Microsoft

Automated Partition Management for Analysis Services Tabular Models

Microsoft BI Technical Article

Writer: Christian Wade, Senior Program Manager, Microsoft Corp.

Contributor: Owen Duncan, Senior Content Developer, Microsoft Corp.

Published: November 2016

Applies to: Microsoft SQL Server 2016 Analysis Services, Microsoft Azure Analysis Services

Summary: This whitepaper and associated samples describe partition management automation by using the Tabular Object Model (TOM).

Contents

Introduction

Analysis Services tabular models can store data in a highly-compressed, in-memory cache for optimized query performance. This provides fast user interactivity over large data sets.

Large datasets normally require table partitioning to accelerate and optimize the data-load process. Partitioning enables incremental loads, increases parallelization, and reduces memory consumption. The Tabular Object Model (TOM) serves as an API to create and manage partitions. TOM was released with SQL Server 2016 and is discussed in more detail here. Model Compatibility Level 1200 or above is required.

This document describes how to use the AsPartitionProcessing TOM code sample for automated partition management with minimal code changes.

The sample,

- Is intended to be generic and configuration driven.
- Works for both Azure Analysis Services and SQL Server Analysis Services tabular models.
- Can be leveraged in many ways including from an SSIS script task, Azure Functions and others.

Note: Loading data into the in-memory cache is often referred to as *processing*. This terminology is used by this document.

Partitioning Strategy & Assumptions

Rolling-window pattern

AsPartitionProcessing follows the rolling-window pattern, which is common in traditional Analysis Services implementations. The data is kept within a predefined date range and incremented as necessary. This maintains memory usage within a predictable range over time.

Partition granularity

Yearly, monthly and daily partition granularities are supported. Partition granularity is defined per table in the configuration.

Parallelization

Initial setup processing is sequential. Incremental processing can be performed in parallel.

Initial setup will create and process the partitions for the first time based on the configuration. This is performed one partition at a time to avoid running out of memory (data is not fully compressed during processing). For a large data set, the initial load may typically take a few hours depending on factors such as the query performance of the source system.

Incremental processing can be configured to execute as a fully parallelized operation for all tables within a model. It can also be configured to process different tables one at a time. The reason to consider processing one table at a time is again to work within memory constraints. When processing multiple partitions within a single table, they are always done in parallel.

Online & offline processing

Incremental processing can be performed as an online operation, or offline for less memory usage; it is configuration driven. Online incremental processing requires a copy of the data to be maintained in memory for queries until the new data is ready, and then switches to the new data.

Configuration and logging database

Traditional Analysis Services implementations that require partitioning often use a configuration and logging database. AsPartitionProcessing is meant to work in this way, although this is optional. It can be set up to log messages to other targets. This enables easy partition configuration, and diagnosis of issues resulting from automated processing operations.

Date key format

Date keys in source table are assumed to be integers formatted as yyyymmdd, which is common for data warehouses and marts. If this is not available, it should be possible to derive such a column in a database view.

Getting Started

Requirements

Before you get started, you'll need these tools:

SQL Server 2016 with latest service pack - Install the database engine and SSAS in tabular mode. You can download and install the free SQL Server 2016 Developer Edition here.

SQL Server Data Tools – Download and install the latest version here.

SQL Server Management Studio - Download and install the latest version here.

Visual Studio 2015 – Download and install the free Community Edition here.

AsPartitionProcessing Solution

Get the AsPartitionProcessing solution here: https://github.com/Microsoft/AsPartitionProcessing.

1. Open the solution in Visual Studio and build the project. The hint path for the client library DLLs is the following (assuming installation is on C:\ drive):
   C:\Program Files (x86)\Microsoft SQL Server\130\SDK\Assemblies
2. Ensure AsPartitionProcessing.SampleClient is set as the startup project.

AdventureWorks

The quickest way to understand the code sample is to run it on the AdventureWorksDW sample database. The backup file, AdventureWorksDW.bak, is included in the solution.

The tabular project, AsPartitionProcessing.AdventureWorks**,** is also provided in the solution. It should be used instead of the version from CodePlex because partitioning has been removed from the Internet Sales and Reseller Sales tables. Instead, these tables each have a single partition with the same name as the table, which is the default when you create a new table in SSDT. This partition acts as the *template partition* used by the AsPartitionProcessing sample.

