How to call an SAP Enterprise Service from Microsoft Excel

Applies to:
SAP Systems Enterprise Service enabled: like ERP 2006, SAP SCM 5.1 and many others
Microsoft Office 2003 products

Summary
This article contains a simple example on how SAP Enterprise Service consumption can be done by Microsoft Excel 2003.

Authors: Igor Becker and John Miller
Company: SAP AG
Created on: 28 November 2007

Author Bio
Igor Becker is with SAP since 2004. He studied economics and engineering in Karlsruhe. After working for an IT-Consulting company he joined SAP software development for topics like Demand Planning, Enterprise Services, Duet, Auto ID and xMII.

John Miller has worked for SAP since 2005. He studied physics, mathematics, and computer science in the USA and worked for a startup company developing solutions based on OLAP and Excel before it was purchased by Microsoft. At SAP, he has worked in several areas and many teams. He currently works on the Duet design, POC, and prototype team.
# Table of Contents

- **How to Begin** ................................................................................................................................................. 3
- **Create a Project** ............................................................................................................................................ 3
- **What is your Kind of Data** ............................................................................................................................. 5
- **Where is the Enterprise Service** ................................................................................................................... 6
- **Create a Web Reference** .............................................................................................................................. 9
  - **Workaround for Namespaces** .................................................................................................................... 9
- **Create the COM Add-In** .............................................................................................................................. 10
  - **Programming the Add-In behavior** .......................................................................................................... 10
  - **Enable the COM Add-In** .......................................................................................................................... 11
- **Implement the DataProvider** ....................................................................................................................... 11
- **Implement the Data Placement Component** ............................................................................................... 13
- **Related Content** .......................................................................................................................................... 14
- **Appendix** ..................................................................................................................................................... 14
  - **COM add-in registration** .......................................................................................................................... 14
  - **Class DataProvider.cs** ............................................................................................................................ 14
  - **Class DataPlacementEngine.cs** .............................................................................................................. 16
- **Copyright** ..................................................................................................................................................... 21
How to Begin

You maybe know the situation, in which you are asked to create a prototype. In most cases you are asked 2 questions:

- Is it feasible?
- When can it be ready for presentation?

This article will help you to answer both of these questions if you are asked to create a basic Excel prototype consuming an SAP Enterprise Service. In order to keep this guide brief and focused on the topic of building a service consumer, we assume that the Enterprise Service is implemented, published, and accessible by HTTP.

Create a Project

To consume an Enterprise Service from Microsoft Office there are many different possibilities. The options which we considered were:

- Microsoft Office WS Toolkit.
- Shared Add-In implemented in C#
- Visual Studio Tools for Office (VSTO)

The first option, the SOAP Toolkit, was released in 2000. We decided to select from the two other options, which leverage the capabilities of the .Net platform. We chose to implement the Shared Add-In project since it is both simple and included in Visual Studio by default (VSTO is a separate package that everyone may not have access to). There are many considerations to take into account when selecting your approach which fall outside of this guide. These include things such as: deployment, security, maintenance, etc.

So when the decision was done, we started Visual Studio 2005 and created a project of type Shared Add-In. We simply followed the wizard steps. When asked for application host type we marked the Excel only.
For loading options, we did not select the Add-In to load each time Excel starts, which means our Add-In will have to be started manually. We also did not make the Add-In available for all users on the machine, which means our add-in will be available only to the user who installs it. Page 4 of the wizard gives us simple checkboxes to choose which features we want:

- [ ] I would like my Add-in to load when the host application loads.
- [ ] My Add-in should be available to all users of the computer it was installed on, not just the person who installs it.

After pressing “Finish” on the wizard window Visual Studio creates the project files. A new C# project is created with a default Connect.cs file containing a class that implements the necessary interfaces to be loaded as an add-in in Excel. This class contains many comments and describes what will happen when the application is loaded. The next goal is to locate the service provider and acquire the service definition file (we expected it defined in WSDL).
What is your Kind of Data

The web service we wanted to consume was the reading of SAP SCM Demand Planning (called later Demand Planning) figures. To understand what kind of data we are working with, here a snapshot of the planning table how it is used in the SAP system (SAP transaction /sapapo/sdp94):

Here it is easy to see, that working with these figures in Excel could makes sense and the prototype has its right to exist ;)

Where is the Enterprise Service

It was not difficult for us to find the Enterprise Service, which exposes the planning data, because one of our foregoing projects was the Enterprise Service enablement of Demand Planning. If you do not have this kind of close connection to the SAP application, ensure the service you need exists and provides sufficient data for your prototype. The services released for the SOAP runtime can be reviewed in the transaction wsadmin:

To get the WSDL definition of the web service mark the entry you are interested in and click on the icon. After logon on the next screen, you get to the screen with the link to WSDL file, which has to be saved locally.
The WSDL description of our web service looks like this:
Now the WSDL file is saved locally and we can switch to our .Net project to create a reference class based on this web service definition.
Create a Web Reference

In Visual Studio click on Project -> Add Web Reference and in the input field for URL paste the local path to the WSDL file you saved in the preceding step and click on “Go”. If everything was ok with the definition the service and its methods are recognized and you can see a similar picture to this one:

Workaround for Namespaces

After you click on “Add Reference” the Reference.cs class is created. Be careful! At least in our examples, we could not get the generated class to work due to a namespace problem. We could not determine if the main reason was the WSDL defining namespaces and attributes incorrectly or the generation of the C# code by Visual Studio 2005; however, there was evidence on many newsgroups and websites that this may be a known issue with the automated class generator in Visual Studio 2005. The time was short for building a prototype, so we manually fixed the generated class by simply deleting the namespace value.

From the original which looks like this:

```csharp
[System.Xml.Serialization.XmlElementAttribute(
    Form = System.Xml.Schema.XmlSchemaForm.Qualified,
    Namespace = "http://sap.com/xi/APO/Global")]
```

Making it look like:

```csharp
[System.Xml.Serialization.XmlElementAttribute(
    Form = System.Xml.Schema.XmlSchemaForm.Qualified,
    Namespace = "")]
```
Another approach (which also works) was to set the value of Form to System.Xml.Schema.XmlSchemaForm.Unqualified.

At least for a prototype developer (who is the target group of this article) this workaround should be sufficient. When working on a deliverable solution, please ensure that you overcome this issue with an approved and well tested means.

Now you are ready to enable your COM Add-In to make it visible in Excel.

**Create the COM Add-In**

In the next step you can add the button “MyButton” to the Excel Toolbar.

**Programming the Add-In behavior**

We used the Microsoft support article [http://support.microsoft.com/default.aspx?scid=kb;en-us;Q302901](http://support.microsoft.com/default.aspx?scid=kb;en-us;Q302901) as a reference to create the button “MyButton” when the Add-In is connected. Please refer to this article to modify the method OnStartupComplete of the class Connect.cs. We specifically create a command bar, a button, and the event handler for the button’s click event.

The code implementing the button looks like:

```csharp
try {
    MyButton = (CommandBarButton)oStandardBar.Controls["My Custom Button"];
} catch (Exception) {
    MyButton = (CommandBarButton)oStandardBar.Controls.Add(1, omissing, omissing, omissing, omissing);
    MyButton.Caption = "Read Demand Plan";
    MyButton.Style = MsoButtonStyle.msoButtonCaption;
}
```

To see how the CommandBar is defined please refer to the article mentioned above. There is also some explanation on how the Add-In works in general.
Enable the COM Add-In

To enable the switching our Add-In on and off we need to enable the menu item dealing with the COM Add-Ins. The following page explains how to do this:
http://www.cpearson.com/excel/AddingCOMAddinsMenuItem.aspx

If you have done everything correctly, the button “COM Add-Ins…” will show up on the Excel toolbar:

After clicking on it, you should see your COM Add-In (in our example it was called ESACall), which can be switched on now or switched off if not necessary any more. Please refer to the section ”COM add-in registration” in the appendix to see how to check the registry entry in case something does not work as you would expect it.

The button with caption “Read Demand Plan” (see the button definition above) is then displayed right beside the clicked button. Now we need to handle the event, which is fired when this button is clicked in order to retrieve data from SAP system, means to call the Enterprise Service.

