Microsoft Dynamics AX 2012 R3

Implementing and deploying Transportation management engines

White paper

Pawel Kruk

May 2014

This document describes the typical steps for building Transportation management (TMS) engines and utilizing them in Microsoft Dynamics AX.

www.microsoft.com/dynamics/ax

Send feedback.

Contents

Implementing and deploying Transportation management engines

The Transportation management (TMS) module includes a number of extension points that let you implement custom algorithms to perform tasks that are related to the rating of transport and freight reconciliation. The implementations of the algorithms are called Transportation management engines (also referred to as TMS engines or engines). The engines are delivered as implementations of specific .NET interfaces and deployed on the Microsoft Dynamics AX Application Object Server (AOS) tier. Each Transportation management engine can be switched on and off, and it can also be tuned at runtime, based on Microsoft Dynamics AX data. Some of the most important objectives of these engines are as follows:

- Calculation of transportation rate
- Calculation of travel distance from point to point
- Calculation of the time it takes to travel from point to point
- Zone identification of addresses
- Distribution of transportation charges for shipments across source document lines (also referred to as apportionment of charges).

Microsoft Dynamics AX 2012 R3 includes a number of fully functional engines that are available out of the box. However, in many cases, new engines might be required in order to satisfy contract requirements. For example, there is no engine that calculates freight charge by using a particular algorithm, or an engine might be required to retrieve rating data directly from a web service provided by the carrier.

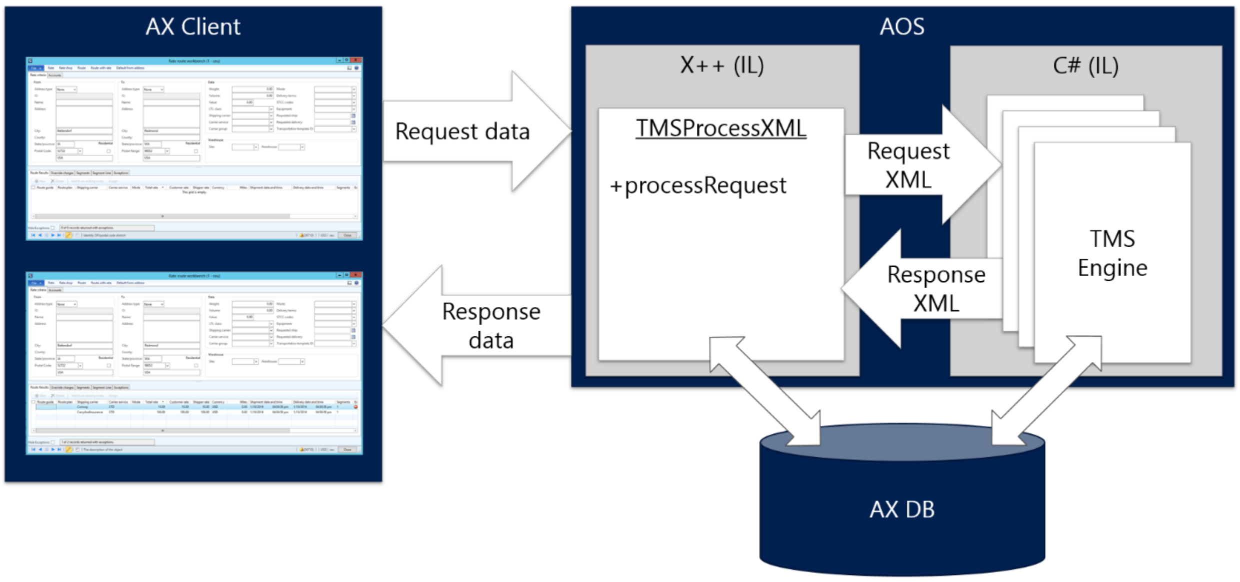
This document explains the typical steps for building Transportation management engines and utilizing them in Microsoft Dynamics AX. It is targeted to engineers who want to learn how to implement and deploy custom Transportation management engines.

Prerequisites

Microsoft Visual Studio Tools for Microsoft Dynamics AX must be installed. For more information, see http://technet.microsoft.com/en-us/library/gg889157.aspx.