Deploy and process the AdventureWorks tabular model.

SampleClient

AsPartitionProcessing.SampleClient is a console application with a reference to the AsPartitionProcessing class library. It can easily be converted to work for customer projects. Alternatively, it provides sample client code to execute from an SSIS package, Azure Function, or other mechanism.

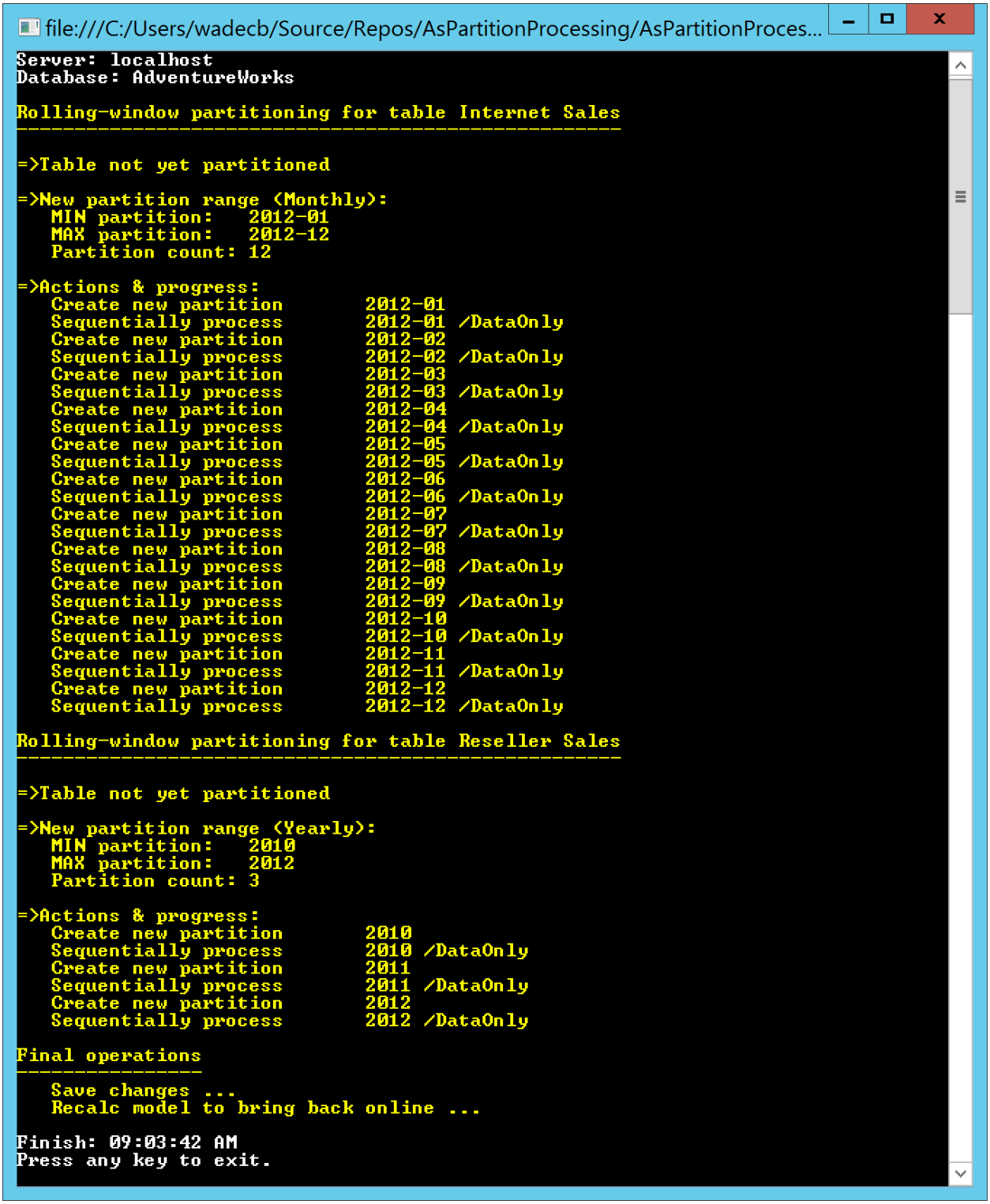
Open Program.cs. Note that `UseDatabase = false`. This means the InitializeAdventureWorksInline method will be executed to initialize parameters.

```
PartitionedModelConfig partitionedModel = new PartitionedModelConfig(
    partitionedModelConfigID: 1,
    analysisServicesServer: "localhost",
    analysisServicesDatabase: "AdventureWorks",
    initialSetUp: true,
    incrementalOnline: true,
    incrementalParallelTables: true,
    integratedAuth: true,
    userName: "",
    password: "",
    partitionedTables:
    new List<PartitionedTableConfig>
    {
        new PartitionedTableConfig(
            partitionedTableConfigID: 1,
            maxDate: Convert.ToDateTime("2012-12-01"),
            granularity: Granularity.Monthly,
            numberOfPartitionsFull: 12,
            numberOfPartitionsForIncrementalProcess: 3,
            analysisServicesTable: "Internet Sales",
            sourceTableName: "[dbo].[FactInternetSales]",
            sourcePartitionColumn: "OrderDateKey"
        ),
        new PartitionedTableConfig(
            partitionedTableConfigID: 2,
            maxDate: Convert.ToDateTime("2012-12-01"),
            granularity: Granularity.Yearly,
            numberOfPartitionsFull: 3,
            numberOfPartitionsForIncrementalProcess: 1,
            analysisServicesTable: "Reseller Sales",
            sourceTableName: "[dbo].[FactResellerSales]",
            sourcePartitionColumn: "OrderDateKey"
        )
    }
);
```

