Extending the Project Server 2003 Portfolio Analysis Cube

A practical example using the Microsoft Project 2003 SDK

Overview, Objectives & Skill Requirements

Microsoft Project Server 2003 includes a wide range of features and capabilities. This paper will address the Portfolio Analyzer, providing an overview of how it works and how to make it do more.

My teaching objectives include:

- A basic understanding of the portfolio analysis cube and the process that creates it.
- Identification and use of the tools required to extend the cube.
- An introduction to the cube extension process using Microsoft's Task Earned Value Solution Starter.
- Using the Task Earned Value as a foundation for building your own custom cube.

While knowledge and understanding can benefit anyone who administers a Project Server, actually implementing a custom cube is going to require a specific set of skills. You or someone on your team will need the following skills:

- Project Server Implementation and Administration
- Visual Basic development using the Visual Studio 6.0 development tool set
- Microsoft SQL Server administration
    - Enterprise Manager
    - Query Analyzer
- T-SQL stored procedure development
    - Select
    - Insert

Clearly there are things that we just can't cover in this short presentation. Our list of non-objectives includes:

- How to install and configure a Project Server
- Using Visual Basic .Net or Visual Studio .Net
- Creating a development copy of your Project Server database

Tools, Configuration & Resources

The Project 2003 Software Development Kit (SDK) provides a tremendous wealth of resources for anyone with an interest in customizing or extending a Microsoft Project Server environment. You can download the whole SDK or just the Portfolio Analyzer OLAP Extensions Solution Starter on the following web site:

http://msdn.microsoft.com/office/understanding/project/tools/

This presentation builds the SDK foundation in an effort to clarify and expand the information available in one focused area of interest. While essential elements of the OLAP Extensions Solution Starter may be repeated or paraphrased in this presentation, **you are strongly encouraged to read through at least the Task Earned Value portions of the SDK before proceeding**.

The Task Earned Value and other Solution Starters in the Project 2003 SDK are written for use in a Visual Studio 6.0, Visual Basic 6.0 development environment. While it may be possible to use the newer Visual Studio .Net tools, this topic will *not* be addressed in this paper.

Doing development work on a production server is *never* a good idea. Building an OLAP cube can be resource intensive and can significantly [severely] impact performance. I strongly [passionately] recommend that you take the time to acquire and construct a development server before starting any of the code development activities presented in this paper. The career you save may be your own!

Your development server can be constructed on an older computer. It does not have to be an expensive modern system. I have used and recommend the following as a *minimum* system hardware configuration:

Pentium-III, 500 MHz processor
384 MB main memory
20 GB hard drive
Ethernet network interface

The following software components must be installed:

Microsoft Windows Server	(2000 or later, I use the 2003 Standard release)
SQL Server 2000	(again, I use the Standard release)
SQL Server 2000 Analysis Services	
Microsoft Project Server 2003	
Visual Basic & Visual Studio 6.0	
Project 2003 Software Development Kit (SDK)	
Microsoft Office Project Professional 2003	
The latest service packs for all of these products	

Installing and updating software on your development server will generally be easier if your system is connected to the Internet. If you connect your development system to network (Internet or local), I recommend that you do *Not configure a SMTP (mail) server* during (or after) the Project Server installation. This will ensure that your Project Server users do not receive duplicate or unwarranted notifications.

A Project Server database with enough content to test your new Portfolio Analyzer Cube is needed. Select and implement one of the following options:

- If you have already created projects and enterprise custom fields on your production server, you can backup and restore a copy of your project server database on the development system. While the process for completing this task is not difficult (and is a very useful skill), it is beyond the scope of this paper. If you use this option, be sure to de-configure your Project Server SMTP settings.

- Install and configure the sample database included with Project Server distribution CD.

- Create your enterprise custom fields on the new server along with at least two or three projects that use these fields.

Be sure to build the standard Portfolio Analyzer cube before proceeding. Trying to extend a broken cube is more trouble than anyone deserves.

Cube Basics, Structure & the Cube Build Process

Many people have been confused about what the SQL Server 2000 Analysis Services application is and how it is used with Project Server. Key points that you need to understand include:

- Analysis Services is first and foremost an On Line Analytical Processing (OLAP) tool. OLAP allows you to create and efficiently analyze a multidimensional data structure (Cube). Think of the Cube as a large N-dimensional matrix where the sum and total of every intersection has been calculated in advance.

- The Analysis Services applications stores data in a potentially large number of individual data files. These files are *not* part of (or stored within) a SQL database. Note that as a result, *your analysis cube is not backed up* when you backup your SQL database.

- A "Measure" is a ***numeric*** data field. Note that while dates are technically stored as numbers, they are not used as Measures.

- A "Dimension" is a field used to filter or categorize data (Measures). Each dimension must have an indexed list of values stored in a table. In the Project Server these are SQL database tables and generally referred to as "staging tables".

- To create an analysis Cube you must define a single Fact table that includes:
    1. Numeric data fields (Measures) you want to analyze
    2. Index fields for each Dimension you will use to filter or categorize your data.

- The Analysis Services application features a powerful GUI interface called the Analysis Manager. This tool can be used to manually create, manipulate and explore OLAP cubes. In addition to its other features, it is a wonderful tool for initial cube design and prototype construction.

> Please note that a Cube built and maintained by the Project Server application has no way of knowing about changes made elsewhere. If you use the Analysis Manager to alter or add to a Project Server cube, you should anticipate that your changes will disappear or be damaged the next time Project Server updates the cube.

Structure

Launch the SQL Enterprise Manager and look at the tables in your project server database. Observe that many of the table names begin with "MSP_CUBE". These are the staging tables that Project Server creates to support the Portfolio Analyzer cubes. Most of the MSP_CUBE tables are Dimension staging tables. The very few that end in _FACT are used to stage Measures.

The default cube used in Portfolio Analyzer (MSP_PORTFOLIO_ANALYZER) is actually a virtual cube. Virtual cubes are constructed entirely within Analysis Services by integrating other basic cubes. The two basic cubes staged within the Project Server database include the assignments fact cube (MSP_CUBE_ ASSN_FACT) and the resource availability fact cube (MSP_CUBE_RES_AVAIL_FACT).

Notice that there are 30 table names that start with MSP_CUBE_ENTERPRISE_ PROJECT_OUTLINE and 30 more that start with MSP_CUBE_ENTERPRISE_ RESOURCE_OUTLINE. These tables correspond to the enterprise project and resource outline fields in the Project Server database.

Let's explore a bit…

Right-click on one of the MSP_CUBE_ENTERPRISE_ PROJECT_OUTLINE tables and select `Return all rows` from the Open Table submenu. If you have previously defined values for the corresponding field in the Project Enterprise Global, you should see the values here. The PROJECT_OUTLINECODE_NAME field holds the value that will be displayed in the cube Dimension. The PROJECT_OUTLINECODE_ID field is the index value. Note that because an enterprise outline code field can have more than one level, the table also includes a second index field used to point to the parent entry in the table. Close the table.

