Access to the SAP BW OLAP BAPI using C# and SAP .NET Connector 2.0

Hermann Däubler, Program Manager, Microsoft Corporation

Summary

The following paper should help to get a quick start on how to access the OLAP BAPI in SAP BW using the new SAP .Net Connector 2.0. The main focus is to demonstrate how easy it became to call the BAPI functions as SAP .NET Connector generates all the code which is necessary to handle the RFC (Remote Function Call) part. What's still left to the programmer is the connection to the BW system and processing BAPI result sets which of course might be still quite some work.

Applies to

- SAP BW 3.0B, Basis SP 39, ABAP SP 39, BW SP 21
- SAP .NET Connector 2.0
- Microsoft Visual Studio .NET 2003

Keywords

OLAP BAPI, SAP .NET Connector, C#

Level of difficulty

Technical consultants, Developers

Contact

For feedback or questions you can contact the Collaboration Technology Support Center at ctsc@sap.com. Please check the .NET interoperability area in the SAP Developer Network http://www.sdn.sap.com/sdn/developerareas/dotnet.sdn for any updates or further information.
Introduction

It might be useful in certain customer scenarios to extract data from SAP BW using a self-written program instead of a full-blown reporting tool. This paper shows how easy it is to call the SAP BW OLAP BAPI functions with the help of SAP .NET Connector 2.0. There are different options how data can be retrieved. Like with the XMLA interface it’s necessary to send a MDX (Multidimensional eXpressions) query and NOT SQL to get data out of SAP BW thru the OLAP BAPI. SAP BW infocubes are based on a relational star / snowflake schema. But thru the reporting APIs like XMLA or OLAP BAPI they are presented as multidimensional cubes to the outside world which is the reason for using MDX as the query language. The paper describes the usage of the function GetFlatData() to process the MDX query result.

The sample is based on a SAP demo cube which is part of the business content in SAP BW. Therefore it should be possible for every SAP BW 3.0B customer to setup the same environment which was used for the sample.
**SAP BW Demo Cube 0D_DECU**

To use exactly the same test environment which was used for the sample it’s necessary to activate the 0D_DECU demo cube.

**Activating demo cube 0D_DECU**

The cube can be activated in the administrator workbench (transaction RSA1) under section “Business Content”:

![Figure 1: Demo Cube Activation](image)

**Loading data into the demo cube 0D_DECU**

When the Demo Cube is activated you will find several .CSV load files in the SAP work directory. Use the appropriate Infosources and Infopackages to load the master data and the infocube (fact table)
SAP Note 370397 provides some hints regarding loading the SAP Demo cubes. There were two issues related to loading the .CSV files. A few files contained some additional character after the last data row which confused BW. A simple workaround was to open the .CSV file in Notepad and delete any additional ‘newline’ or whatever at the end (just use DEL or BACKSPACE until the cursor stands directly next to the last visible character in the data file).

The second issue was more difficult to solve. For whatever reason the order of the columns in two .CSV files (master data for 0D_SOLD_TO and trans data for the fact table) was different than the transfer structure which was generated. The solution was to change the order of the columns in the Datasource/Transfer Structure.

For 0D_SOLD_TO it has to be:
0D_SOLD_TO, 0D_COUNTRY, 0D_INDUSTRY, 0LONGITUDE, 0LATITUDE, 0ALTITUDE, 0SRCID, 0PRECISID, 0D_REGION, 0D_CITY, 0D_STREET, 0D_PHONE

For the fact table it has to be:
D_CO_CODE, D_DIS_CHAN, D_DIV, D_MATERIAL, D_PROD_HIE, D_SALE_EMP, D_SALE_ORG, D_SOLD_TO, D_VERSION, D_INV_QTY, SALES_UNIT, D_COST, DOC_CURRNCY, D_NETVLINV, D_TAXAMOUN, CALDAY, D_VTYPE
SAP BW OLAP BAPI

You will find infos about the OLAP BAPI on the Internet. Go to http://help.sap.com and search for “OLAP BAPI”. But first select “SAP NetWeaver” as the application. The BAPI provides methods to browse BW metadata and master data (MDataProviderBW) and methods to execute multi-dimensional result sets and fetch data (MDataSetBW):

Figure 3 : OLAP BAPI
Code samples

Use SAP BW OLAP BAPI methods to retrieve metadata

After you installed SAP .NET Connector open a new C# Console Program Project in Visual Studio .NET and add the SAP Connector Proxy (in Solution Explorer right-click on the project name and choose “Add”->“Add New Item …”). More infos can be found in the description and the samples which come with the SAP .NET Connector. After adding the Proxy click on “Build”->“Build Solution”.