Place a breakpoint at the Main method and step through the code in the AsPartitionProcessing.SampleClient project to understand how to interact with the methods exposed by the class library. The PerformProcessing method is the key method.

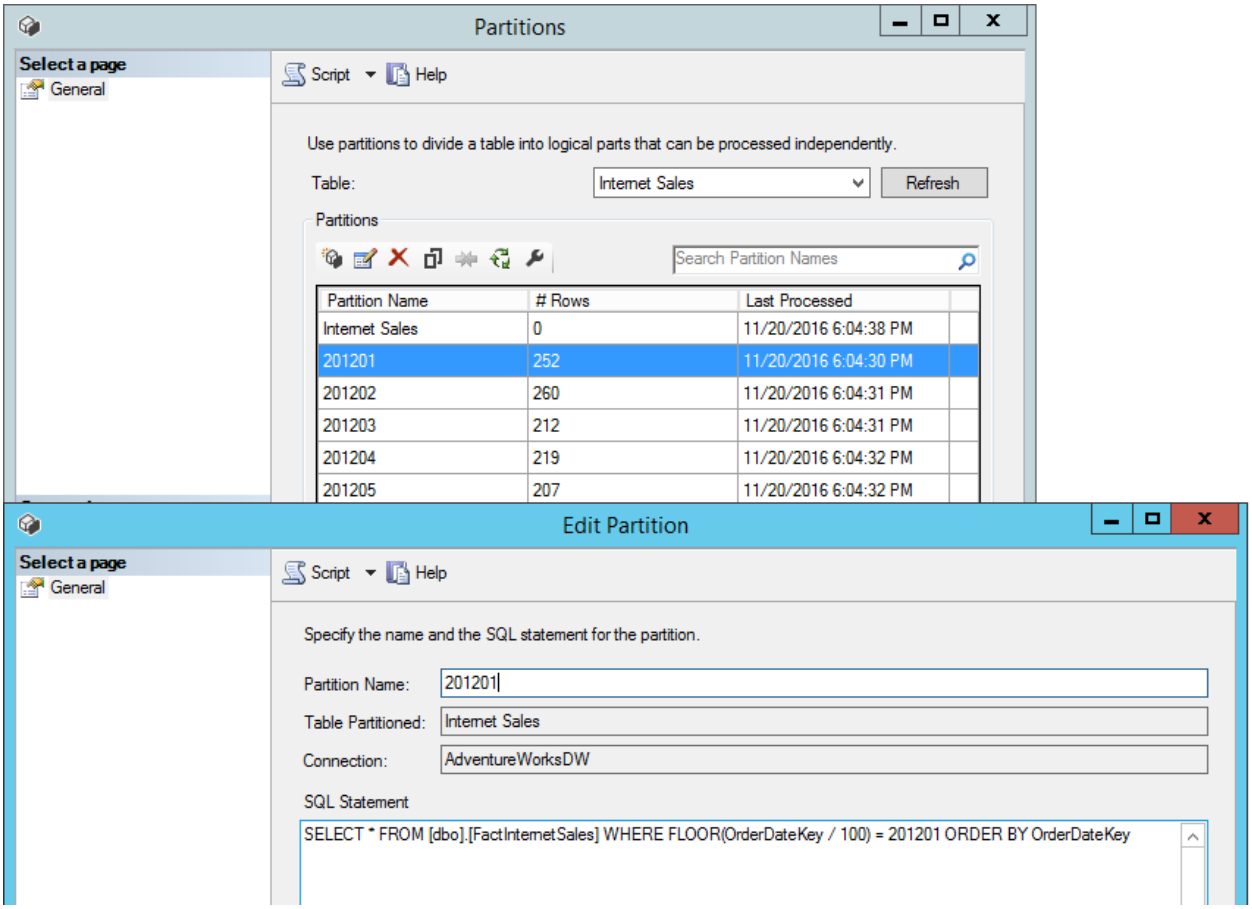
```
PartitionProcessor.PerformProcessing(modelConfig, LogMessage);
```

