```
+ *********************************************************************
|
|     AdventureWorks Tabular AMO Setup and Execute file
|
+ *********************************************************************
```

Contents

I. Introduction

‘AdventureWorks Tabular AMO’ is a C# sample program to demonstrate how to create a Tabular model using the AMO2Tabular V2 library (AMO2Tabular). The main purpose of the sample is to illustrate how you would create a tabular model, as a developer, using a programming language and AMO2Tabular.

This guide is about how to get the best learning experience from this sample. The short explanation about the sample comes in the readme; here I explore en detail how the sample works and how the library is used.

Again, and not tired of repeating it; a very important note here is to say, that by design: models created using the AMO2Tabular V2 library cannot be used in Microsoft SQL Server Data Tools (formerly known as BIDS). Models created using AMO2Tabular, can be queried and used from Microsoft SQL Server Management Studio. The main reason why models created with AMO2Tabular V2 don't work in Data Tools is because the library, on purpose, doesn't add any annotations. Data Tools decorate Analysis Services objects with annotations that later retrieve in order to deal with the object in the tool; in the absence of those annotations Data Tools will crash on the first interaction after opening an imported model.

II. Requirements

A. Software

- SQL Server 2012 (Standard, Enterprise or Developer edition)
- Visual C# 2010 Express or Visual Studio 2010 (Professional, Premium or Ultimate edition)
- Visual Studio 2010 SP1

B. Environment

- SQL Server 2012 RDBMS Engine installed in local machine.
- SQL Server 2012 Analysis Services installed in local machine and running in Tabular mode.
- 'AdventureWorksDW2012' relational database installed; download from here http://msftdbprodsamples.codeplex.com/downloads/get/165405
- (optional) 'AdventureWorks Tabular Model SQL Server 2012' installed; download from here http://msftdbprodsamples.codeplex.com/downloads/get/353143

C. Experience and proficiency

In this sample I assume you are a developer with experience in AMO and C#.

D. Roles and users setup

- The following users need to be added to the local machine, to be able to test the roles functions. These users don't need any special privileges:
  - AmoAdmin
  - AmoAnalyst

        - AmoOperator
        - brian3
        - syed0
        - lynn0
        - ed0
        - ken0
        - laura1
        - TestUser
    - Here, and for the success of this part of the sample, I'm assuming that you are running the sample in the same machine you are hosting both SQL Server engine and Analysis Services

III. Install and Setup

The following set of steps will take you through a detailed installation and setup of the sample

1) If you are reading this note you already downloaded the required files and had unzipped them; in case you haven't unzipped the files, do it now to your preferred location for developer projects.
2) If you haven't installed SQL Server 2012, this is the right time to do it.
    - When installing SQL Server, make sure you install the SQL Engine and the Analysis Services service in Tabular mode.
    - It is also very convenient to have both services (engine and tabular) running under the same local account for the purpose of this sample.
    - Add yourself (or the user who will run this sample) to the service administrators list when installing the tabular instance. By default I add myself as service administrator to both instances; but, I don't know about you. This step is crucial to run the sample; if the user running the sample is not an Analysis Services service administrator, the user cannot create a tabular database and the sample will fail around the third line of code (I don't count using statements, namespace, comments, opening/closing braces, etc.).
3) Download and install 'AdventureWorksDW2012' relational database; from here http://msftdbprodsamples.codeplex.com/downloads/get/165405
4) (highly recommended) Download, install and deploy 'AdventureWorks Tabular Model SQL Server 2012'; from here http://msftdbprodsamples.codeplex.com/downloads/get/353143

- Verify ‘AdventureWorks Tabular Model SQL Server 2012’ is correctly deployed.
- Open SSMS
- Right click on ‘AdventureWorks Tabular Model SQL Server 2012’ database and select new query in MDX
- Execute the following query: evaluate summarize(generate(values('Date'[Calendar Year]), values('Product Category'[Product Category Name])), [Calendar Year], [Product Category Name], "Total Sales" , 'Sales Territory'[Total Sales], "Total Inventory Value", 'Product Inventory'[Total Inventory Value])
- Compare your results to appendix ‘Results for database verification’, at the end of this document.