Architectural background

The following illustration shows a simplified view of the TMS system.



Within TMS, a number of operations might require some kind of data processing that is specific to a particular carrier, such as transportation rate calculation. Typically, this kind of calculation requires a lot of input data, such as the origin and delivery addresses, the size, weight, and number of packages, and the requested delivery date. For a rate shopping operation, you can track this information from the **Rate route workbench** form. When you initiate a rate shopping request, request XML is constructed in TMS by using one of the X++ classes that are derived from **TMSProcessXML_Base**. The request XML is passed to the processing system encapsulated in the .NET assembly named **Microsoft.Dynamics.Ax.TMS** (also referred to as the TMS managed system). The further processing involves instantiation and utilization of one or more Transportation management engines. The final response from the TMS managed system consists of XML, which is interpreted into a result that is persisted in the Microsoft Dynamics AX database.

The source code of the TMS managed system is available in the Microsoft Dynamics AX Application Object Tree (AOT), under the following path: \Visual Studio Projects\C Sharp Projects\Microsoft.Dynamics.AX.Tms.

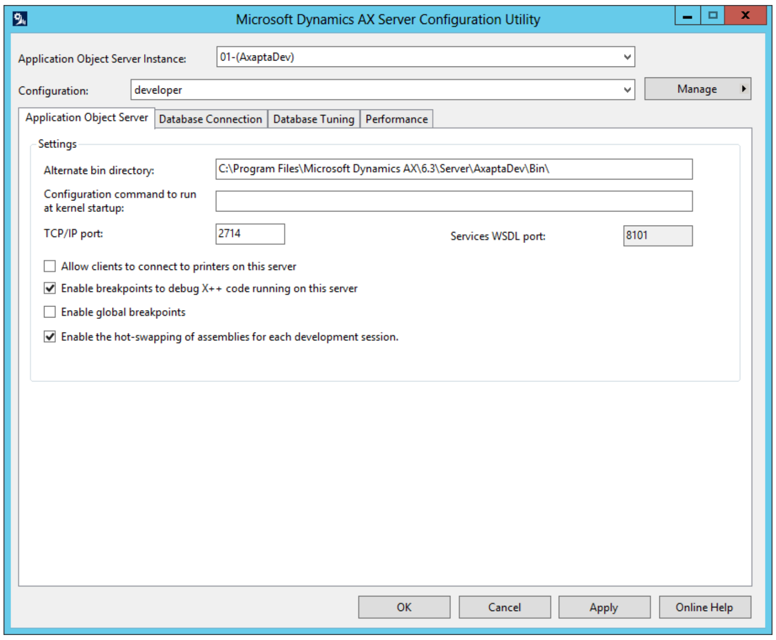
All of the engines that are available out of the box in AX 2012 R3 are defined within the TMS managed system itself. We recommend that all custom engines be implemented in a stand-alone assembly. This assembly should be constructed by using Visual Studio Tools for Microsoft Dynamics AX, and its project should be hosted in the AOT. The Microsoft Dynamics AX server infrastructure ensures that the actual project output is deployed to a server-binary location upon AOS startup. Microsoft Dynamics AX models are the recommended vehicles for distributing the TMS engine implementations to customers. For more information about Microsoft Dynamics AX models, see http://technet.microsoft.com/en-us/library/hh335184.aspx.

Tutorial: Construct a Hello-World rate engine

Follow these steps to implement a simple rate engine.

1. Follow these steps to enable debugging and hot-swapping of assemblies on the Microsoft Dynamics AX server:
    1. Open the Microsoft Dynamics AX Server Configuration Utility.
    2. Create a new configuration.
    3. Select the **Enable breakpoints to debug X++ code running on this server** and **Enable the hot-swapping of assemblies for each development session** check boxes.

       The following illustration shows the new configuration.