Right-click on the table MSP_CUBE_PROJECTS and select `Return all rows` from the Open Table submenu. The index for this table is the PROJ_UID column. This is a short form for the project unique identifier or ProjectUniqueID and can be used to link data from several tables in different parts of the database to a specific (unique) project. Please note that the WPROJ_ID can not be used for this purpose. The MSP_CUBE_PROJECTS table holds project index data for a variety of dimensions. The PROJ_NAME field is used to create the Project List dimension.

Right-click on the table MSP_CUBE_ASSN_FACT and select `Return all rows` from the Open Table submenu. This table contains one line of data for each date, that each resource is assigned to work on each project in the project database. For example if a resource named Tom Jones is assigned five days of work on a task in Project-A, the project server will insert 5 lines in this table. You will find a data column in this table for each Measure included in the MSP_ASSN_FACT cube.

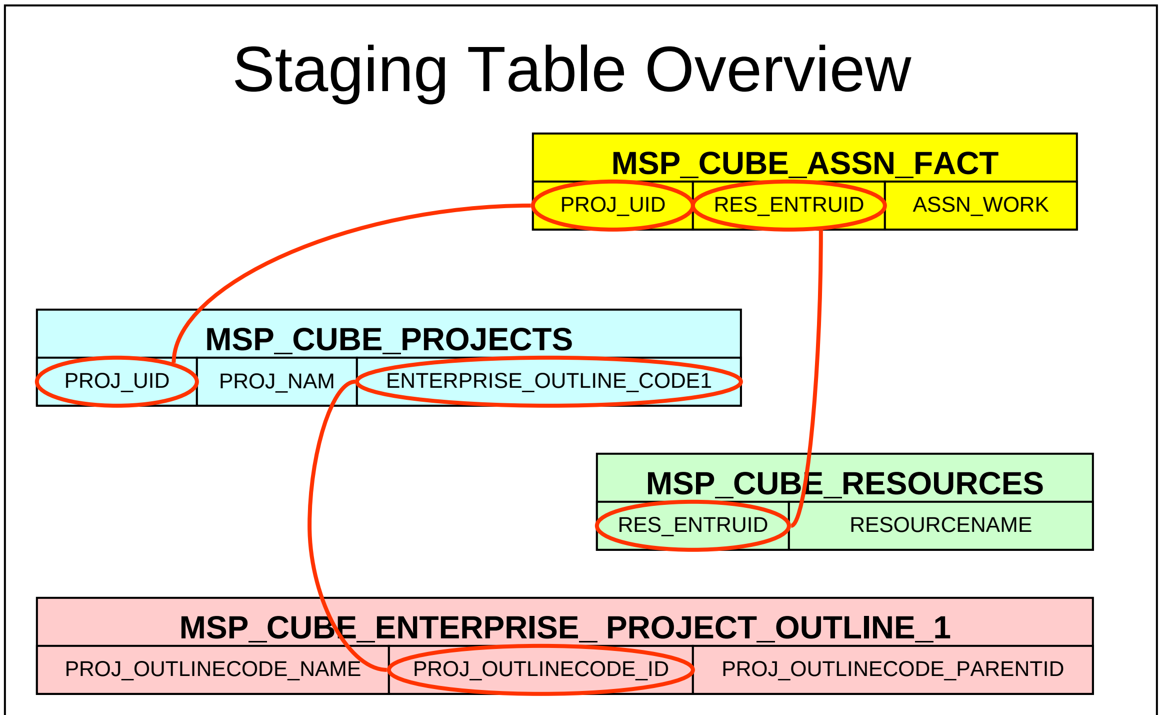

Figure 1

If you define values for the Enterprise Project Outline Code 1 field on your project server, the list of values will be stored in the MSP_CUBE_ENTERPRISE_ PROJECT_OUTLINE_1 table and the value you selected for each project will be stored as an index to this table in the ENTERPRISE_PROJECT _CODE1 column of the MSP_CUBE_PROJECTS table.

Build process

Project Server starts the cube build process when you select the `Update Now` option on the PWA | Admin | Manage Enterprise Features menu. You may also schedule an automatic start time for the build process on the same menu.

Project Server uses the dynamic link library (DLL) file Projolap.dll to implement the cube building service. When the OLAP cube update is initiated (manually or by schedule) the Project Data Service (PDS) starts Projolap in two sequential steps. First the database staging tables are built in the Project Server SQL database. Then Projolap sends commands to the Analysis Server directing it to build the cube.

Database staging tables (step 1)

Projolap extracts data from the Project Web Access (PWA) Views tables (MSP_VIEW_) and the Microsoft Project tables (MSP_ other than MSP_VIEW_, MSP_WEB_) to create the cube staging tables (MSP_ CUBE_).

Cube generation (step 2)

Projolap then sends commands to the Analysis Server using the Decision Support Objects (DSO) interface. The Analysis Server then reads data directly from staging tables previously created in the Project Server SQL database and constructs an array of individual data files for each dimension and fact cube. As mentioned earlier, these data files are physically stored under the Microsoft Analysis Services\Data directory on the Analysis Server.

Note that if the cube name you specify on the PWA | Admin | Manage Enterprise Features menu does not exist, Projolap will build a new cube with the name you specify. If your cube will not build after manually manipulating it with the Analysis Manager, you can simply use the Analysis Manager to delete the cube and run `Update now` from PWA to create it again from scratch.

The Task Earned Value Cube Extension

In Microsoft Project Server 2003 the default Portfolio Analyzer cube is limited to resource availability and project level assignment data. Enterprise outline code fields are automatically added to the cube after you begin using them in a project. But if you want to use other enterprise fields or analyze task level details in the Portfolio Analyzer, you must extend the default cube implementation.

The Portfolio Analyzer OLAP Extensions Solution Starter delivered with the Project 2003 SDK provides a complete and functional example for extending the cube. The Task Earned Value cube extension in particular can easily be used as a foundation for creating your own custom cube. Let's take a closer look…

When Projolap updates your cube, it looks for a registered dynamic link library (DLL) containing two Microsoft defined functions. If found, UserStagingTablesUpdate will be executed at the end of Step 1 (staging table update) and UserOLAPUpdate will be executed at the end of Step 2 (cube generation).

The Task Earned Value example has several components including:

- SQL scripts required to create and extend tables needed by the extension
- VB code required to create the MSPOLAPBREAKOUT DLL which includes:
  - UserStagingTablesUpdate
  - UserOLAPUpdate
- The OLAPBreakoutTest application for testing and debugging the DLL
- pj11OLAPExtensions compiled help module for detailed documentation

SQL Database Configuration

Project Server uses the MSProjectServerUser SQL account and the security settings defined in the MSProjectServerRole to run Projolap. This role can not create or alter the structure of a table in the SQL database.