Implement the DataProvider

The snippet, which is executed, when the button “Read Demand Plan” is clicked, is the following:

```java
// retrieve Demand Plan
DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan data = null;
try
{
    DataProvider dataProvider = new DataProvider();
    data = dataProvider.retrieveDP();
}
catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show("Error when calling ESA service:");
    + ex.Message);
}
The DataProvider class (see appendix, page 14 for full implementation) provides the correct inputs to call the Enterprise Service. There are 2 main steps to making the call: (1) setting the credentials for security (for a prototype it was sufficient to have a test user with login / password combination that we could use) and (2) filling in the input parameters defined by the generated web reference.

In this stage of the development, you should be able to compile the code and execute the web service call. In case there are deserialization problems on the SAP system side check the XML you sent and make sure all mandatory parameters were set correctly in your query. If you can get a meaningful error message from the SAP system telling you anything that has to do with the business application (in our case error messages from Demand Planning), then check the semantics of your parameters. In our example, we were not completely sure whether the date fields sent to backend were transformed correctly so we recorded our service call using the transaction sicf:

In transaction sicf, go to Virtual Hosts / Services. In the tree navigate through sap->bc->srt->xip->.. your service endpoint. Mark the service name and go to Edit-> Recorder -> Activate Recording. The screen similar to this one appears:

Activate Recording

When recording is enabled you can view it later by selecting Edit-> Recorder -> Display Recording. Check the completeness and semantic correctness of your request coming from the Excel Add-In and also get a general idea on how the system responds. This knowledge about the system response will help you to write the data into an Excel Worksheet which we cover next in this guide.

If the error is still hard to locate you can resubmit the request directly from the recordings view or by using the transaction sproxy (here the soap envelope must be deleted first to submit only the content). In case you set an external breakpoint in the proxy class you are able to debug the complete implementation of the called service.
Implement the Data Placement Component

When the data object of type DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan is back, we need to place the data in the Excel worksheet. We created a class specifically for this purpose and called it the DataPlacementEngine (s. Appendix, page 16).

```csharp
// display in EXCEL
try {
    DataPlacementEngine mapper = new DataPlacementEngine();
    mapper.MapData(data, applicationObject);
} catch (Exception ex) {
    System.Windows.Forms.MessageBox.Show("Error when mapping data:" + ex.Message);
}
```

The DataPlacementEngine (see appendix for implementation) loops over the complex response object lines and builds an Excel table of figures. After the table is complete, we can do any additional formatting to make the data more readable.

If you compare this table to the planning table taken from the SAP SCM Demand Planning, we can get a first impression of how this information can be brought to Microsoft Excel.
Related Content
- [http://www.cpearson.com/excel/AddingCOMAddinsMenuitem.aspx](http://www.cpearson.com/excel/AddingCOMAddinsMenuitem.aspx)
- [http://help.sap.com/content/documentation/esoa/index.htm](http://help.sap.com/content/documentation/esoa/index.htm)

Appendix

COM add-in registration
In addition to normal COM registration, a COM add-in needs to register itself with each Office application in which it runs. To register itself with a particular application, the add-in should create a subkey, using its ProgID as the name for the key, under the following location:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\<OfficeApp>\Addins\<ProgID>
```

The add-in can provide values at this key location for both a friendly display name and a full description. In addition, the add-in should specify its desired load behavior by using a DWORD value that is named LoadBehavior. This value determines how the add-in is loaded by the host application, and is made up of a combination of the following values:

- 0 = Disconnect - Is not loaded.
- 1 = Connected - Is loaded.
- 2 = Bootload - Load on application startup.
- 8 = DemandLoad - Load only when requested by user.
- 16 = ConnectFirstTime - Load only once (on next startup).

* The typical value specified is 0x03 (Connected | Bootload).


Class DataProvider.cs

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using ESACallAddIn.WebReferenceToDPFigures;
namespace ESACallAddIn
{
    class DataProvider
    {
        # region constants
        private static string LOGIN = "<Here your login>";
        private static string PASSW = "<Here your password>";
        private static string DOMAIN = "";
        # endregion

        System.Net.NetworkCredential userDefined = null;