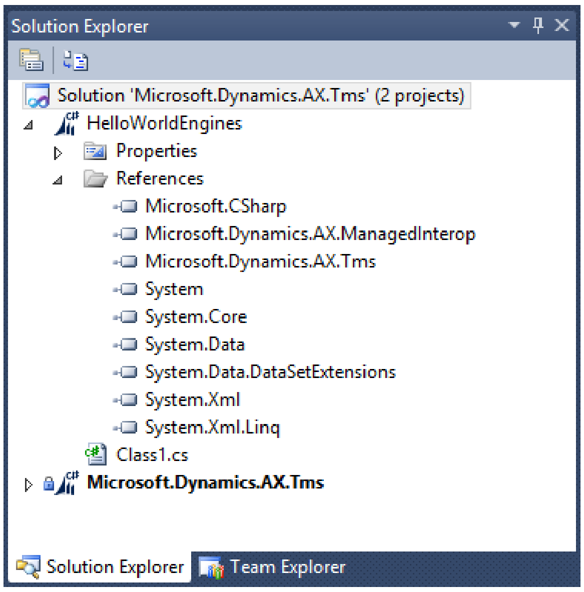


Application Object Server Instance: 01-(AxaptaDev)
Configuration: developer
Alternate bin directory: C:\Program Files\Microsoft Dynamics AX\6.3\Server\AxaptaDev\Bin\
Configuration command to run at kernel startup:
TCP/IP port: 2714
Services WSDL port: 8101
☐ Allow clients to connect to printers on this server
☑ Enable breakpoints to debug X++ code running on this server
☐ Enable global breakpoints
☑ Enable the hot-swapping of assemblies for each development session.

    4. Click **OK** to restart the AOS service.
2. From the AOT, follow these steps to open the source of Microsoft.Dynamics.Ax.TMS in a new instance of Visual Studio:
    1. Navigate to \Visual Studio Projects\C Sharp Projects\Microsoft.Dynamics.AX.Tms.
    2. On the context menu, click **Edit** to open Visual Studio.
3. Follow these steps to add a new C# Class Library project to the solution:
    1. In Solution Explorer, right-click your solution node, and then click **Add** > **New Project**.
    2. In the C# project templates, select **Class Library**.

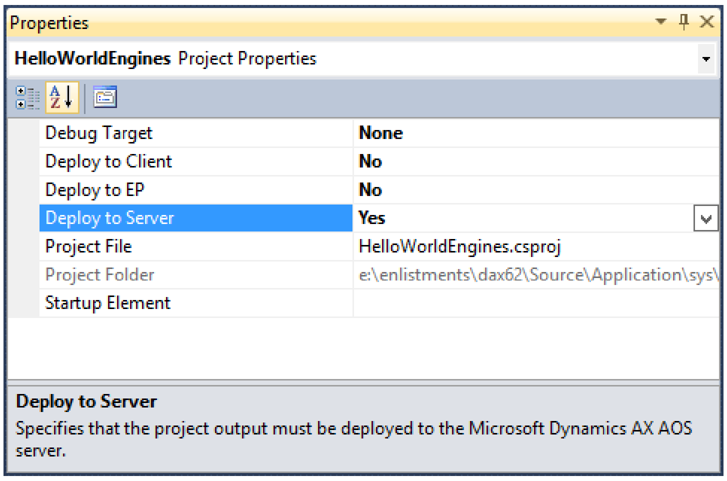
   3. Enter **HelloWorldEngines** as the project name.
   4. Click **OK**.
4. In Solution Explorer, right-click the **HelloWorldEngines** project node, and then click **Add HelloWorldEngines to AOT**.
5. Follow these steps to add a project-to-project reference from your project to Microsoft.Dynamics.AX.Tms:
   1. In Solution Explorer, right-click the **HelloWorldEngines** project node, and then click **Add Reference**.
   2. On the **Projects** tab, select **Microsoft.Dynamics.AX.Tms**.
   3. Click **OK**.

   The following illustration shows what your solution should now look like in Solution Explorer.


Solution Explorer
Solution 'Microsoft.Dynamics.AX.Tms' (2 projects)
HelloWorldEngines
Properties
References
Microsoft.CSharp
Microsoft.Dynamics.AX.ManagedInterop
Microsoft.Dynamics.AX.Tms
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Xml
System.Xml.Linq
Class1.cs
Microsoft.Dynamics.AX.Tms
Solution Explorer
Team Explorer

6. Follow these steps to enable deployment of the project output to the Microsoft Dynamics AX server:
   1. In Solution Explorer, select the project node.
   2. In the **Properties** window, set the **Deploy to Server** property to **Yes**.