The console output should be displayed like this:



Use SSMS to inspect the partitions created. Partition source queries take the simple form:

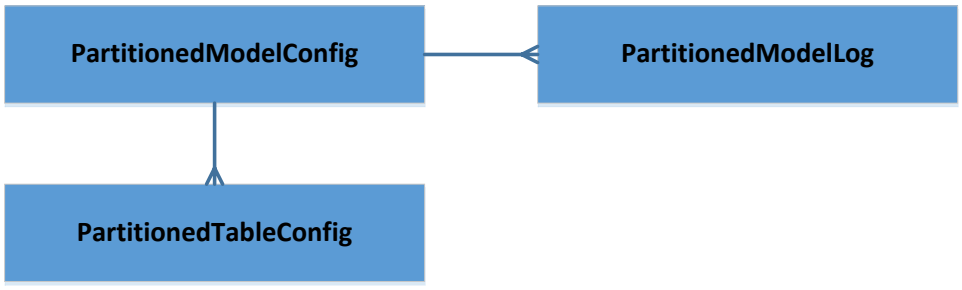
SELECT * FROM <*source table*> WHERE <*partition filter*>



Configuration & Logging Database

Typically, partitioning configuration and logging is done using a database. AsPartitionProcessing contains the CreateDatabaseObjects.sql script to create the necessary tables, and contains the methods for reading and writing to the database.

Data Model



PartitionedModelConfig

Configuration information for a partitioned AS tabular model:

Column	Description
PartitionedModelConfigID	Primary key.
AnalysisServicesServer	Name of the Analysis Services instance. Can be SSAS or an Azure AS URL.
AnalysisServicesDatabase	Name of the Analysis Services database.
InitialSetUp	True for initial set up to create partitions and process them sequentially. False for incremental processing. See Partitioning Strategy & Assumptions section above for more information.
IncrementalOnline	When initialSetUp=false, determines if processing is performed as an online operation, which can require more memory, but allows users to query the model during processing. True to keep the model online (process Full). See Partitioning Strategy & Assumptions section above for more information.
IncrementalParallelTables	When initialSetUp=false, determines if separate tables are processed in parallel. Note: partitions within a table are always processed in parallel. True to process tables in parallel. See Partitioning Strategy & Assumptions section above for more information.
IntegratedAuth	Should always be set to true for SSAS implementations that will run under the current process account. For Azure AS, normally set to false.
UserName	Only applies when integratedAuth=false. Can be used for Azure AD UPNs to connect to Azure AS.
Password	Only applies when integratedAuth=false. Can be used for Azure AD UPNs to connect to Azure AS.

