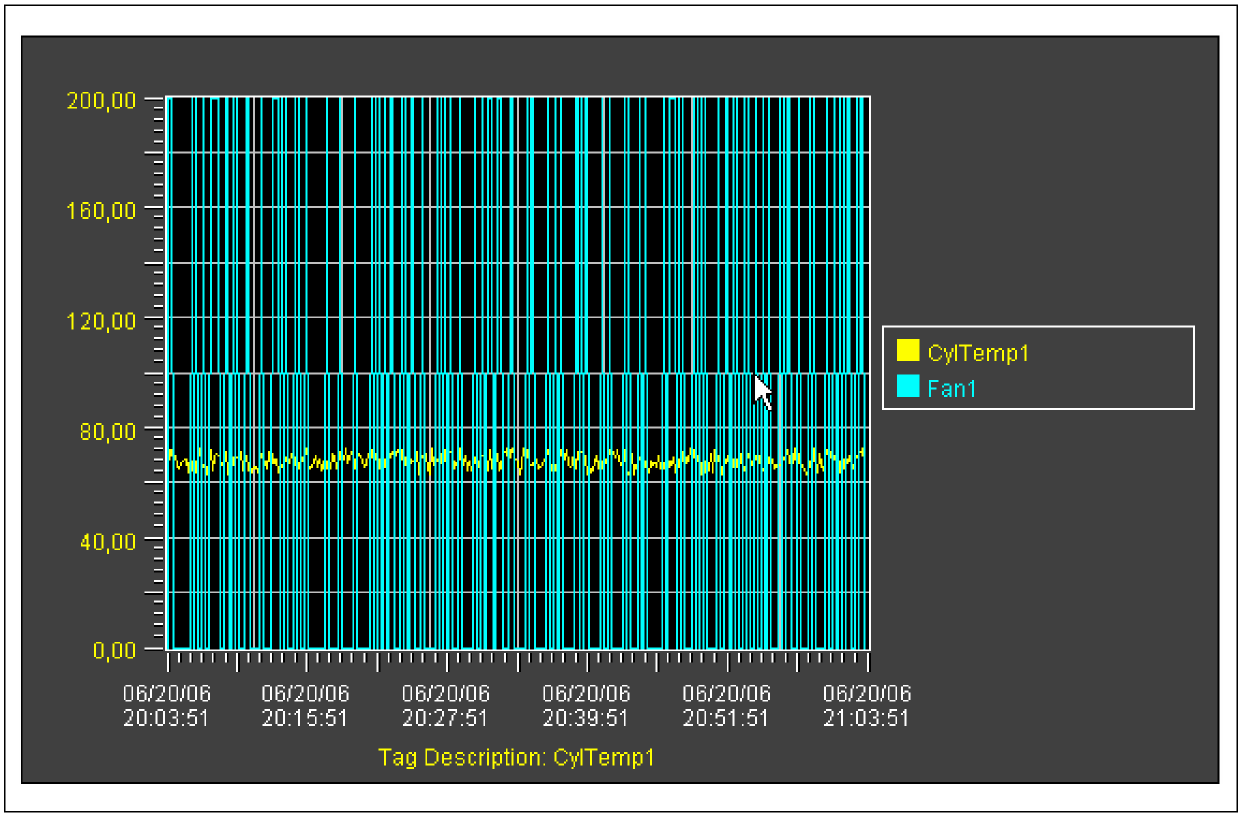
Note: you can use the same principle for any other display method, e.g. for getting a sequence of tag values in a table:



xMII Integration for InProcess

Read Tag Value	Current Value	Signature
Read Tag Value	68,0000	
Read Tag Value	67,0000	

If you want to display xMII charts you can use the same method by calling a function which will trigger the graphical display using the chart servlet in xMII:


200,00
160,00
120,00
80,00
40,00
0,00
06/20/06 20:03:51
06/20/06 20:15:51
06/20/06 20:27:51
06/20/06 20:39:51
06/20/06 20:51:51
06/20/06 21:03:51
Tag Description: CylTemp1
🟨 CylTemp1
🟦 Fan1

Copyright