You must manually create and extend the tables needed by the Task Earned Value cube in the project server database. Open the SQL Query Analyzer and select your project server database.

Open the createTasksCubeTable.txt file in your TaskEarnedValue\Utils directory, paste the contents of this file into the SQL Query Analyzer and execute the script to create the fact table and 31 new dimension staging tables.

Open the extendViewTaskTable.txt file in the TaskEarnedValue\Utils directory, paste the contents of this file into the SQL Query Analyzer and execute the script to extend the MSP_VIEW_PROJ_TASK_TP_BY_DAY table.

The QYLIBSQL.sql file in the \Microsoft Office Project Server 2003\BIN directory contains the scripts used to update the MSP_VIEW_PROJ_TASK_TP_BY_DAY table. You must extend this script so that the Project Server can populate the data fields you added in the previous step.

Before starting, create a backup copy of the QYLIBSQL.sql file somewhere on your server. Open the original QYLIBSQL.sql file in a text editor and locate the three lines that start with:

```
// TASKTIMEPHASEDBYDAY QUERIES
50800 SELECT ProjectUniqueID,TaskUniqueID,TaskTimeStart,…
// EXTENDED: 50800 SELECT ProjectUniqueID,TaskUniqueID,…
```

Comment out (do not delete) the second line and add the contents of the qlibsql.txt file in the TaskEarnedValue\Utils directory immediately below the third line. Note that some of these lines are very long and you must *not* break them up with carriage returns. Here is what it should look like when you finish:

```
// TASKTIMEPHASEDBYDAY QUERIES
// 50800 SELECT ProjectUniqueID,TaskUniqueID,TaskTimeStart,…
// EXTENDED: 50800 SELECT ProjectUniqueID,TaskUniqueID,…
// EXTENDED for the Portfolio Manager OLAP extension, for Task Earned…
50800 SELECT ProjectUniqueID,TaskUniqueID,TaskTimeStart,…
```

If you make a mistake while editing this file, recover your backup copy and start over.

Take a moment and use the `Update now` option in PWA to rebuild the default cube. Any difficulties with the default cube should be resolved before proceeding. If it worked before you started and it doesn't work now, start by looking at your changes to the QYLIBSQL.sql file.

Executing the Code

Double-click the OLAPDemoGroup.vbg file in the TaskEarnedValue directory. If everything is configured correctly Visual Studio should launch and load all of the required Task Earned Value application code. If it doesn't, be sure Visual Studio and Visual Basic version 6.0 with service pack 6 or better is installed.

Double-click on the frmOLAPTest form, push the F7 key and scroll down to locate the Form_Load subroutine. Make the following changes:

- change ServerName to the system name of your development server
- change ProjectServer to the name of your project server SQL database
- change OLAPServerName to the system name of your development server
- change CubeNameInAnalysisManager to the cube name you entered on the PWA | Admin | Manage Enterprise Features menu

```
txtDBConnect.Text  = "Provider=sqloledb;Data Source=ServerName;
                      Initial Catalog=ProjectServer;UID=MSProjectServerUser;
                      PWD="
txtOLAPServer.Text = "OLAPServerName"
txtOLAPDB.Text     = "CubeNameInAnalysisManager"
```

If you click the Save icon now, it will minimize the number of times you have to perform this step in the future.

Press the F5 key and the OLAP Breakout Test form should appear on your screen. Verify that the connection, OLAP server and cube fields are filled out correctly. Then enter the password for the MSProjectServerUser account at the end of the Db Connection String field.

Click the Initialize Connection button and then click OK to close the successful completion popup dialog. If your connection was not successful, double check the changes you made in the DB Connection String field. Be sure this works before proceeding.

Click the OLAP Breakout 1 button to test the UserStagingTablesUpdate function. Click OK to close the successful completion popup dialog. If the test failed, check to be sure the staging tables were created correctly when you applied the SQL script in the TasksCubeTable.txt. Again, be sure this works before proceeding.

Click the OLAP Breakout 2 button to test the UserOLAPUpdate function. Click OK to close the successful completion popup dialog. A test failure at this point is unusual. I would start by going to PWA and select the Update now button to verify that the default cube is still working correctly. If this works and you still can't run the OLAP Breakout 2 test, I would use the Analysis Manager to delete the entire cube and rebuild it using `Update now` in PWA.

Congratulations! You have successfully created the MSP_TASK_EARNED_VALUES cube.

Open PWA | Admin | Manage Views and click the `Add View` button. Select the Portfolio Analyzer in the View Type options. Click the `Field List` icon and observe that nothing has changed. This is because the MSP_TASK_EARNED_VALUES cube is not part of the default MSP_PROTFOLIO_ANALYZER cube.

You must select the MSP_TASK_ EARNED_VALUES cube as the source for your new view. To do this click the `Commands and Options` icon, select the `data source` tab, select MSP_TASK_EARNED_VALUES in the Use Data from options. The Field List should now include the measures Task BCWS, Task ACWP, Task BCWP… and dimensions for any enterprise task outline code fields you may have created. Give the view a name, add some fields to the table and save it.

If you want Project Server to automatically build the MSP_TASK_EARNED_VALUES cube every time Projolap is executed, there are a couple more things you have to do.

Go back to Visual Studio and stop the OLAP Breakout Test application if it is still running. Select the `File | Make Project Group` option to create the MSPOLAPBREAKOUT.dll library file.

Create a directory under Microsoft Office Project Server 2003\BIN with the name OLAPExtensions. Copy your new MSPOLAPBREAKOUT.dll file into the new Microsoft Office Project Server 2003\BIN\OLAPExtensions directory.

Open a Command Prompt (DOS) window and cd into the new Microsoft Office Project Server 2003\BIN\OLAPExtensions directory. To register your new DLL enter the command regsvr32 as follows:

```
C:\regsvr32 mspolapbreakout.dll
```

To verify that everything is working, use the Analysis Manager to delete the entire project server cube, rebuild the cube from scratch using the PWA `Update now` command and when the cube build is finished, open the Portfolio Analyzer view you created earlier.

Code Walk

We have covered the cube build process, the basic cube extension process, what the final result looks like and the fundamentals of deployment. Now let’s take a closer look at how the MSPOLAPBREAKOUT library really works. Our purpose here is to understand the general flow of the application paying particular attention to key points we will need to address if the cube is be extended further.

The SDK Task Earned Value code can be divided into three major functions, OLAP Breakout Test, UserStagingTablesUpdate and UserOLAPUpdate. The latter two functions form the MSPOLAPBREAKOUT library.

