Chapter 5. B2B and Security

In many cases SAP Process Integration is not only used for A2A (application 2 application) interfaces, but also B2B (business 2 business) related scenarios. Some of SAP PI functionalities were especially designed with this in mind, for example Party object used in Integration Directory or Virtual Receiver, both shown in this chapter. Additionally, security is one of the aspects we all need to take into consideration when building interfaces. Security has many aspects, ranging from controlling access, to message display, to using secure means of sending/receiving messages (as with FTPS adapter) and using secure storage functionality. All of these important topics are covered in this chapter, which makes it a significant chapter to review.

5.1 SXMB_MONI - controlling access to message display

Introduction

This article shows how you can restrict access to message display for particular interface/namespace/service etc. Note: this article is not for XI Basis users, but rather for XI developers who need to understand how message display works with the SAP standard authorization concept.

Article

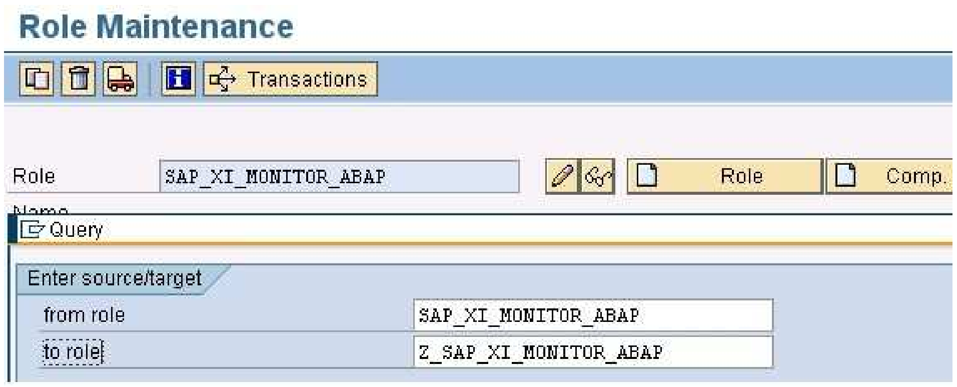
There have been several posts on the XI forum in which users had problems or questions concerning how we can control access to message display. This article will describe how you can use the authorization concept to restrict access to message payload by using standard authorization object.

Let's start at the beginning:

From **SAP Note 742324 - New authorization concept**

We learn that in order to use authorization concept for XI message monitoring, we should use S_XMB_MONI authorization object instead of S_XMB_DSP.

As a start, we can copy standard role SAP_XI_MONITOR_ABAP for displaying XML messages (TCODE - PFCG) (Figure 132):



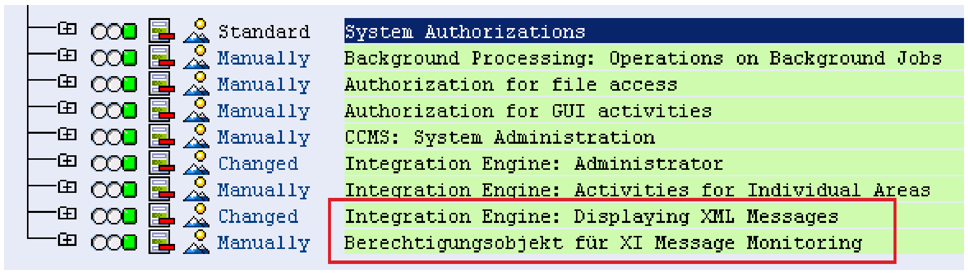
Role: SAP_XI_MONITOR_ABAP

from role: SAP_XI_MONITOR_ABAP

to role: Z_SAP_XI_MONITOR_ABAP

Figure 132

Now we can delete both objects for XML message display (Figure 133):



```
Standard   System Authorizations
Manually   Background Processing: Operations on Background Jobs
Manually   Authorization for file access
Manually   Authorization for GUI activities
Manually   CCMS: System Administration
Changed    Integration Engine: Administrator
Manually   Integration Engine: Activities for Individual Areas
Changed    Integration Engine: Displaying XML Messages
Manually   Berechtigungsobjekt für XI Message Monitoring
```