5) Verify the solution file. Open the ‘Tabular AMO.sln’ solution file with Visual Studio or by double clicking on it.
   - Check no yellow (warning) icons are in the any of the elements of the project.
   - Check the region Add Roles, Users and RLS; the code should be grayed. This is correct at this moment.
6) Build the solution either by selecting build the solution, from the Build menu, or by pressing [CTRL+SHIFT+B].
   - Builds the library -AMO2Tabular-
   - Builds the sample -AdventureWorks Tabular AMO-
7) Run the solution. The sample should run with no problems, if Microsoft SQL Server is installed, both instances (rdbms and tabular) are running and the user executing the sample has read access to ‘AdventureWorksDW2012’ and has server administrator privileges to the tabular instance.
   - You should see console messages stating what the sample is doing.
8) Verify the sample run correctly
   - Open SSMS
   - Right click on ‘AdventureWorks Tabular AMO 2012’ database (congrats!!! You just have created a tabular database from your solution) and select new query in MDX
   - Execute the following query: evaluate summarize(generate(values('Date'[Calendar Year]), values('Product

```
Category'[Product Category Name])), [Calendar Year], [Product
Category Name], "Total Sales" , 'Sales Territory'[Total Sales],
"Total Inventory Value", 'Product Inventory'[Total Inventory Value])
```

- Your results should be the same as those obtained when you verified 'AdventureWorks Tabular Model SQL Server 2012' database. Compare to appendix 'Results for database verification' at the end of this document.

IV. Executing the sample

A. Setting up the scenario

1) This sample is about a piece of code capable of creating a complex tabular model; think in the number of clicks and visual steps it takes to you to create a model using the standard interface and you'll get a sense of the number of lines of code you application will have.
2) To build the model, the AMO2Tabular library of functions is used; AMO2Tabular is specifically designed for the purpose of managing tabular models.
3) Using the sample, many repeated times, will create the same database over and over.
4) By design, the sample doesn't delete the database. You'll have to delete the database every time you want to re-run the sample.

B. Understanding all but role security

The following itemized list will walk you through the major steps in the sample.

Before you open the solution, open SQL Server Management Studio -SSMS- and connect to the local instance of Analysis Services. Be sure to delete any instance of 'AdventureWorks Tabular AMO 2012'. Keep SSMS opened.

Open the sample solution.

1) Creating a Tabular database

   The TabularDatabaseAdd() function, with the given parameters in the sample, not only creates a database object, in tabular mode, but it creates a DataSource object -DS- and a DataSourceView object -DSV-.

   The important thing to understand here is that the DSV object will contain the blue-print of what you can build in your tabular model. In other words, you cannot add a table to a tabular model that is not defined in the DSV, when using AMO2Tabular. You are free to create any table in your model, as long as its definition exists in the DSV. The same applies to columns; if they are not defined in the DSV, you cannot add them to the model. The DSV will be populated with the entire schema of the relational data source; data source that was provided through the OleDb connection string passed in the parameters of TabularDatabaseAdd().

Also, a DataSource object is known as the Connection object in tabular models; and DataSourceViews are not exposed to users in tabular mode.

Execute time:

i. Set a breakpoint at: CreateDateTable(AMO2TabularDb, Resources.cubeName);

ii. Press F5 and let the code run to the breakpoint.

iii. Switch to SSMS and refresh your list of AS databases.

iv. You should see a new database: ‘AdventureWorks Tabular AMO 2012’

v. Expand the database tree and you’ll see that only a connection object exists.

2) Creating the first table

Why is it needed to have a different function to create the first table, rather than have only one function capable of creating all tables?

Because, before you create the first table, a cube needs to be created and a name must be given. Also, because we don’t want to hardcode cube names anywhere. In order to be able to ask the name of the cube, we decided to have a separate function to create the first table and ask for the name of the cube; internally what happens is that the cube gets created first and the table is added afterwards.