OLAP Breakout Test is a straight forward windows application that simulates the Projolap interface for testing the MSPOLAPBREAKOUT functions during development. The developer is prompted for the information Projolap will normally pass to the DLL and then provides simple test functions to verify the connection to the server and the two OLAP breakout functions.

Double-click the OLAPDemoGroup.vbg file in the TaskEarnedValue directory. Double-click on the frmOLAPTest form, push the F7 key to display the code. Take a few moments to review the code and spot the calls to UserStagingTablesUpdate and UserOLAPUpdate.

The UserStagingTablesUpdate function is implemented in the UserOptionalCode class module. All the function really does is open the database connection, call the populateStagingTables subroutine and return a status value to the calling program.

Step 1 (building the staging tables) is done in the ProjectDatabaseAccess code module. Select the module in Visual Studio for reference during this presentation.

The MSP_VIEW_PROJ_TASK_TP_BY_DAY table in the project server SQL database contains one entry for each calendar day, for the duration of each task, in each project. This time phased data set allows us to analyze task data for specific time periods of interest within the portfolio analyzer and is used as the basic data source for the MSP_CUBE_TASK_FACT table.

Consider the following example of a small to small moderate project server implementation. A company with 50-60 projects, each with 100-150 tasks spanning a duration of 1-4 years could easily have more than 500,000 records in the MSP_VIEW_ PROJ_TASK_TP_BY_DAY table. When people talk about a cube build taking hours to complete, the record count in this table is measured in millions. This is an environment where program efficiency counts…

The populateStagingTables subroutine starts by calling the populateTaskOutline DimensionTables subroutine to create and populate an elegant memory structure (outlineCodeInfo) that stores the values defined in all 30 of the available enterprise task outline code fields. The outlineCodeInfo structure is then used to populate each of the 30 MSP_CUBE_TASK_OUTLINECODE staging tables.

The next subroutine executed is populateTaskOutlineCodeTranslationTable. This subroutine clears and populates the MSP_CUBE_RESERVED_CACHED_ REDIRECTION_TASK_CODE table. One entry is created in this table for each unique value used in any enterprise task outline code, in any task, in any project. Notice that while the outlineCodeInfo structure stores all of the possible task outline codes, the translation table stores only the values used in one or more of your published projects.

The populateTaskFactTable subroutine is where most of the real work is done. It starts by looking up the date range parameters defined for the cube build on the PWA | Admin | Manage Enterprise Features menu. Then it constructs a complex SQL command and executes it to populate the task fact table. Then it goes back through the fact table and "decumulates" the most of the task data values (measures).

Let's take a closer look at the SQL command used by getCubeTaskFactInsertSQL. It is created by a call to the getCubeTaskFactInsertSQL fnction. If we start with a basic project server, customize the enterprise task outline code 1 field and add a small two day project that uses this field, the SQL command returned by getCubeTaskFactInsertSQL would look (with a few added carriage returns) like this:

```
INSERT INTO MSP_CUBE_TASK_FACT
  (PROJ_UID, TASK_UID, TIME_ID, TASK_ACT_COST, TASK_ACT_FIXED_COST, TASK_ACT_OVT_WORK,
  TASK_ACT_WORK, TASK_ACWP, TASK_BASE_COST, TASK_BASE_WORK, TASK_BCWP, TASK_BCWS,
  TASK_COST, TASK_FIXED_COST, TASK_OVT_WORK, TASK_REG_WORK, TASK_WORK , ENT_TASK_CODE1,
  ENT_TASK_CODE2, ENT_TASK_CODE3, ENT_TASK_CODE4, ENT_TASK_CODE5, ENT_TASK_CODE6,
  ENT_TASK_CODE7, ENT_TASK_CODE8, ENT_TASK_CODE9, ENT_TASK_CODE10, ENT_TASK_CODE11,
  ENT_TASK_CODE12, ENT_TASK_CODE13, ENT_TASK_CODE14, ENT_TASK_CODE15, ENT_TASK_CODE16,
  ENT_TASK_CODE17, ENT_TASK_CODE18, ENT_TASK_CODE19, ENT_TASK_CODE20, ENT_TASK_CODE21,
  ENT_TASK_CODE22, ENT_TASK_CODE23, ENT_TASK_CODE24, ENT_TASK_CODE25, ENT_TASK_CODE26,
  ENT_TASK_CODE27, ENT_TASK_CODE28, ENT_TASK_CODE29, ENT_TASK_CODE30)
SELECT
  TD.ProjectUniqueID,
  TD.TaskUniqueID, DATEDIFF(dd, '2005-09-12', TD.TaskTimeStart) + 1,
  ISNULL(TD.TaskTimeActualCost, 0) / 100,
  ISNULL(TD.TaskTimeActualFixed Cost, 0) / 100,
  ISNULL(TD.TaskTimeActualOvertimeWork, 0) / 60000, ISNULL(TD.TaskTimeActualWork, 0) /
  60000,
  ISNULL(TD.TaskTimeACWP, 0) / 100,
  ISNULL(TD.TaskTimeBaselineCost, 0) / 100,
  ISNULL(TD.TaskTimeBaselineWork, 0) / 60000,
  ISNULL(TD.TaskTimeBCWP, 0) / 100, ISNULL(TD.TaskTimeBCWS, 0) / 100,
  ISNULL(TD.TaskTimeCost, 0) / 100, ISNULL(TD.TaskTimeFixedCost, 0) / 100,
  ISNULL(TD.TaskTimeOvertimeWork, 0) / 60000,
  ISNULL(TD.TaskTimeRegularWork, 0) / 60000,
  ISNULL(TD.TaskTimeWork, 0) / 60000 ,
  (SELECT TOP 1 CubeTaskEntOutlineCodeValue
     FROM MSP_CUBE_RESERVED_CACHED_REDIRECTION_TASK_CODE
     where [ViewTaskEntOutlineCodeValue] = ISNULL(TENT.TaskEnterpriseOutlineCode1ID, -1)
       AND [TaskOutlineCodeIndex] = 1),
  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
  -1, -1, -1, -1, -1, -1, -1
From
  MSP_VIEW_PROJ_TASK_TP_BY_DAY TD
  JOIN MSP_VIEW_PROJ_TASKS_STD TSTD ON TD.TaskUniqueId = TSTD.TaskUniqueId
                                  AND TD.ProjectUniqueID = TSTD.ProjectUniqueID
  JOIN MSP_VIEW_PROJ_TASKS_ENT TENT ON TD.TaskUniqueId = TENT.ENT_TaskUniqueId
                                  AND TD.ProjectUniqueID = TENT.ENT_ProjectUniqueID
where
  TSTD.ProjectUniqueID > 0 and
  TD.TaskTimeStart >= {d'2005-09-12'} and
  TD.TaskTimeStart <= {d'2005-09-13'}
```

Code Example 1

Pay close attention to the portion of this command I have highlighted. This is the SQL script that returns the value of the Enterprise Task Outline Code1 field. If we had customized and used the first 5 Enterprise Task Outline Code fields, this script would be repeated 4 more times replacing the next four (-1) values. The integer value -1 is used in many tables to indicate that no value has been selected.