Figure 4 : SAP Connector Proxy

Configure the connection to the BW application server. In the folder BOR you can use the subfolder “Alphabetical” to find the OLAP BAPI functions.
Figure 5: looking for OLAP BAPI methods

Under MDDataProviderBW choose GetCubes and put it via “drag&drop” on the design window of the generated SAPProxy. SAP .NET Connector generates code to call the BAPI function and to define the parameters which are needed. By “moving” the BAPI method to the design window it will automatically add icons for the additional classes which are needed to handle the method parameters and result sets. Before you continue click on “Build”->”Build Solution”.

SAP
Figure 6: choosing OLAP BAPI method “GetCubes”

In the “code-view” of the SAPProxy1.cs file you will find the BAPI call and which parameters are used:
The little C# sample consists mainly of three parts:

1. connect to SAP BW
2. call the OLAP BAPI function
3. process the result set

This is the code which needs to be written. All the BAPI classes were generated by SAP .NET Connector 2.0.
Source listing of C# sample 1 - retrieve metadata :

IMPORTANT : if you just copy the following code into Visual Studio you will get errors during compilation in case you chose a different name than SDNC_SAMPLE01 for your C# project. So just change the namespace in the source listing accordingly.

```csharp
using System;
using SAP.Connector;
	namespace SDNC_SAMPLE01
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>

class Class1
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
        int i;
        Destination bw_system = new Destination();

        // assemble connection string
        bw_system.AppServerHost = "hdmbw30b";
        bw_system.Username = "rosetta";
        bw_system.Password = "obelix";
        bw_system.Client = 900;
        bw_system.SystemNumber = 1;

        // establish connection to BW
        SAPProxy1 sap_bw_proxy = new SAPProxy1(bw_system.ConnectionString);

        // parameters for BAPI call
        BAPI6110CUBTable ds_cubtab = new BAPI6110CUBTable();
        BAPI6110CUB ds_cubrow;
        BAPIRET2 ds_ret2;

        // BAPI call to get existing cube names ( no search restrictions )
        sap_bw_proxy.Bapi_Mdprovider_Get_Cubes("", "", "", out ds_ret2, ref ds_cubtab);

        // process result set
        for (i = 0; i < ds_cubtab.Count; i++)
        {
            ds_cubrow = ds_cubtab[i];
            Console.WriteLine(ds_cubrow.Cube_Nam);
        }

        Console.WriteLine("\n\nPress RETURN");
        Console.ReadLine();
    }
}
```
The following screenshot shows the output of the C# program. It lists infocube names as well as BW queries which were released for OLE DB for OLAP (query properties in SAP BW Query Designer). Query names have the format `<infocube>/<query name>`. This means that BW queries look like infocubes when using the SAP BW OLAP BAPI. In this case sending a MDX query thru the OLAP BAPI will in fact trigger an existing BW query. Otherwise the MDX query will be finally transformed into SQL “on the fly” by the BW OLAP engine to access the selected infocube.

![Figure 8: output of C# sample 1 – retrieving BW metadata (existing cubes)](image)

**Use SAP BW OLAP BAPI methods to execute a MDX query**

The first sample to list all available cubes was very easy. The second one which should send a MDX query and show the result set in a command prompt window is more complicated. The MDX sample application which comes with MS Analysis Services was used to find a suitable MDX statement (see Limitations at the end of the document which explains the issue of writing an appropriate MDX command). After installing the
SAP BW OBDO (OLE DB for OLAP) driver it’s possible to connect directly to BW. In the connection window just enter the BW SID (here it was HTG) as the Server and the name of the ODBO driver “Mdrmsap” as the Provider. Then a SAP logon window should appear which asks for Client, User, Password and Language. There is an issue with the connection to SAP BW which can be ignored – sometimes an error message pops up: “Unexpected error when populating MDX functions collection. Invalid use of Null.”. Just ignore it. Click on “OK” and continue. Select the required cube and run some query examples.