No execution time here; check next step.

3) Creating all other tables

Creating each table is a long and detailed task; for that reason, and to minimize code clutter, a set of Create<tableName>Table() wrapping functions were created.

Inside each of those Create<tableName>Table() functions you find the following sections:

i. TableAdd(); that adds a table to the model. The false parameter, at the end of the parameters list, tells AMO2Tabular not to update the server instance yet. The table is added as a copy of the table definition in the DSV; that happens to be a copy of the table as defined in the relational data source.

ii. Update Column names; change in the model the column names.

iii. Remove not used columns; remove from the model those columns from the original table you don’t want to expose to the user. This is different than setting the visibility property to false; a removed column is no longer part of the model and, in consequence, cannot be used in any expression. On the other hand, setting the visibility of a column to false do not preclude you from using it in expressions; the visibility property is a setting for client tools, not a security feature.

iv. Add CalculatedColumns, Measures and Hierarchies; these sections are pretty straightforward, as they do what they say they’ll do.

v. Updating server instance; here is where the server gets updated with all the elements and updates of the new table. We could have updated the server at each of the above steps; but, that would have affected the overall performance of the application because each update is a round trip of Database.Alter(). To have updated the server instance at each step, we would have had to set the updateInstance parameter to true, which is the default; that’s why we had to explicitly set it to false.

Let’s have some tables (all of them) created.

Execute time:

i. Set a breakpoint at:
AMO2Tabular.RelationshipAdd(AMO2TabularDb, "Currency", "Currency Id", "Internet Sales", "Currency Id");

This is the first RelationshipAdd() function call after all Create<tableName>Table invocations.

ii. Press F5 and let the code run to the breakpoint.

iii. Switch to SSMS and refresh your list of AS databases.

iv. Expand the database tree and you'll see that only one connection object exists and 15 new tables should be there.

v. Right click on the database and select New Query, MDX

vi. If you try to run a query on any table, you'll get an error message stating that tables have not been processed. This is true; because, so far, we have only defined them.

4) Creating relationships

This section is very much straightforward; here all relationships for the model are created. Most of all relationships are defined as Active relationships; only a few were created as In-Active.

When you create an active relationship, make sure that new active relationship does not open multiple paths between two tables. By design, tabular models do not support alternate paths between two tables. In other words, if table Sales is the end of multiple path relationship that starts from Geography and the user wants to slice Sales by Geography; then, and to avoid ambiguities, there must be one and only one possible relationship path from Geography to Sales.

If creating a new active relationship violates the no-multiple relationship paths rule; the RelationshipAdd() function will throw an exception.

5) Creating all objects that have dependencies on relationships

This section is required when table elements, that have dependencies on objects outside the own table, are created.

If the defining expression of a measure or calculated column has one or more references to columns or tables outside its own table, then all required relationships that would connect all of those referenced columns to the expression need to be defined before the expression can be used; and, defining the object is the first usage of the expression.

No execution time here; check next step.

6) Creating perspectives

   Once all tabular objects are created, you can create perspectives. Perspectives are a visualization aid for client tools; perspectives are not a security feature.

   No execution time here; check next step.

7) Updating model and processing for the first time

   You are here, ready to update and process your tabular model. While coming here you skipped over the Roles section; this is a defined at the beginning of this section, we'll cover Roles in the next section.

   Execute time:

   i. Set a breakpoint at: tableName = "Internet Sales";

      This is the first instruction in the Manage Partitions region.

   ii. Press F5 and let the code run to the breakpoint.

   iii. Switch to SSMS and refresh your list of AS databases.

   iv. Right click on the database and select New Query, MDX

   v. Try to run a query on any table. You should get your expected results.

   vi. In case you can't come out with any query to test the database, try with this: evaluate row("Reseller Total Sales", [Reseller Total Sales], "Total Product Inventory Value to last day", [Total Inventory Value])

   vii. You should get these values: 80450596.9823   23603975.57

   viii. Also, if you run the same query against the 'AdventureWorks Tabular Model SQL 2012' sample database; you should get the same results.