Because we customized and ***used*** the Enterprise Task Outline Code1 in a project, the value we selected in our project (and the -1 value representing no value) was cached earlier by the populateTaskOutlineCodeTranslationTable subroutine in the MSP_CUBE_ RESERVED_CACHED_ REDIRECTION_TASK_CODE table. While this may seem like overkill for looking up one outline code, consider for a moment the efficiency gains obtainable when looking up several outline codes.

The MSP_VIEW_PROJ_TASK_TP_BY_DAY table is structured and optimized for use in PWA views. As a result most of the values are stored as cumulated (to date) totals. For example if a task of 4 days duration costs $100 per day, the four daily cost values would be stored as 100, 200, 300 & 400 respectively. If we store the data in the analysis cube this way the total task cost would be $1000 instead of $400. The subroutine populateTaskFactTable calls the getDeaggregateChildSQL function to build another complex SQL command and then executes it to “decumulate” these values.

The UserOLAPUpdate function is also implemented in the UserOptionalCode class module. All the function really does is open the SQL database connection, call the BuildCube subroutine and return a status value to the calling program.

Step 2 (building the cube) is done in the ProjectCubeAccess code module. Select the module in Visual Studio for reference during this presentation.

The BuildCube subroutine starts by creating a connection to the Analysis Server and then calls the subroutine populateTaskOutline DimensionTables to create and populate the same elegant memory structure (outlineCodeInfo) used earlier in Step 1.

The Decision Support Objects (DSO) programming interface used to communicate with the Microsoft Analysis Services server makes building the cube a fairly straight forward process.

The function CreateTaskEarnedValueCube is called to create an empty MSP_TASK _EARNED_VALUE cube.

The subroutine setTaskEarnedValueFactTable links the cube to the MSP_CUBE_TASK_ FACT table in the SQL database and creates each of the Measures (data fields) used in the cube.

The addDimensionsToTaskEarnedValueCube subroutine then passes parameters to the AddSharedDimension subroutine to create shared dimensions for the cube.

BuildCube then issues DSO commands to Update the cube structure and Process the data from the SQL staging tables into the Analysis Services cube.

And that's all there is to it…

Making it work for your company or client

Custom enterprise outline codes are easy. Customize an enterprise outline code, use it in a project, rebuild the cube and they appear…

But,

- What about that enterprise text field you are using to store a status indicator?
- What about the Task Name?
- What about the work breakdown structure (WBS)?
- What about the enterprise flag field you are using for a Review Completed indicator?

What to add and where to add it

In this exercise we will add four new Dimensions to the MSP_TASK_EARNED_ VALUES cube. The process we will use has five components:

1. Find the fields we want to add in the Project Server database.
2. Create 4 new SQL tables in the Project Server database to stage our fields.
3. Add the 4 new fields to the MSP_TASK_CUBE_FACT table.
4. Add code to the ProjectDatabaseAccess module to:
   a. Populate the new staging tables.
   b. Populate the new fields in the MSP_TASK_CUBE_FACT table.
5. Add code to the ProjectCubeAccess module to add our new dimensions.

As the four fields we are adding are all task related, they can be found in the MSP_VIEW_PROJ_TASKS_STD and MSP_VIEW_PROJ_TASKS_ENT tables. We will need to know the SQL data type used in the database and decide on a name for each new Dimension. Here is the data needed for our example:

Field Name	SQL Data Type	Table	Dimension Name
TaskEnterpriseText1	ntext	ENT	TaskStatus
TaskName	nvarchar	STD	TaskName
TaskOutlineNumber	ntext	STD	WBS
TaskEnterpriseFlag1	tinyint	ENT	Reviewed

You may have noticed by now that there are a lot of tables in the project server database and a good naming structure can make it easier to find the table you are looking for. Give some thought to the names you use for your new staging tables. For this exercise our staging table names will start with AAA_TaskCube_ and end with the dimension name. If your client or organization has a three or four letter acronym or label, I would strongly consider using it instead of AAA.

The staging table needs to contain two fields. One field will hold the value you stored in the project server field, the other will be used as an index. You can create the required tables using the `New Table` command in the SQL Enterprise Manager, or you can write a SQL script and execute it in the SQL Query Analyzer. A sample script can be found below as Code Example 2.

Four new fields need to be added to the MSP_TASK_CUBE_FACT table. These fields will store an index (ID) value from one of the new dimension staging tables. You add these fields using either the SQL Enterprise Manager or Query Analyzer tools. A sample script is included at the bottom of Code Example 2.

```
-- Create the 4 new staging tables

CREATE TABLE AAA_TaskCube_TaskStatus (
  TaskStatus nvarchar(24) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  ID int NULL
)

CREATE TABLE AAA_TaskCube_TaskName (
  TaskName nvarchar(255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  ID int NULL
)

CREATE TABLE AAA_TaskCube_WBS (
  WBS nvarchar(20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  ID int NULL
)

CREATE TABLE AAA_TaskCube_Reviewed (
  Reviewed nvarchar(10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  ID int NULL
)

-- Add the 4 new fields to the task fact table

ALTER TABLE MSP_CUBE_TASK_FACT ADD
  TaskStatus_ID  int  NULL,
  TaskName_ID    int  NULL,
  WBS_ID         int  NULL,
  Reviewed_ID    int  NULL
```

Code Example 2

I find it helpful to copy the TaskEarnedValue directory into a new directory before starting a new cube extension project. The new directory helps me keep my different cube projects sorted out and ensures that I have an unmodified reference if something gets messed up.