Figure 133

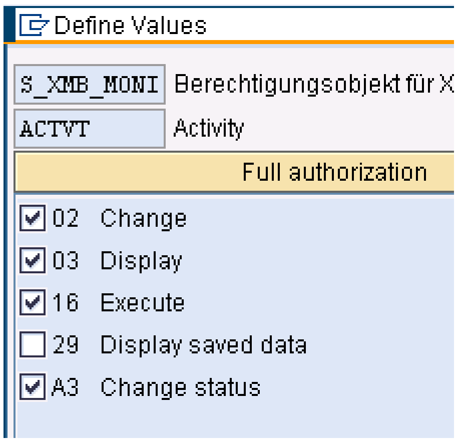
Once we delete these objects we can manually add object S_XMB_MONI. (Figure 134)

In this exercise, we'll try to give full access to the following activities (descriptions from object's S_XMB_MONI documentation):

- **02** = Change the SOAP header/body of an XML message
- **03** = Display the SOAP header/body of an XML message
- **16** = Reschedule failed XML message
- **A3** = Change the status of an XML message manually

Access will be restricted to:

- **29** = Display the payload of a XML message


Define Values

S_XMB_MONI	Berechtigungsobjekt für X
ACTVT	Activity

Full authorization

- ✓ 02 Change
- ✓ 03 Display
- ✓ 16 Execute
- ☐ 29 Display saved data
- ✓ A3 Change status

Figure 134

We can also tell exactly which interface/service/namespace can be used:

- **SXMBPARTY**= Communication Party
- **SXMBPRTAG**= Issuing Agency

- **SXMBPRTTYP**= Identification Scheme
- **SXMBSERV**= Service
- **SXMBIFNS** = Interface Namespace

Once we fill these we can generate the role and add it to one of our users.

The user will now be able to use SXMB_MONI, but when trying to see the message payload will see (Figure 135):

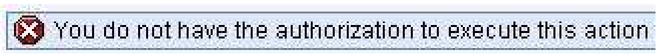

You do not have the authorization to execute this action

Figure 135

You can use standard report: **RSUSR070** to see which roles have S_XMB_MONI object assigned.

We trust this article will help you understand how message display can be restricted to some non basis XI developers.

<u>End Notes</u>

There needs to be clear distinction between access to message display and access to display/view parts of the message. The approach shown in this article allows you only to restrict access to display of a whole message. Sometimes it is necessary to be able to view the message and only hide certain parts of it (such as credit card information, payments, etc.), but in standard this is not supported by SAP PI/XI. In this case I would advise the use of encryption algorithms which can be used to encrypt the data before it reaches the middleware and encrypts the data at the receiving application.

5.2 Secure Storage Service with User Defined Functions

Introduction

This article will demonstrate how you can use secure storage service from your user defined functions with Exchange Infrastructure.

Article

Can you or should you store passwords in user defined functions? (For example, for JCO calls). The answer is never, for the following reasons:

1) you will need to change the password when your landscape changes (from DEV to QAT to PRD).
2) anyone with access to your mapping will be able to see the password if you store it in plain text.
3) you may need to change the password from time to time.

So is there an alternative? Perhaps a standard one? **Secure Storage Service** is one of the options. It is a place where you can store your passwords.

Step 1

Import necessary libraries as External Archives into your Integration Repository:

tc_sec_securestorage_service.jar

which can be found in:
\j2ee\cluster\server0\bin\services\tc~sec~securestorage~service

and **frame.jar**

which can be found in: \j2ee\cluster\server0\bin\system

Step 2

Create a user defined function in your Message Mapping with the code as shown below:

```
String myObjectRet = new String();
Object o;
try { javax.naming.Context
ctx = new javax.naming.InitialContext();
o = (Object)
ctx.lookup("tc~sec~securestorage~service");
SecureStorageRuntimeInterface secStore =
(SecureStorageRuntimeInterface) o; //Continue with implementation
RemoteSecureStorageClientContextInterface myContext =
secStore.getSecureStorageClientContext(); //Store an object
//remove the comment below below if you want to Store an object (this
is only for blog purposes - you should't set the pass in the same UDF -
you should only restore it in UDF //myContext.storeObject(new
String("myPassword"), "pass"); //Retrieve an object String myObject =
(String) myContext.retrieveObject("pass"); myObjectRet = myObject; }
catch (RemoteException e1) { // TODO Auto-generated catch block //
e1.printStackTrace();
//remove the comment below below if you want to Store an object // }
catch (ObjectStorageException e1) { // TODO Auto-generated catch block
} catch (ObjectRetrievalException e1) { // TODO Auto-generated catch
block } catch (NamingException e1) { // TODO Auto-generated catch
block
}
return myObjectRet;
```