8) Managing partitions

Now that the model is complete and processed; let’s see what can be done with partitions.

This final section is about to show some of the different capabilities of AMO2Tabular to manage partitions.

Maybe, from the beginning we should have set the partitions we wanted for each table; but having done such would have been completely different than real life experience. Hence, here we start with partitions as defined by default when a table is created.

First, you cannot drop all partitions from a table; the server won’t allow you. For that reason we define one or two partitions before dropping the default partition of a table; after the default partition was dropped, you can process the newly added partitions.

When merging partitions, remember that all source partitions are merged into the destination partition and the properties of the destination partition are not updated by the source partitions. Therefore, if you immediately (after the merge) process the destination partition you will lose all source partitions… why? Well, because you haven’t updated the select statement of the partition and, consequently, you’ll get the data defined in the original select statement of the destination partition; correspondingly, the name of the destination partition still reflects the name of the original destination partition.

To be sure that you understand the above statements; in the sample, we update the select statement and the name of the destination partition immediately after merging the partitions. One final word, on merging partitions, always issue a ProcessRecalc, at database level, when you are done merging partitions; otherwise, for that table or related tables you’ll get an error stating incomplete references.

C. Roles enabled

This part of the sample deals with security and how users are affected by it.

I will not explain here how Tabular Models implement security; but rather will show how each of the elements is used. The key elements to tabular model security are: Roles, RoleMembers and Row Level Security -rls-.

Before we can test the security elements, an environment needs to be created. For that purpose we have the first step in this section. The remaining steps are to follow the sample.

1) Environment setup

This section is a preparation prerequisite that happens outside Visual Studio and SSMS. To be able to do complete the following steps, you need to have Administrator privileges in the machine where you are running the sample.

i. Add the following users to the local machine:

   AmoAdmin, AmoAnalyst, AmoOperator, brian3, syed0, lynn0, ed0, ken0, laura1, TestUser

   Give them passwords that meet your company security standards.

ii. These users should not have any special privileges in the local machine

iii. Names have to be exactly as shown above or the sample will fail. AMO2Tabular verifies the existence of the users and will throw an exception if a user cannot be found (either locally or in the domain, if a domain is given).

iv. The following names are taken from the Employees table and are needed to test RLS: brian3, syed0, lynn0, ed0, ken0, laura1

2) Creating roles, adding role members and defining row level security

   To avoid code cluttering, the following procedure was added to handle role creation: buildRoles().

   The buildRoles() function is just a shortcut to invoke:

   - RoleAdd()
   - RoleMemberAdd()
   - RlsAdd()

   With certain logic to iterate over Role Members and RLS assignments.

   Now, it is execute time:

i. In SSMS, delete the 'AdventureWorks Tabular AMO 2012' database.

ii. Open the solution; if it is not opened.

iii. At the beginning of the code, just before all the ‘using’ statements.

   Remove the comment mark from: // #define localUsersCreated

iv. Set a breakpoint at: string roleName = "Admins";

   This is the first instruction in the Add Roles, Users and RLS region.

v. Press F5 and let the code run to the breakpoint.

vi. Switch to SSMS and refresh your list of AS databases.

vii. Right click on the database and select Process Database. Do a Full process on the database.

viii. Right click on the database and select New Query, MDX

ix. Try to run the following query: evaluate row("Reseller Total Sales", [Reseller Total Sales], "Total Product Inventory Value to last day", [Total Inventory Value])

x. You should get these values: 80450596.9823   23603975.57

xi. Set a new breakpoint at: AMO2TabularDb.Process(AMO.ProcessType.ProcessDefault);

   This is the first instruction in the Process newly created database region.

xii. Next you will open several instances of SSMS; each running under a different user, one for each of the above created ones.

xiii. Get a copy of the path of the executable for SSMS. It should be something like:

   a. Click Start

b. Select All Programs

c. Select Microsoft SQL Server 2012

d. Right click on SQL Server Management Studio, select properties.