Open Visual Studio and select the ProjectDatabaseAccess module. Our first task is to populate the new staging tables. We will create a new subroutine for this purpose. Code Example 3 illustrates one approach to doing this.

```
Private Sub populateAAAStagingTables(dbConnection As ADODB.Connection)
    Dim strSQLCommand As String
    Dim sqlrecordset As New ADODB.Recordset
    Dim sqlCmd As ADODB.Command

    Dim sTrace As String
    Dim sProjectTextValue As String
    Dim iTextIndex As Integer

    Dim fSuccess As Boolean
```

```
fSuccess = False

On Error GoTo LblErrorHandler

' Now delete existing WBS Dimension table records and repopulate
Set sqlCmd = New ADODB.Command
sqlCmd.ActiveConnection = dbConnection
sqlCmd.CommandText = "DELETE FROM AAA_TaskCube_WBS"
sqlCmd.CommandType = adCmdText
Call sqlCmd.Execute(, , adExecuteNoRecords)
sTrace = "AAA_TaskCube_WBS records dropped"
Call Trace(sTrace)
' First create a 'No Value' entry
strSQLCommand = "Insert INTO AAA_TaskCube_WBS VALUES ('No Value', -1 )"
Set sqlCmd = New ADODB.Command
sqlCmd.ActiveConnection = dbConnection
sqlCmd.CommandText = strSQLCommand
sqlCmd.CommandType = adCmdText
sqlCmd.Execute
' Then find and add all of the other unique values
' It is necessary to cast the TaskOutlineNumber as a nvarchar data type
'   because the ntext value can make search comparisons difficult
strSQLCommand = "select distinct cast(TaskOutlineNumber as " & _
                "nvarchar(20)) as 'WBS' from MSP_VIEW_PROJ_TASKS_STD order by 'WBS'"
sqlrecordset.Open strSQLCommand, dbConnection
iTextIndex = 1
While Not sqlrecordset.EOF
    sProjectTextValue = sqlrecordset.Fields("WBS").Value
    strSQLCommand = "Insert INTO AAA_TaskCube_WBS" & _
                    " VALUES ('" & sProjectTextValue & "'," & iTextIndex & " )"
    Set sqlCmd = New ADODB.Command
    sqlCmd.ActiveConnection = dbConnection
    sqlCmd.CommandText = strSQLCommand
    sqlCmd.CommandType = adCmdText
    sqlCmd.Execute
    iTextIndex = iTextIndex + 1
    sqlrecordset.MoveNext
Wend
sqlrecordset.Close
Call Trace("AAA_TaskCube_WBS table populated")

' Now delete existing TaskName Dimension table records and repopulate
Set sqlCmd = New ADODB.Command
sqlCmd.ActiveConnection = dbConnection
sqlCmd.CommandText = "DELETE FROM AAA_TaskCube_TaskName"
sqlCmd.CommandType = adCmdText
Call sqlCmd.Execute(, , adExecuteNoRecords)
sTrace = "AAA_TaskCube_TaskName records dropped"
Call Trace(sTrace)
' First create a 'No Value' entry
strSQLCommand = "Insert INTO AAA_TaskCube_TaskName VALUES ('No Value', -1 )"
Set sqlCmd = New ADODB.Command
sqlCmd.ActiveConnection = dbConnection
sqlCmd.CommandText = strSQLCommand
sqlCmd.CommandType = adCmdText
sqlCmd.Execute
' Then find and add other values
strSQLCommand = "Select Distinct TaskName from MSP_VIEW_PROJ_TASKS_STD"
sqlrecordset.Open strSQLCommand, dbConnection
iTextIndex = 1
While Not sqlrecordset.EOF
    sProjectTextValue = sqlrecordset.Fields("TaskName").Value
    strSQLCommand = "Insert INTO AAA_TaskCube_TaskName" & _
                    " VALUES ('" & sProjectTextValue & "'," & iTextIndex & " )"
    Set sqlCmd = New ADODB.Command
    sqlCmd.ActiveConnection = dbConnection
    sqlCmd.CommandText = strSQLCommand
    sqlCmd.CommandType = adCmdText
    sqlCmd.Execute
    iTextIndex = iTextIndex + 1
    sqlrecordset.MoveNext
```

```
 Wend
 sqlrecordset.Close
 Call Trace("AAA_TaskCube_TaskName table populated")

 ' Now delete existing TaskStatus Dimension table records and repopulate
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
 sqlCmd.CommandText = "DELETE FROM AAA_TaskCube_TaskStatus"
 sqlCmd.CommandType = adCmdText
 Call sqlCmd.Execute(, , adExecuteNoRecords)
 sTrace = "AAA_TaskCube_TaskStatus records dropped"
 Call Trace(sTrace)
 ' First create a 'No Value' entry
 strSQLCommand = "Insert INTO AAA_TaskCube_TaskStatus VALUES ('No Value', -1 )"
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
 sqlCmd.CommandText = strSQLCommand
 sqlCmd.CommandType = adCmdText
 sqlCmd.Execute
 ' Then find and add other values
 ' It is necessary to cast the TaskOutlineNumber as a nvarchar data type
 '   because the ntext value can make search comparisons difficult
 strSQLCommand = "select distinct cast(TaskEnterpriseText1 as nvarchar(24)) as " & _
                 "'TaskStatus' from MSP_VIEW_PROJ_TASKS_ENT order by 'TaskStatus'"
 sqlrecordset.Open strSQLCommand, dbConnection
 iTextIndex = 1
 While Not sqlrecordset.EOF
     sProjectTextValue = sqlrecordset.Fields("TaskStatus").Value
     strSQLCommand = "Insert INTO AAA_TaskCube_TaskStatus" & _
                     " VALUES ('" & sProjectTextValue & "'," & iTextIndex & " )"
     Set sqlCmd = New ADODB.Command
     sqlCmd.ActiveConnection = dbConnection
     sqlCmd.CommandText = strSQLCommand
     sqlCmd.CommandType = adCmdText
     sqlCmd.Execute
     iTextIndex = iTextIndex + 1
     sqlrecordset.MoveNext
 Wend
 sqlrecordset.Close
 Call Trace("AAA_TaskCube_TaskStatus table populated")

 ' Now delete existing Reviewed Dimension table records and repopulate
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
 sqlCmd.CommandText = "DELETE FROM AAA_TaskCube_Reviewed"
 sqlCmd.CommandType = adCmdText
 Call sqlCmd.Execute(, , adExecuteNoRecords)
 sTrace = "AAA_TaskCube_Reviewed records dropped"
 Call Trace(sTrace)
 ' This Flag field has only three possible entries. No, Yes & No Value
 ' Create a 'No Value' entry
 strSQLCommand = "Insert INTO AAA_TaskCube_Reviewed " & _
                 "VALUES ('No Value', -1 )"
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
 sqlCmd.CommandText = strSQLCommand
 sqlCmd.CommandType = adCmdText
 sqlCmd.Execute
' Create a 'No' entry
 strSQLCommand = "Insert INTO AAA_TaskCube_Reviewed " & _
                 "VALUES ('No', 0 )"
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
 sqlCmd.CommandText = strSQLCommand
 sqlCmd.CommandType = adCmdText
 sqlCmd.Execute
' Create a 'Yes' entry
 strSQLCommand = "Insert INTO AAA_TaskCube_Reviewed " & _
                 "VALUES ('Yes', 1 )"
 Set sqlCmd = New ADODB.Command
 sqlCmd.ActiveConnection = dbConnection
```

```
    sqlCmd.CommandText = strSQLCommand
    sqlCmd.CommandType = adCmdText
    sqlCmd.Execute
    Call Trace("AAA_TaskCube_Reviewed table populated")

    fSuccess = True

LblErrorHandler:

   Set sqlCmd = Nothing

   If (Not fSuccess) Then
      Call TraceAndRaise("Error in populateAAADimensionTables.")
   End If

End Sub
```