        public DataProvider() {
            userDefined = new System.Net.NetworkCredential(LOGIN, PASSW, DOMAIN);
        }
    }
}
```
public DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan retrieveDP()
{
    // instantiate service object
    SAPAPODP_PlanKeyFigure001QRSERVICE service
        = new SAPAPODP_PlanKeyFigure001QRSERVICE();

    // set service call credentials
    service.Credentials = userDefined;

    // message header -> fill the message header here

    // query -> fill the input parameters here

    // log -> create a log object here
    Log log = new Log();

    // execute
    DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan result
        = DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan
            service.DemandPlanKeyFigureValueByElementsQueryResponse_In
            (ref messageHeader, query, out log);
    return result;
}
Class DataPlacementEngine.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using XL = Microsoft.Office.Interop.Excel; // generated proxy
using ESACallAddIn.WebReferenceToDP Figures;

namespace ESACallAddIn
{
    class DataPlacementEngine
    {
        public DataPlacementEngine()
        {
        }

        public void MapData(object data, object applicationObject)
        {
            // get the references to the Excel object model
            XL.Application app = (XL.Application)applicationObject;
            XL.Worksheet sheet = (XL.Worksheet)app.ActiveSheet;
            XL.Range range = (XL.Range)sheet.Cells[1, 1];

            # region Demand Plan
            int overallRows = 0;
            int overallCols = 0;

            IEnumerator planlevels =
                ((DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPla
                n)data).PlanningLevel.GetEnumerator();

            int level = 0;
            // for each plan level
            while (planlevels.MoveNext())
            {
                // first (header) row
                if (level == 0)
                {
                    range.Value2 = "Key Figure Name";
                    range = range.get_Offset(0, 1);
                }
                else
                {
                }
// characteristics titles
range.Value2 = "Developer Name";

// (((DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanPlanningLevel)planlevels.Current).Characteristic[level-1].DemandPlanCharacteristicID;
    range = range.get_Offset(0, 1);
}  
    level++;
}
overallCols = level;

// time periods
IEnumerator periods = ((DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan)data).TimeSeriesPeriod.GetEnumerator();
while (periods.MoveNext()){  
    range.Value2 = ((DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanTimeSeriesPeriod)periods.Current).Description;
    range = range.get_Offset(0, 1);
    overallCols++;
}
overallRows++;

// next row
range = range.get_Offset(1, -overallCols);
// total row
planlevels.Reset();

int kf = 0;
IEnumerator keyfigures = (((DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlan)data).PlanningLevel[0]).CharacteristicValueCombination[0].KeyFigure GetEnumerator();

bool otherLevelsDone = true;
while (keyfigures.MoveNext())
{
    planlevels.Reset();
    while (planlevels.MoveNext())
    {

}
DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanPlanningLevel currentLevel =

(DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanPlanningLevel)planlevels.Current;

// total row
if ((currentLevel.OrdinalNumberValue == 1)
    && (otherLevelsDone))
{
    switch (currentLevel.CharacteristicValueCombination[0].KeyFigure[kf].DemandPlanKeyFigureID)
    {
    case "9ACASUP":
        range.Value2 = "Available Capacity";
        break;

    case "9ACAP_MAX":
        range.Value2 = "Maximum Capacity";
        break;
    case "9ACAUSE":
        range.Value2 = "Capacity Used";
        break;
    }
    range = range.get_Offset(0, 1);
    range.Value2 = "Total";
    range = range.get_Offset(0, 1);
    int v = 0;
    IEnumerator values =
    currentLevel.CharacteristicValueCombination[0].KeyFigure[kf].Value.GetEnumerator();
    while (values.MoveNext())
    {
        range.Value2 =
        currentLevel.CharacteristicValueCombination[0].KeyFigure[kf].Value[v].Value;
        range = range.get_Offset(0, 1);
        v++;
    }
    overallRows++;
    range = range.get_Offset(1, -overallCols);
otherLevelsDone = false;
}
// first drill down
else {

    Enumerator cvc =
        currentLevel.CharacteristicValueCombination.GetEnumerator();

    string currentChar =
        currentLevel.Characteristic[0].DemandPlanCharacteristicID;

    // do not fill the first cell
    range = range.get_Offset(0, 1);

    int c = 0;
    while (cvc.MoveNext())
    {
        // display the current value of characteristic in this level
        DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanPlanningLevelCharacteristicValueCombination
            currentCVC
            =
                (DemandPlanKeyFigureValueByElementsResponseMessage_syncDemandPlanPlanningLevelCharacteristicValueCombination)cvc.Current;

        if (currentCVC.CharacteristicValue[0].DemandPlanCharacteristicID == currentChar)
        {
            range.Value2 =
                currentCVC.CharacteristicValue[0].DemandPlanCharacteristicValue;
            range = range.get_Offset(0, 1);
        }

        // loop over all values in this level for this cvc, this keyfigure
        int v = 0;
        IEnumerator values =
            currentLevel.CharacteristicValueCombination[c].KeyFigure[kf].Value.GetEnumerator();

        while (values.MoveNext())
```csharp
{ 
    range.Value2 = 
    currentLevel.CharacteristicValueCombination[c].KeyFigure[kf].Value[v].Value;
    range = range.get_Offset(0, 1);
    v++;
}
c++;
range = range.get_Offset(1, -overallCols + 1);
overallRows++;
}
otherLevelsDone = true;
range = range.get_Offset(0, -1);
}
}
kf++;
}

// format
XL.Range wholeRange = (XL.Range)sheet.Cells[overallRows, overallCols];
wholeRange.AutoFormat(XL.XlRangeAutoFormat.xlRangeAutoFormatClassic3, true, true, true, true, true, true);
range.EntireColumn.AutoFit();
range = range.get_Offset(0, 1);
range.EntireColumn.AutoFit();

# endregion
}
}
}```