   The following illustration shows the project properties after this change has been made.



Debug Target	None
Deploy to Client	No
Deploy to EP	No
Deploy to Server	Yes
Project File	HelloWorldEngines.csproj
Project Folder	e:\enlistments\dax62\Source\Application\sys\
Startup Element	

Deploy to Server
Specifies that the project output must be deployed to the Microsoft Dynamics AX AOS server.

7. Follow these steps to implement a rate engine called HelloWorldRateEngine:
   1. In Solution Explorer, rename Class1.cs to **HelloWorldRateEngine.cs**.
   2. Implement the **HelloWorldRateEngine** class.

      The following example shows how to implement this class.

```
namespace HelloWorldEngines
{
    using System;
    using System.Xml.Linq;
    using Microsoft.Dynamics.Ax.Tms;
    using Microsoft.Dynamics.Ax.Tms.Bll;
    using Microsoft.Dynamics.Ax.Tms.Data;
    using Microsoft.Dynamics.Ax.Tms.Utility;

    /// <summary>
    /// Sample rate engine class using rating formula of quantity * factor.
    /// </summary>
    public class HelloWorldRateEngine : BaseRateEngine
    {
        private const string RATE_FACTOR = "RateFactor";
        private decimal rateFactor;

        /// <summary>
        /// Initializes the engine instance.
        /// </summary>
        /// <param name="rateEngine">Rate engine setup record.</param>
        /// <param name="ratingDto">Rating data transfer object.</param>
        public override void Initialize(
            TMSRateEngine rateEngine,
            RatingDto ratingDto)
        {
            base.Initialize(rateEngine, ratingDto);
            RateEngineParameters parameters =
                new RateEngineParameters(TMSEngine.RateEngine,
                    rateEngine.RateEngineCode);

            // Try to retrieve decimal value of RateFactor parameter specified
            // on engine setup Parameters
            if (!Decimal.TryParse(parameters.RetrieveStringValue(RATE_FACTOR),
                out rateFactor))
            {
                // Throw TMS Exception. The exception message is shown in infolog.
                // Additional exception data is recorded in
                // "Transportation system error log"
                throw TMSException.Create(
                    "HelloWorldRateEngine requires definition of RateFactor parameter with valid decimal value",
                    TMSExceptionType.TMSEngineSetupException);
            }
        }

        /// <summary>
        /// Calculates rate.
        /// </summary>
        /// <param name="transactionFacade">The request transaction facade.</param>
        /// <param name="shipment">Rated shipment element.</param>
        /// <param name="rateMasterCode">Rate master code.</param>
        /// <returns>Updated rating data transfer object.</returns>
        public override RatingDto Rate(
            TransactionFacade transactionFacade,
            XElement shipment,
            string rateMasterCode)
        {
            // Use extension method to sum down the item
            // quantity from the shipment XML element
            decimal quantity = shipment.SumDown(ElementXmlConstants.Quantity);

            // Retrieve or create rating element
            XElement rateEntity = shipment.RetrieveOrCreateRatingEntity(this.RatingDto);

            // Use extension method to record rate
            // This method does not record additional information
            // like unit counts, currency etc.
            rateEntity.AddRate(TmsRateType.Rate, quantity * rateFactor);
            return this.RatingDto;
        }
    }
}
```

8. In Solution Explorer, select the **HelloWorldEngines** project node, and then click **Add HelloWorldEngines to AOT**.
9. Follow these steps to deploy the engine assembly:
    1. In Solution Explorer, right-click your project node, and then click **Deploy**.
    2. Restart AOS.