Code Example 3

To execute our new populateAAAStagingTables we will add a call in the populateStagingTables subroutine. Please refer to the highlight text in Code Example 4.

```
' Based on the info stored in outlineCodeInfo, populate the task outline dimension tables
Call populateTaskOutlineDimensionTables(dbConnection, outlineCodeInfo)

' Create the AAA specific staging tables
Call populateAAAStagingTables(dbConnection)

' Based on the info stored in outlineCodeInfo, populate the task outline code translation
Call populateTaskOutlineCodeTranslationTable(dbConnection, outlineCodeInfo)
```

Code Example 4

Now we need to modify the getCubeTaskFactInsertSQL routine so that it will select the index values for our new our new dimension fields and insert them into the MSP_CUBE_ TASK_FACT table. To keep this clean and make it easier to add additional fields at some later date, we will create two new functions (getAAAFieldRef, getAAACodeRef) and insert calls to execute them in the getCubeTaskFactInsertSQL subroutine. Code Example 5 presents both of the required subroutines. Code Example 6 demonstrates where to insert the subroutine calls in getCubeTaskFactInsertSQL.

```
' getAAAFieldRef will return a string containing the AAA specific fields
Private Function getAAAFieldRef() As String
    getAAAFieldRef = ""

    getAAAFieldRef = ", TaskStatus_ID, TaskName_ID, WBS_ID, Reviewed_ID "
End Function


' getAAACodeRef will return a string containing the AAA specific SQL statement
Private Function getAAACodeRef() As String

    getAAACodeRef = ""

    ' Add the TaskStatus_ID field selector
    getAAACodeRef = getAAACodeRef & ",isnull((select Top 1 ID " & _
                                  "From AAA_TaskCube_TaskStatus " & _
                                  "where TaskStatus like " & _
                                  "(select cast(TaskEnterpriseText1 " & _
                                  "as nvarchar(24)))),-1) "

    ' Add the TaskName_ID field selector
    getAAACodeRef = getAAACodeRef & ",isnull((select Top 1 ID " & _
                                  "from AAA_TaskCube_TaskName b " & _
                                  "where b.TaskName like TSTD.TaskName),-1) "

    ' Add the WBS_ID field selector
    getAAACodeRef = getAAACodeRef & ",isnull((select Top 1 ID " & _
                                  "from AAA_TaskCube_WBS b, MSP_TASKS t " & _
                                  "where TD.TaskUniqueID = t.Task_UID and " & _
                                  "b.WBS like t.TASK_OUTLINE_NUM),-1) "

    ' Add the Reviewed_ID field selector
    ' This is simply a matter of using the MSP_VIEW_PROJ_TASKS_ENT value
    getAAACodeRef = getAAACodeRef & ",TENT.TaskEnterpriseFlag1 "

End Function
```

Code Example 5

The following code snippets are from the getCubeTaskFactInsertSQL subroutine.

```
------
   For nTaskOutlineCodeIndex = 1 To NUM_TASK_CODES_TO_EXPORT
      strSQLCommand = strSQLCommand & ", ENT_TASK_CODE" & CStr(nTaskOutlineCodeIndex)
   Next nTaskOutlineCodeIndex

   ' This is where we add the new AAA fields to the Insert portion of the SQL script
   strSQLCommand = strSQLCommand & getAAAFieldRef

   strSQLCommand = strSQLCommand & " ) "

------

   strSQLCommand = strSQLCommand & strSQLOutlineCodeRef

   ' insert the AAA specific field selections here
   strSQLAAACodeRef = getAAACodeRef
   strSQLCommand = strSQLCommand & strSQLAAACodeRef

   strSQLCommand = strSQLCommand & _
```

Code Example 6

Select the ProjectCubeAccess module in Visual Studio. Our last task is to add the code required to add our new Dimensions to the cube. We will create two new subroutines and modify two existing subroutines to accomplish this task.

The AAA_CreateSharedDimensions subroutine shown in Code Example 7 will make it easy to add additional dimensions at some future date.

```
Private Sub AAA_CreateSharedDimensions(sServerName As String, sDBName As String)
    Call AAA_CreateNewDimension(sServerName, sDBName, "WBS")
    Call AAA_CreateNewDimension(sServerName, sDBName, "TaskName")
    Call AAA_CreateNewDimension(sServerName, sDBName, "TaskStatus")
    Call AAA_CreateNewDimension(sServerName, sDBName, "Reviewed")
End Sub
```

Code Example 7

All the AAA_CreateSharedDimensions subroutine does is call the AAA_CreateNew Dimension subroutine shown in Code Example 8 once for each new dimension. This is where the new dimensions are really created.

```
Private Sub AAA_CreateNewDimension(sServerName As String, sDBName As String, sDimName As
String)
    Dim dsoServer As New DSO.Server
    Dim dsoDB As DSO.MDStore
    Dim dsoCube As DSO.MDStore
    Dim dsoDS As DSO.DataSource
    Dim dsoDim As DSO.Dimension
    Dim dsoLev As DSO.Level
    Dim dsoMember As DSO.MemberProperty
    Dim I, C As Integer
    Dim cTemp As Integer
    Dim sTNum As String
    Dim stemp As String

    ' Constants used for ColumnType property
    ' of the DSO.Level object.
    ' Note that these constants are identical to
    ' those used in ADO in the DataTypeEnum enumeration.
    Const adWChar = 130
    Const adInteger = 3
    Const adDouble = 5

    Dim fSuccess As Boolean
    fSuccess = False

    On Error GoTo LblErrorHandler

    Call Trace("AAA_CreateNewDimensions starting")

    ' Create a connection to the Analysis server.
    dsoServer.Connect sServerName

    ' Set the database object.
    Set dsoDB = dsoServer.MDStores(sDBName)

    ' Set the data source for the database object.
    ' A data source is required to run this example.
    If dsoDB.DataSources.Count = 0 Then
        MsgBox "Database " & dsoDB.Name & _
            " has no data sources."
    Else
        Set dsoDS = dsoDB.DataSources(1)
    End If

    ' First remove any existing instances of this dimension
    If dsoDB.Dimensions.Find(sDimName) Then
        For Each dsoCube In dsoDB.MDStores
            If dsoCube.Dimensions.Find(sDimName) Then
            stemp = dsoCube.Name
                dsoCube.Dimensions.Remove sDimName
                Call Trace("Dimension " & sDimName & " removed from " & stemp & " cube")
            End If
        Next
        dsoDB.Dimensions.Remove sDimName
        Call Trace(sDimName & " removed from sharred dimensions")
    End If

    ' Create dimension.
    Set dsoDim = dsoDB.Dimensions.AddNew(sDimName, sbclsRegular)
    Set dsoDim.DataSource = dsoDS   ' Dimension data source
    dsoDim.FromClause = """AAA_TaskCube_" & sDimName & """"
    dsoDim.JoinClause = ""

    ' Add ALL link level.
    Set dsoLev = dsoDim.Levels.AddNew("(ALL)")
    dsoLev.LevelType = levAll
    dsoLev.MemberKeyColumn = "All " & sDimName

    ' Add link level.
    Set dsoLev = dsoDim.Levels.AddNew(sDimName, sbclsRegular)
    dsoLev.MemberKeyColumn = """AAA_TaskCube_" & sDimName & """.""ID"""
    dsoLev.MemberNameColumn = """AAA_TaskCube_" & sDimName & """.""" & sDimName & """"
    dsoLev.ColumnSize = 4
    dsoLev.columnType = 3
    dsoLev.EstimatedSize = 4

    ' Update the dimension
    dsoDim.Update

    ' Process the dimension
    dsoDim.Process processFull

            Call Trace(sDimName & " dimension created")

    fSuccess = True

LblErrorHandler:

   If (Not fSuccess) Then
      Call TraceAndRaise("Error in AAA_CreateNewDimensions")
   End If

End Sub
```