PartitionedTableConfig

Configuration information for a partitioned table within an AS tabular model:

Column	Description
PartitionedTableConfigID	Primary key.
PartitionedModelConfigID	Foreign key to PartitionedModelConfig table.
MaxDate	The maximum date that needs to be accounted for in the partitioned table. Represents the upper boundary of the rolling window.
Granularity	Partition granularity, which can be Yearly, Monthly or Daily. Daily = 0,

	Monthly = 1, Yearly = 2
NumberOfPartitionsFull	Count of all partitions in the rolling window. For example, a rolling window of 10 years partitioned by month would require 120 partitions.
NumberOfPartitionsForIncrementalProcess	Count of *hot partitions* where the data can change. For example, it may be necessary to refresh the most recent 3 months of data every day. This only applies to the most recent partitions.
AnalysisServicesTable	Name of the partitioned table in the tabular model.
SourceTableName	Name of the source table in the relational database.
SourcePartitionColumn	Name of the source column from the table in the relational database.

PartitionedModelLog

Log of partitioning execution:

Column	Description
PartitionedModelLogID	Primary key.
PartitionedTableConfigID	Foreign key to PartitionedTableConfig table.
ExecutionID	GUID generated for the execution run.
LogDateTime	Date and time the message was logged.
Message	The log message.

Sample Configuration

The SampleConfiguration.sql script initializes the configuration for AdventureWorks. The script can be modified for use in customer implementations. Execute the script to initialize the database.

```
INSERT INTO [dbo].[PartitionedModelConfig]
VALUES(
     1                          --[PartitionedModelConfigID]
    ,'localhost'                --[AnalysisServicesServer]
    ,'AdventureWorks'           --[AnalysisServicesDatabase]
    ,1                          --[InitialSetUp]
    ,1                          --[IncrementalOnline]
    ,1                          --[IncrementalParallelTables]
    ,1                          --[IntegratedAuth]
);

INSERT INTO [dbo].[PartitionedTableConfig]
VALUES(
     1                          --[PartitionedTableConfigID]
    ,1                          --[PartitionedModelConfigID]
    ,'2012-12-01'               --[MaxDate]
    ,1                          --[Granularity]   1=Monthly
    ,12                         --[NumberOfPartitionsFull]
    ,3                          --[NumberOfPartitionsForIncrementalProcess]
    ,'Internet Sales'           --[AnalysisServicesTable]
    ,'[dbo].[FactInternetSales]'--[SourceTableName]
```

```
    ,'OrderDateKey'             --[SourcePartitionColumn]
),
(
     2                          --[PartitionedTableConfigID]
    ,1                          --[PartitionedModelConfigID]
    ,'2012-12-01'               --[MaxDate]
    ,2                          --[Granularity]   2=Yearly
    ,3                          --[NumberOfPartitionsFull]
    ,1                          --[NumberOfPartitionsForIncrementalProcess]
    ,'Reseller Sales'           --[AnalysisServicesTable]
    ,'[dbo].[FactResellerSales]'--[SourceTableName]
    ,'OrderDateKey'             --[SourcePartitionColumn]
);
```

Database connection info

Connection information to the configuration and logging database can be set in App.config in the userSettings section.

```
<userSettings>
    <AsPartitionProcessing.SampleClient.Settings>
        <setting name="ConfigServer" serializeAs="String">
            <value>localhost</value>
        </setting>
        <setting name="ConfigDatabase" serializeAs="String">
            <value>AsPartitionProcessing</value>
        </setting>
        <setting name="ConfigDatabaseIntegratedAuth" serializeAs="String">
            <value>True</value>
        </setting>
    </AsPartitionProcessing.SampleClient.Settings>
</userSettings>
```

Test Different Configurations

In this section, we will update the configuration, execute the sample, and view the log messages.

In Program.cs, change the `UseDatabase` constant to be assigned `true`.

Execute the SampleClient application, and the log query. If the SampleClient application was previously run from the Getting Started section, messages will be shown saying the partitions already exist and are already processed.