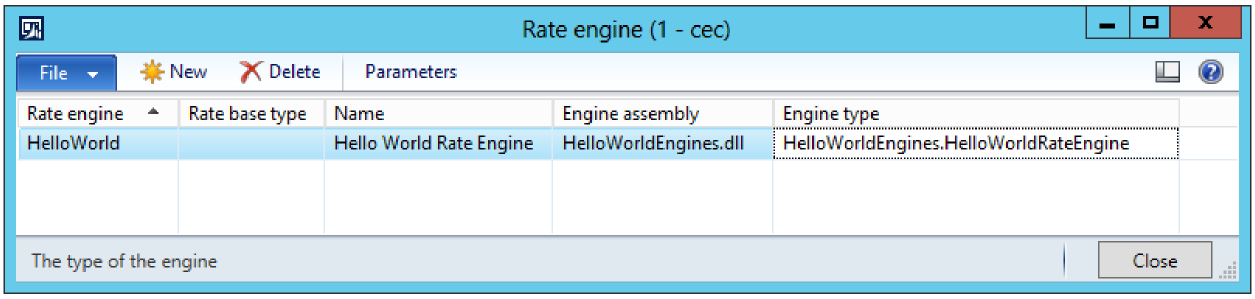
Your engine assembly is now available for use in Microsoft Dynamics AX. You can verify that HelloWorldEngines.dll is available in the following folder: [AOS installation location]\bin\VSAssemblies.

Tutorial: Enable a Hello-World rate engine

Follow these steps to enable the engine that you implemented and deployed in the previous tutorial.

1. Follow these steps to create a rating engine in Microsoft Dynamics AX:
    1. Click **Transportation Management** > **Setup** > **Engines** > **Rate engine** to open the **Rate engine** form.
    2. Create a new record that refers to the engine that you created in step 4 of the previous tutorial:
        - **Rate engine:** HelloWorld
        - **Name:** Hello World Rate Engine
        - **Engine assembly:** HelloWorldEngines.dll
        - **Engine type:** HelloWorldEngines.HelloWorldRateEngine

    The following illustration shows the new record.



Rate engine	Rate base type	Name	Engine assembly	Engine type
HelloWorld		Hello World Rate Engine	HelloWorldEngines.dll	HelloWorldEngines.HelloWorldRateEngine

The type of the engine

Note: Because your engine does not source any data from Microsoft Dynamics AX, you don't need to construct and assign a rate base type for it.

2. Follow these steps to specify a value for the **RateFactor** parameter:
    1. In the **Rate engine** form, click **Parameters**.
    2. Create a parameter record for **RateFactor**, and assign a value of **3**.

Your engine is now ready for use.

To test the engine, assign it to the rating profile of an active carrier, and run the rating, based on a source document that includes at least one line with a specific quantity. The total rate is computed as (Total lines quantity) * (RateFactor value). The Hello-World engine implementation is not currency-sensitive.

For more information about how to associate shipping carriers with rate engines, see Set up shipping carriers and carrier groups.

More about TMS engine implementation

- **Extension types** – The following table enumerates the most important interfaces and abstract classes for building engine extensions. All these base types are defined in the Microsoft.Dynamics.Ax.Tms.Bll namespace.

Name	Purpose	Notes
BaseRateEngine	Use this class to implement a new rate engine.	This class is the base class of concrete rate engines. It implements the **IRateEngine** interface. The concrete class derived from **BaseRateEngine** requires definition of methods for the calculation of transportation cost. By default, this class does not support the voiding of shipment operations.
IRateEngine	Use this interface to implement a new rate engine if you don't want to use the **BaseRateEngine** class. In particular, use this interface to implement a rate engine that communicates with external services through a web service.	This interface is used to calculate transportation cost and to run the voiding of shipments.
IApportionmentEngine	Use this interface to implement a new apportionment engine.	This interface is used to assign the transportation rate to each line element under a particular XML node.
ITransitTimeEngine	Use this interface to implement a new transit time engine.	This interface is used to calculate transportation time in days for a particular shipment. The transit time engine can take into consideration factors such as the source and destination addresses, mileage, and the type of service.