Code Example 8

We must add code in the BuildCube subroutine to call the AAA_CreateShared Dimensions subroutine. The snippet highlighted in Code Example 9 shows you where to make the addition.

```
   ' Create the new task outline dimensions
   Call createTaskOutlineDimensions(dsoDatabase, dsoCube, dbConnection, outlineCodeInfo)

   ' Create the AAA_TASK_FACT shared dimensions
   Call AAA_CreateSharedDimensions(sOLAPServerName, sOLAPDatabaseName)

   ' Add tne new and existing shared dimensions
   Call addDimensionsToTaskEarnedValueCube(dsoDatabase, dsoCube, outlineCodeInfo)
```

Code Example 9

Our last modification adds the code highlighted in Code Example 10 to the addDimensionsToTaskEarnedValueCube subroutine. This code will add our new dimensions to the MSP_TASK_EARNED_VALUES cube.

```
   Call AddSharedDimension(dsoDatabase, dsoCube, strJoin, "Project Versions", _
                           "(""MSP_CUBE_PROJECTS"".""PROJ_VERSION_UID""=""" & _
                           "MSP_CUBE_PROJ_VERSIONS"".""PROJ_VERSION_UID"" )")

   'If needed and if they exist you can add other shared dimension like "Project
   '  Location" with other calls to AddSharedDimension

   Call AddSharedDimension(dsoDatabase, dsoCube, strJoin, "WBS", _
   " (""MSP_CUBE_TASK_FACT"".""WBS_ID""=""AAA_TaskCube_WBS"".""ID"" )")

   Call AddSharedDimension(dsoDatabase, dsoCube, strJoin, "TaskName", _
   " (""MSP_CUBE_TASK_FACT"".""TaskName_ID""=""AAA_TaskCube_TaskName"".""ID"" )")

   Call AddSharedDimension(dsoDatabase, dsoCube, strJoin, "TaskStatus", _
   " (""MSP_CUBE_TASK_FACT"".""TaskStatus_ID""=""AAA_TaskCube_TaskStatus"".""ID"" )")

   Call AddSharedDimension(dsoDatabase, dsoCube, strJoin, "Reviewed", _
   " (""MSP_CUBE_TASK_FACT"".""Reviewed_ID""=""AAA_TaskCube_Reviewed"".""ID"" )")
```

Testing and deployment

Now double-click on the frmOLAPTest form and press your `F5` key to launch the OLAP Breakout Test application. Select the `OLAP Breakout 1` button and then the OLAP `Breakout 2` button to build your new MSP_TASK_EARNED_VALUES cube. If you receive any errors, debug and resolve them before proceeding.

Use Visual Studio to build a new MSPOLAPBREAKOUT library.

BEFORE you copy the new MSPOLAPBREAKOUT.dll file into your OLAPExtensions directory, open a Command Prompt (DOS) window and cd into the Microsoft Office Project Server 2003\BIN\OLAPExtensions directory. Unregister your existing MSPOLAPBREAKOUT DLL using the following regsvr32 command:

```
C:\regsvr32 mspolapbreakout.dll –u
```

After you have unregistered the MSPOLAPBREAKOUT DLL, delete it and copy the new MSPOLAPBREAKOUT.dll file into the Microsoft Office Project Server 2003\BIN\OLAPExtensions directory. Register your new DLL using the following command:

```
C:\regsvr32 mspolapbreakout.dll
```

To verify that everything is working, use the Analysis Manager to delete the entire project server cube, rebuild the cube from scratch using the PWA `Update now` command and when the cube build is finished, open the Portfolio Analyzer view you created earlier. Click the `Field List` icon and verify that your new TaskName, TaskStatus, WBS and Reviewed dimensions are available and functional.

After you have your new cube working and validated on your development server, there are a few more steps required to deploy the new cube on your production server.

1. ***BACKUP your production server carefully with particular care to the SQL database!*** There is little or no forgiveness available for anyone who omits this step…

2. Repeat all of the steps in the **SQL Database Configuration** section of this document on your production server.

3. Execute the SQL script presented in Code Example 2 on your production server.

4. Create an OLAPExtensions directory on your production server and copy your verified and tested mspolabbreakout.dll library file into it.

5. Open a Command Prompt window on the production server and register the new library with the regsvr32 utility.

6. Use the Analysis Manager to delete the entire project server cube, rebuild the cube from scratch using the PWA `Update now` command and when the cube build is finished, create a new Portfolio Analyzer view using the MSP_TASK_EARNED_VALUES cube as the source.

Wrap Up

The 'tell it', 'explain it' and 'do it again' approach has proven over time to be both repetitive and effective. The people who don't like it generally skip ahead any way, so if you made it to the end and everything is working, you are probably happy to be among the small group of people who have successfully tackled this topic.

A copy of this document and the modified code it describes is available from the author.

While every effort has been made to test and verify the information presented in this document, there will always be room for improvement. Courteous and constructive comments or criticism is always appreciated and will be warmly received.

If you are having difficulties getting the cube to work, please feel free to contact the author. The easy answers are mostly free and the author makes his living on resolving the rest.

Contact information for Bob Segrest is listed below:

Bob Segrest
Bob.Segrest@BSegE.com
Http://www.BSegE.com
Office: 01 (540) 937-5875
Mobile: 01 (540) 229-2258