Here we see a query which selects two dimensions on one axis and three measures on the other axis. The idea is to achieve a similar output with the C# program in a command prompt window:

Figure 9: BW access using the MDX sample application from MS Analysis Services
The formatting of the output should be flexible enough to handle a switch of “columns” and “rows” in the MDX query correctly. Now the values of 0D_CO_CODE and 0D_SOLD_TO are listed on the left side. As there are no fields in a command prompt window which can be re-sized with the mouse it’s necessary to cut some text strings which are too long. Otherwise the output doesn’t look nice. The measures won’t be modified as the numbers have to be correct and show all digits:
Figure 10: BW access using the MDX sample application from MS Analysis Services

Here are two screenshots which show how the output of the second C# sample looks like in a command prompt window using the same query as above with the MDX sample application:

Figure 11: output of C# sample 2 – sending a MDX query to SAP BW

Switch of “columns” and “rows”:
The procedure in Visual Studio .NET is exactly the same as before. But this time different BAPI functions are needed. Move the following OLAP BAPI functions by “drag&drop” on the design window of the SAPProxy (under MDDatasetBW):

CreateObject
GetAxisData
GetAxisInfo
GetFlatData

The following BAPI calls will be generated:

Bapi_Mddataset_Create_Object
Bapi_Mddataset_Get_Axis_Info
Bapi_Mddataset_Get_Axis_Data
Bapi_Mddataset_Get_Flat_Data

Before you continue click on "Build"->"Build Solution".
IMPORTANT: if you just copy the following code into Visual Studio you will get three errors during compilation because of “newline” in the string constants which represent the MDX query. Just remove them. A second issue might be the namespace in case you chose a different name than SDNC_SAMPLE02 for your C# project. So just change the namespace in the source listing accordingly.

```
using System;
using SAP.Connector;

namespace SDNC_SAMPLE02
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // variables for processing the result set and formatting on the screen
            int i;
            int ii;
            int pos;
            int nr_ax0_dims;
            int nr_ax1_dims;
            int nr_ax0_tuples;
            int res_group;
            int max_col_width;
            int first_col_width = 10;
            int ds_xl_row_nr;

            // parameters which are needed to call the BAPI functions
            string dataset_id;
            BAPIRET2 ds_ret2;

            BAPI6111MDXTable ds_mdxtab = new BAPI6111MDXTable();
            BAPI6111MDX ds_mdx_stmt;
            BAPI6111FSDTable ds_fsdhtab = new BAPI6111FSDTable();
            BAPI6111FSDTable ds_fsdttab = new BAPI6111FSDTable();
            BAPI6111FSD ds_fdsdrow;
            BAPI6111AXDTable ds_axdtab = new BAPI6111AXDTable();
            BAPI6111AXITable ds_axitab = new BAPI6111AXITable();
            BAPI6111AXI ds_axirow;
            BAPI6111AXLTable ds_axltab = new BAPI6111AXLTable();
        }
    }
}````
BAPI6111DPRTable ds_dprrtab = new BAPI6111DPRTable();
BAPI6111FCITable ds_fcitab = new BAPI6111FCITable();
BAPI6111MPPTable ds_x0_mpptab = new BAPI6111MPPTable();
BAPI6111MPP ds_x0_mpprow;
BAPI6111MPPTable ds_x1_mpptab = new BAPI6111MPPTable();
BAPI6111MPP ds_x1_mpprow;
BAPI6111OPKTable ds_opktab = new BAPI6111OPKTable();
BAPI6111OPVTable ds_opvtab = new BAPI6111OPVTable();

Destination bw_system = new Destination();

// assemble connection string
bw_system.AppServerHost = "hdmbw30b";
bw_system.Username = "rosetta";
bw_system.Password = "obelix";
bw_system.Client = 900;
bw_system.SystemNumber = 1;

// establish connection to BW
SAPProxy1 sap_bw_proxy = new SAPProxy1(bw_system.ConnectionString);

// specify a MDX query on the SAP Demo Cube. As the length of the
// field "Line" is limited to 75 characters it's necessary to split
// the statement into <n> parts. Every part will be added to the
// MDX "table".

ds_mdx_stmt = new BAPI6111MDX();
ds_mdx_stmt.Line = "select non empty
crossjoin({[0D_CO_CODE].[1000],[0D_CO_CODE].[2000]}) ,";
ds_mdxtab.Add(ds_mdx_stmt);

dmdxstmt = new BAPI6111MDX();
dmstmt.Line = "MBED_SOLD_TO]. {0000020100}:[0D_SOLD_TO]. {000002500}");
on columns,";
ds_mdxtab.Add(ds_mdx_stmt);

dmstmt = new BAPI6111MDX();
dmstmt.Line = "([Measures].[0D_COST],[Measures].[0D_NETVLINV],
[Measures].[0D_INV_QTY]) ";
ds_mdxtab.Add(ds_mdx_stmt);

dmstmt = new BAPI6111MDX();
dmstmt.Line = " on rows from [0D_DECU]");
ds_mdxtab.Add(ds_mdx_stmt);

// send the MDX statement and get a dataset_id back which is used
// as a parameter in the following BAPI calls.
sap_bw_proxy.Bapi_Mddataset_Create_Object(out dataset_id, out
ds_ret2, ref ds_mdxtab);

// get some meta data to figure out how many tuples and how many
// dimensions are on the two axis (rows and columns in the MDX statement)
sap_bw_proxy.Bapi_Mddataset_Get_Axis_Info(dataset_id, out ds_ret2,
  ref ds_axitab, ref ds_axitab, ref ds_axitab, ref ds_axitab,
  ref ds_axitab );
ds_axirow = ds_axitab[0];
nr_ax0_dims = System.Convert.ToInt32(ds_axirow.Dims);
nr_ax0_tuples = System.Convert.ToInt32(ds_axirow.Coordinates);

ds_axirow = ds_axitab[1];
nr_ax1_dims = System.Convert.ToInt32(ds_axirow.Dims);

// get the values/names of the dimensions which will be on the vertical axis of the output. This is necessary because the result set of the query doesn't include these values in all cases (e.g. "ALL ... ").
sap_bw_proxy.Bapi_Mddataset_Get_Axis_Data("1",dataset_id,"","",
  out ds_ret2, ref ds_x1_mpptab, ref ds_opktab, ref ds_opvtab );

// get the values/names of the dimensions which will be on the horizontal axis of the output. These values are not included in the result set of the query.
sap_bw_proxy.Bapi_Mddataset_Get_Axis_Data("0",dataset_id,"","",
  out ds_ret2, ref ds_x0_mpptab, ref ds_opktab, ref ds_opvtab );

// get the data of the MDX query result. This is a "flat" representation of the multidimensional result. Imagine the query result as a 2-dimensional picture like you see it in the MDX sample application which comes with Analysis Services. On one axis you see the different dimensions and on the other axis you see the measures. The flattened result set includes every value of all the rows. There is an official spec which describes the "flattening" algorithm:
//
//
// Example:
//
//Company1 Employee1 1 2
//Company1 Employee2 3 4
//Company2 Employee1 5 6
//Company2 Employee2 7 8
//
//In this example the "flat" result set would look like this:
//
//row 001 : Company1
//row 002 : Employee1
//row 003 : 1
//row 004 : 2
//row 005 : Company1
//row 006 : Employee2
//row 007 : 3
//row 008 : 4
//...
//
//What's missing are the names "Measure1" and "Measure2". That's why a separate BAPI call is necessary to get it.
But there are also cases where the vertical dimension values are missing (e.g. "All ...."). Therefore another BAPI call was necessary to get the complete list (see get_axis_data above).

The tricky part now was to find a simple way to get the 2-dimensional printout by going thru the result set in a simple for-loop. It worked with the sample query and a few variants (e.g. different number of dimensions or switching "rows" and "columns". But it's not guaranteed that it works in all cases. The sample code should just help to get a quick start on this topic.