Latest execution log query

In addition to the console output, the following query on the configuration and logging database shows the execution results. This can be used to test the different configurations below.

```
SELECT [Message]
FROM [dbo].[PartitionedModelLog]
WHERE ExecutionID =
(    SELECT [ExecutionID] FROM [dbo].[PartitionedModelLog]
     WHERE [LogDateTime] = (SELECT MAX([LogDateTime]) FROM [dbo].[PartitionedModelLog])
)
ORDER BY [LogDateTime]
```

Incremental mode

Run the following UPDATE statement to switch to incremental mode:

```
UPDATE [dbo].[PartitionedModelConfig] SET [InitialSetUp] = 0
```

Execute SampleClient application, and the log query. The following results should be shown. Only the specified number of most recent partitions are processed as an online operation.

```
Start: 12:41:02 PM
Server: localhost
Database: AdventureWorks

Rolling-window partitioning for table Internet Sales
----------------------------------------------------

=>Current partition range (Monthly):
   MIN partition:   2012-01
   MAX partition:   2012-12
   Partition count: 12

=>New partition range (Monthly):
   MIN partition:   2012-01
   MAX partition:   2012-12
   Partition count: 12

=>Actions & progress:
   Parallel process partition 2012-10 /Full
   Parallel process partition 2012-11 /Full
   Parallel process partition 2012-12 /Full

Rolling-window partitioning for table Reseller Sales
----------------------------------------------------

=>Current partition range (Yearly):
   MIN partition:   2010
   MAX partition:   2012
   Partition count: 3

=>New partition range (Yearly):
   MIN partition:   2010
   MAX partition:   2012
   Partition count: 3

=>Actions & progress:
   Parallel process partition 2012 /Full

Final operations
----------------
Save changes ...

Finish: 12:41:07 PM
```

Increment partition range

Run the following UPDATE statement to increment the partition range by one period.

```
UPDATE [dbo].[PartitionedTableConfig] SET [MaxDate] = '2013-01-01'
```

Execute SampleClient application, and the log query. The following results should be shown. The oldest partition is removed from both tables, a new one is added and the most recent partitions are processed.

```
Start: 12:47:26 PM
Server: localhost
Database: AdventureWorks

Rolling-window partitioning for table Internet Sales
----------------------------------------------------

=>Current partition range (Monthly):
   MIN partition:   2012-01
   MAX partition:   2012-12
   Partition count: 12

=>New partition range (Monthly):
   MIN partition:   2012-02
   MAX partition:   2013-01
   Partition count: 12

=>Actions & progress:
   Remove old partition       2012-01
   Parallel process partition 2012-11 /Full
   Parallel process partition 2012-12 /Full
   Create new partition       2013-01
   Parallel process partition 2013-01 /Full

Rolling-window partitioning for table Reseller Sales
----------------------------------------------------

=>Current partition range (Yearly):
   MIN partition:   2010
   MAX partition:   2012
   Partition count: 3

=>New partition range (Yearly):
   MIN partition:   2011
   MAX partition:   2013
   Partition count: 3

=>Actions & progress:
   Remove old partition       2010
   Create new partition       2013
   Parallel process partition 2013 /Full

Final operations
----------------
Save changes ...

Finish: 12:47:34 PM
```

Offline processing

Run the following UPDATE statement to perform offline processing to potentially use less memory during processing.

```
UPDATE [dbo].[PartitionedModelConfig] SET [IncrementalOnline] = 0
```

Execute SampleClient application, and the log query. The following results should be shown. The partitions are processed using RefreshType of DataOnly, and a Recalc operation is performed to bring the model back online.

```
Start: 12:53:19 PM
Server: localhost
Database: AdventureWorks

Rolling-window partitioning for table Internet Sales
----------------------------------------------------

=>Current partition range (Monthly):
   MIN partition:   2012-02
   MAX partition:   2013-01
   Partition count: 12

=>New partition range (Monthly):
   MIN partition:   2012-02
   MAX partition:   2013-01
   Partition count: 12

=>Actions & progress:
   Parallel process partition 2012-11 /DataOnly
   Parallel process partition 2012-12 /DataOnly
   Parallel process partition 2013-01 /DataOnly

Rolling-window partitioning for table Reseller Sales
----------------------------------------------------

=>Current partition range (Yearly):
   MIN partition:   2011
   MAX partition:   2013
   Partition count: 3

=>New partition range (Yearly):
   MIN partition:   2011
   MAX partition:   2013
   Partition count: 3

=>Actions & progress:
   Parallel process partition 2013 /DataOnly

Final operations
----------------
Save changes ...
Recalc model to bring back online ...

Finish: 12:53:27 PM
```