Name	Purpose	Notes
IZoneEngine	Use this interface to implement a new zone engine.	This interface provides the **RetrieveRatingZone** method, which is used by rate engines, and the **RetrieveRoutingZone** method, which is used to apply a proper routing guide. Typically, the response for a rating engine includes the calculation of the distance in zones from origin to destination. For route guide filtering, the valid response contains the identifier of the zone to which a particular address belongs.
IRateBaseAssigner	Use this interface to implement a new rate base assigner for rate engines that source Microsoft Dynamics AX data by using a rate base.	A rate base assigner is typically used by a rate engine to select a rate base that is applicable to a particular rating. This assignment is usually stored in the TMSRateBaseAssignment table. This table includes a number of generic fields, such as **Dimension1** to **Dimension6**. For proper UI interpretation of these fields, a rate base type must be assigned to the rate engine and to the rating profile. Concrete rate engines and concrete rate base assigners are expected to be decoupled from each other.

- **User messages** – To format a language-specific user message, use the **Microsoft.Dynamics.Ax.Tms.TMSGlobal** class that is defined in the TMS managed system. This class contains an overloaded method, called **getLabel**, that lets you retrieve a Microsoft Dynamics AX label in the current user language.
- **Language-sensitive data** – The input and output XML can contain elements that carry language-sensitive data, such as dates or decimal numbers. The TMS managed system enforces that the current language of the thread is set to invariant. Use **System.Globalization.CultureInfo.CurrentCulture** to serialize and de-serialize XML data.
- **Accessing data from Microsoft Dynamics AX** – A rate engine might require read or write access to the Microsoft Dynamics AX database. We highly recommend that you use proxy classes for .NET interop for interaction with tables in Microsoft Dynamics AX. For more information about proxy classes, see http://msdn.microsoft.com/en-us/library/gg879799.aspx.

  If you use proxy classes to interact with the Microsoft Dynamics AX database, consider using LINQ to Microsoft Dynamics AX to build and run queries. For more information about LINQ to Microsoft Dynamics AX, see http://msdn.microsoft.com/en-us/library/jj677293.aspx.

  Out of the box, the TMS managed system provides access to interaction with some of the TMS-specific tables by using proxy classes for .NET interop and LINQ to Microsoft Dynamics AX. You can use the **Microsoft.Dynamics.Ax.Tms.Data.AXDataRepository** class to retrieve **IQueryable** objects for these tables.
- **Using the common engine data infrastructure** – All the engines that are shipped out of the box with AX 2012 R3 require Microsoft Dynamics AX data in order to do the actual calculations. To reduce the cost of building engines that must source data from Microsoft Dynamics AX, the TMS system includes an implementation of a common engine data infrastructure. This feature enables the recording of engine-specific data, without the need to add new Microsoft Dynamics AX tables and build additional Microsoft Dynamics AX forms to maintain data. For more information about how to set up engine metadata, see "Transportation management engines" and "Set up transportation management engines" under Rating setup in online Help.

The TMS system defines a number of tables and Microsoft Dynamics AX forms that enable the recording of engine-specific data. This data is stored in physical table records that contain a number of generic table fields that can be reused for different purposes, depending on the specific engine implementation. For proper UI interpretation of data at runtime, metadata is recorded. For each group of data records that are used by a particular engine, a number of metadata records are required. These metadata records describe the caption, data type, lookup type, and a few other properties for each generic field that the engine is using.

Use the following table to find the Microsoft Dynamics AX table that contains data and metadata for a particular type of engine.

Engine type	Physical data table	Metadata table	Number of generic fields
Rate engine	TMSRateBaseDetail	TMSRateBaseField (where FieldType=Rate base)	6 **Note:** The last generic field on a rate base can be broken into a sequence of values of the same type, based on the break master. The merged **Detail** view in the **Rate base** form shows multiple values, which are stored in separate records in the TMSRateBaseDetail table.
Rate base assigner	TMSRateBaseDetail	TMSRateBaseField (where FieldType=Assignment)	6
Transit time engine	TMSTransitTimeDetail	TMSTransitTimeField	10
Zone engine	TMSZoneMasterDetail	TMSZoneMasterField	8
Mileage engine	TMSMileageDetail	TMSMileageField	8

Send feedback.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship, and supply chain processes in a way that helps you drive business success.

United States and Canada toll free: (888) 477-7989
Worldwide: (1) (701) 281-6500
www.microsoft.com/dynamics

Microsoft Dynamics