```java
sap_bw_proxy.Bapi_Mddataset_Get_Flat_Data(dataset_id,"","", out ds_ret2, ref ds_fsddtab, ref ds_fsdhtab);
```

To figure out how many "elements" every row in the 2-dimensional picture has. This is needed to do the correct formatting (e.g. switch to a new line at the right position).

```java
res_group = nr_ax1_dims + nr_ax0_tuples;
```

To get a nicer output on the screen, go thru all measures and find the longest string. Skip the dimension values at the beginning. These will be cut after 'first_col_width' characters (originally set to 10).

```java
max_col_width = 0;
for( i=0; i < ds_fsddtab.Count; i++ )
{
    ds_fsddrow = ds_fsddtab[i];
    pos = i%res_group;
    if( pos >= nr_ax1_dims )
        if( ds_fsddrow.Data.Length > max_col_width )
            max_col_width = ds_fsddrow.Data.Length;
}
```

Print out the header info for the columns. The names can be found by calling the get_axis_data function (see BAPI call further up). For every dimension there will be one line.

```java
print out the header info for the columns. The names can be found by calling the get_axis_data function (see BAPI call further up). For every dimension there will be one line.
```

Let's take the example above and switch columns and rows:

```
<table>
<thead>
<tr>
<th>Company1</th>
<th>Company1</th>
<th>Company2</th>
<th>Company2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee1</td>
<td>Employee2</td>
<td>Employee1</td>
<td>Employee2</td>
</tr>
<tr>
<td>Measure1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Measure2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
```

The column header names also come in a "flattened" format:

```
<table>
<thead>
<tr>
<th>Company1</th>
<th>Company1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee1</td>
<td>Employee2</td>
</tr>
<tr>
<td>Company1</td>
<td>Company2</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
```

To get the correct output on the screen it's necessary to
// go thru the list, pick the right values ("Company", "Employee") and print them on the right position.

for( i=0; i < nr_ax0_dims; i++ )
{
    // just print spaces above the vertical dimensions.
    for( ii=0; ii < nr_ax1_dims; ii++ )
    {
        Console.Write("".PadRight(first_col_width,' '));
        Console.Write(" ");
    }
    Console.Write("|");
    // print the headers. If the names are too long - cut it.
    for( ii=i; ii < ds_x0_mpptab.Count; ii+=nr_ax0_dims )
    {
        ds_x0_mpprow = ds_x0_mpptab[ii];
        if( ds_x0_mpprow.Mem_Cap.Length > max_col_width )
            Console.Write(ds_x0_mpprow.Mem_Cap.Substring(0, max_col_width));
        else
            Console.Write(ds_x0_mpprow.Mem_Cap.PadRight(max_col_width,' '));
        Console.Write("|");
        Console.Write("\n");
    }
    // print a line of dashes below the column headers.
    for( ii=0; ii < nr_ax1_dims; ii++ )
    {
        Console.Write("".PadRight(first_col_width,'-'));
        Console.Write("-");
    }
    Console.Write("+");
    for( ii=0; ii < ds_x0_mpptab.Count; ii+=nr_ax0_dims )
    {
        Console.Write("".PadRight(max_col_width,'-'));
        Console.Write("+");
    }
    Console.Write("\n");
}

// go thru the "flattened" result set and print all the rows in a 2-dimensional format.

ds_x1_row_nr = 0;
for( i=0; i < ds_fsddtab.Count; i++ )
{
    ds_fsddrow = ds_fsddtab[i];
    pos = i%res_group;
    if( pos == 0 && i > 0 )
        Console.Write("\n");
    if( pos == nr_ax1_dims )
        Console.Write("|");
if( pos < nr_ax1_dims )
{
    // to get the correct values for the dimensions at the
    // beginning of every row use the meta data which was retrieved
    // with get_axis_data. Otherwise they are missing in some cases
    // ( e.g. "All ..." ).
    ds_x1_mpprow = ds_x1_mpptab[ds_x1_row_nr];
    if( ds_x1_mpprow.Mem_Cap.Length > first_col_width )
        Console.Write(ds_x1_mpprow.Mem_Cap.Substring(0,
            first_col_width));
    else
        Console.Write(ds_x1_mpprow.Mem_Cap.PadRight(
            first_col_width,' '));
    Console.Write(" ");
    ds_x1_row_nr++;
} else
{
    // to print the measures just take the Data field of the result
    // set.
    Console.Write(ds_fsddrow.Data.PadLeft(max_col_width,' '));
    Console.Write("|");
}
Console.WriteLine("\n\n\n\nPress RETURN");
Console.ReadLine();

} // end Main
} // end Class1
} // end namespace
Conclusion

SAP .NET Connector 2.0 is a great tool which relieves developers from all the Remote Function Call details in order to access the SAP BW BAPIs. What’s left are parameter definitions / settings and processing the result sets. Depending on the BAPI method this might be still some work. But it’s definitely an elegant an easy way to connect SAP BW with the Microsoft .NET world.

Limitations

Both C# samples should just help to get a quick start on the topic. They cannot be used unmodified for real application development ( e.g. no error handling included ). Both samples were tested with a simple infocube and a simple BW query. It’s not guaranteed that they work 100% correct with all types of info providers and all levels of complexity.

BEX Analyzer from SAP is not using MDX. Additionally BW queries are not stored in MDX format inside SAP BW. Because of this it’s not possible to just copy MDX statements from existing BW queries to use them in the C# program. MDX has either to be written manually or to be generated / created by a separate tool. One possibility would be the MDX sample application which comes with MS Analysis Services as mentioned in the paper.

The SAP BW OLAP BAPI is a reporting API. It’s not made for unloading masses of data. OpenHubService is the tool which should be used instead.

There is a difference between reporting and unloading. Imagine a BW query which selects certain data from an infocube. If this data would be stored in an external OLAP engine like Microsoft Analysis Services it’s no longer called reporting. Whenever you extract data from the mySAP BI environment into 3rd-party target systems your SAP BW Enterprise Data Warehouse functions as an ‘Open Hub Platform’. Since extracting data from SAP BW by any means requires additional licensing, please make sure to contact your SAP representative.
References

- OLAP BAPI: http://help.sap.com/saphelp_nw04/helpdata/en/64/9b8f3c855f0454e10000000a11405aframeset.htm