Sequential table processing

Run the following UPDATE statement to process tables sequentially for less memory usage. This setting will take longer to process the model.

```
UPDATE [dbo].[PartitionedModelConfig] SET [IncrementalParallelTables] = 0
```

Execute SampleClient application, and the log query. The following results should be shown. Changes are saved for each table individually before moving onto the next table.

```
Start: 01:04:25 PM
Server: localhost
Database: AdventureWorks

Rolling-window partitioning for table Internet Sales
----------------------------------------------------

=>Current partition range (Monthly):
   MIN partition:   2012-02
   MAX partition:   2013-01
   Partition count: 12

=>New partition range (Monthly):
   MIN partition:   2012-02
   MAX partition:   2013-01
   Partition count: 12

=>Actions & progress:
   Parallel process partition 2012-11 /DataOnly
   Parallel process partition 2012-12 /DataOnly
   Parallel process partition 2013-01 /DataOnly
   Save changes for table Internet Sales ...

Rolling-window partitioning for table Reseller Sales
----------------------------------------------------

=>Current partition range (Yearly):
   MIN partition:   2011
   MAX partition:   2013
   Partition count: 3

=>New partition range (Yearly):
   MIN partition:   2011
   MAX partition:   2013
   Partition count: 3

=>Actions & progress:
   Parallel process partition 2013 /DataOnly
   Save changes for table Reseller Sales ...

Final operations
----------------
Recalc model to bring back online ...

Finish: 01:04:33 PM
```

Other Options & Considerations

Custom Logging

The LogMessage method in Program.cs is passed as a delegate into the PerformProcessing method, so it can easily be changed for custom logging requirements.

```
private static void LogMessage(string message, PartitionedModelConfig partitionedModel)
{
    //Can provide custom logging code here
    ...
```

Granularity

Mixed granularity

Mixed granularity for a single table may be useful for scenarios such as near-real time refresh of current-day data, coupled with historical data at a higher grain. This can be achieved by setting up multiple table configurations for the same table. However, this adds complexity to maintenance of configuration date ranges, so it may be better to keep all partitions at the lowest required grain.

Switching granularity

If the granularity of an existing partitioned table is changed, the sample does not delete the old partitions with the old granularity. In this case, it is necessary to manually delete all the existing partitions except the template partition, which has the same name as the table. This is by design to avoid inadvertently deleting data.

Fragmentation

Partitioned tables may suffer from fragmentation over time. Defragmentation can be performed as a processing operation using the RefreshType.Defragmentation enumeration. Detailed discussion and code samples for defragmentation is outside the scope of this document.

Model Deployment

When deploying new versions of partitioned tabular models that already exist on the target environment, it is necessary to be aware of the partitioning process already in-place. As shown by this code sample, partitions are normally created and managed by a separate process. This means the version of the tabular model from source control does not contain the partitions. A simple deployment process such as right-click, Deploy from SSDT will lose the partitions and all the data within them. Two deployment tools that support retaining partitions are BISM Normalizer and the Analysis Services Deployment Wizard. Both these tools support command-line execution for automated deployment. Detailed discussion on this topic including the pros and cons of these tools is outside the scope of this document.

AsPerfMon

Another code sample that may be useful in conjunction with AsPartitionProcessing is the AsPerfMon tool.

Get AsPerfMon here: https://github.com/Microsoft/AsPerfMon.

AsPerfMon can be used to check real-time memory usage during processing. It splits memory usage by database, which is informative when multiple databases share the same server.

This is especially useful for Azure AS since you can't use Task Manager or create Performance Monitor counters. Similar functionality is provided by the Metrics section in the control blade for an Azure AS server in the Azure Portal. By using Metrics, you can check usage for the past day or week. AsPerfMon is for real-time monitoring during processing.

It works by polling the DISCOVER_OBJECT_MEMORY_USAGE Data Management View. This view is representative and very useful, but in some rare cases may double count allocations, so not always 100% accurate.