e. Copy the contents of Target

xiv. Open a Command prompt.

xv. Type the following command; replacing <username> with the appropriate user from the list of users you recently created and <pathToSsms> with the path you also found:

```
RunAs /user:<username> "<pathToSsms>"
```

xvi. You'll be asked to type the password for each user after you press the [Enter] key. Note, there is no feedback from the console while you are typing the password.

xvii. On the Connect to Server window, make sure the server type is Analysis Services and the Server Name is your current machine or Localhost or '.' (A single period without the quotes). You could also check the connecting user is the one for which you are executing the RUNAS command.

xviii. Expand databases and you should not see anything, with the exception of AmoAdmin.

In Tabular models, users are not granted ReadDefinition permissions; hence, users cannot query metadata.

xix. Click the New MDX query button from the bar and accept the login with the given user.

Make sure that you are connected to the 'AdventureWorks Tabular AMO 2012' database; verify it in the Available Databases Selector (it is the small dropdown list box with database names).

Why is it that you now can see 'metadata' in the left pane of the MDX query windows; but, you couldn't see anything in the explorer pane?

Because, the MDX window uses rowset queries to populate its objects; technically you are not querying metadata, you are getting data.

The above is true for all users that have Read access granted; this means, in the case of our sample, that TestUser and AmoOperator cannot see anything in the MDX pane because both of them have ReadAccess set to false. However, AmoOperator can issue process statements (in XMLA), while all other users cannot (with the exception of users with administrative rights). See appendix 'XMLA Process Command' if you would like to try the command

xx. Switch to open SSMS, running under lynn0 credentials, and execute: evaluate row("Reseller Total Sales", [Reseller Total Sales], "Total Product Inventory Value to last day", [Total Inventory Value])

xxi. You should get this value: 1421810.92519999 (empty here)

xxii. Instead of the values you have obtained before:

80450596.9823 23603975.57

xxiii. The reason is simple; lynn0 has access to her sales and the sales of any employee that has her in the line of managers, and she has no access to the 'Product Inventory' data by RLS.

xxiv. Let's check with syed0, she is lynn0's manager. Run the same query for syed0. You should obtain:

1594335.37669999   (empty here)

A little bit higher than lynn0

xxv. If you check with ed0; you should get

(empty here)   23603975.57

Because ed0 works in production, he has access to 'Product Inventory' data; but no access to 'Reseller Sales'

V. Appendixes

A. Results for database verification

Date[Calendar Year]	Product Category[Product Category Name]	[Total Sales]	[Total Inventory Value]
2005	Bikes	10661722.28	4841049.41
2006	Bikes	26486358.2	4817043.56
2007	Bikes	34910877.69	4814274.31
2008	Bikes	22561568.03	4989538.32
2009	Bikes		
2010	Bikes		
2005	Components	615474.9788	18300164.46
2006	Components	3610092.472	18353238.28
2007	Components	5482497.289	18173339.44
2008	Components	2091011.918	18536633.51
2009	Components		
2010	Components		
2005	Clothing	34376.3353	3376.2
2006	Clothing	485587.1546	3471.76
2007	Clothing	1010112.157	3069.98
2008	Clothing	587537.8026	3443.52
2009	Clothing		
2010	Clothing		
2005	Accessories	20235.3646	34387.48
2006	Accessories	92735.3534	34728.86
2007	Accessories	590242.5866	30733.83
2008	Accessories	568844.5832	31385.19

```
 2009            Accessories

 2010            Accessories

 2005                                                            40771.46

 2006                                                            41363.77

 2007                                                            42393.1

 2008                                                            42975.03

 2009

 2010
```

B. XMLA Process Command

```
<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

  <Parallel>

    <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <Object>

        <DatabaseID>AdventureWorks Tabular AMO 2012</DatabaseID>

      </Object>

      <Type>ProcessFull</Type>

    </Process>

  </Parallel>

</Batch>
```