



METASTORM

Metastorm BPM® Release 7.6

Process Orchestrator for .NET Designer's
Guide

May 2008

Metastorm Inc.

email: inquiries@metastorm.com

<http://www.metastorm.com>

Copyrights and Trademarks

© 1996-2008 Metastorm Inc. All Rights Reserved.

Copyright Notice

- Metastorm®, Metastorm BPM®, e-Work®, Process Pod®, Enterprise Process Advantage®, ProVision®, The Best Process Wins®, Proforma®, Metastorm Knowledge Exchange™, Metastorm DNA™, Metastorm Discovery™, STAR™, Insight™, Envision™, and Metastorm Enterprise™ are either registered trademarks or trademarks of Metastorm in the United States and/or other countries.
- Microsoft®, Outlook®, SQL Server™, Windows®, Vista®, Active Directory®, Visual Basic®, JScript®, SharePoint® BizTalk® and IntelliSense are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe® is a registered trademark of Adobe Systems, Inc.
- Netscape® is a registered trademark of Netscape Communications Corporation.
- Other trademarks are the property of their respective owners.

Disclaimer

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Metastorm accepts no responsibility, and offers no warranty whether expressed or implied, for the accuracy of this publication.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the express written permission of Metastorm Inc.

The information in this document is subject to change without notice.

Metastorm Inc.

email: inquiries@metastorm.com

<http://www.metastorm.com>

Metastorm BPM Release 7.6

Process Orchestrator for .NET Designer's Guide

Table of Contents

1	Introduction.....	5
1.1	Acronyms	5
1.2	Terminology.....	6
1.3	Getting Further Information	6
2	What is the Metastorm Process Orchestrator for .NET?.....	8
3	Using the Process Orchestrator for .NET.....	9
3.1	Importing Methods via the Integration Wizard	9
3.2	Visual Studio .NET Integration	10
4	Setting Up the Process Orchestrator for .NET	11
4.1	Supported Environments and Installation Prerequisites	11
4.1.1	.NET Activator	11
4.1.2	Enterprise Component Library for .NET Server Components.....	11
4.1.3	Enterprise Component Library for .NET Sample Client.....	11
4.1.4	Process Events.....	11
4.1.5	Complex Types.....	12
4.1.6	Engine support for Process Orchestrators.....	12
4.1.7	ASP.NET Web Parts.....	12
4.2	Installation	12
4.2.1	Troubleshooting Installation.....	17
5	Calling Imported Methods via the Integration Wizard	18
5.1	Publishing the Library	18
5.2	Associating the Library with a Procedure	20
5.3	Calling Imported Methods via the Integration Wizard	20
5.4	Publishing the Procedure.....	21
5.5	Full Name Binding.....	21
6	Creating Process Events in Visual Studio .NET	22
6.1	Architecture	23
6.2	Publishing the Process Events Library.....	23
6.3	Creating and Publishing Procedures using the Designer.....	24
6.3.1	Creating a Procedure.....	24
6.3.2	Exposing Process Events in Designer.....	24

6.4 Creating a Metastorm BPM Process Code-Behind Project	26
6.5 Using a Metastorm BPM Process Code-Behind Project.....	33
6.5.1 <i>Process Events Class Diagram.....</i>	34
6.5.2 <i>Procedure.....</i>	35
6.5.3 <i>Stages</i>	36
6.5.4 <i>Forms</i>	38
6.5.5 <i>Admin Forms</i>	39
6.5.6 <i>Actions.....</i>	40
6.5.7 <i>Dependencies.....</i>	40
6.5.8 <i>CurrentVersion</i>	41
6.5.9 <i>Private member declarations & Event Handler Initializations.....</i>	41
6.5.10 <i>Public Accessor Methods / Properties</i>	43
6.5.11 <i>IntelliSense.....</i>	44
6.6 Metastorm BPM Process Code Behind Example	45
6.7 Configuring Process Metadata Web Service Connection Details.....	45
6.7.1 <i>Security.....</i>	46
6.7.2 <i>Process Engine Database Connection Settings</i>	46
6.7.3 <i>Authentication.....</i>	47
6.7.4 <i>Process Metadata Web Service Configuration Details Dialog.....</i>	47
6.8 Add-in	50
6.8.1 <i>Using the Add-in:</i>	51
6.8.2 <i>Metastorm Web Service Configuration Details</i>	52
6.8.3 <i>Deploy Process</i>	52
6.8.4 <i>Debug With Process Engine</i>	54
6.8.5 <i>Resynchronizing a Project</i>	55
6.8.6 <i>Solution Browser</i>	57
Appendix A – Simple Type Mapping	59
Appendix B – Examples of Scripts Generated by .NET Activator.....	60
Activator [Assembly Name].....	60
AvailableFunctions [Assembly Name].....	60

Metastorm BPM Release 7.6

Process Orchestrator for .NET Designer's Guide

1 INTRODUCTION

The Metastorm Process Orchestrator for .NET is a bridging technology that connects Metastorm processes and .NET assemblies.

This document is intended to:

- Explain what the Metastorm Process Orchestrator for .NET is.
- Provide an overview of how to use the Metastorm Process Orchestrator for .NET.
- Summarize set-up information for the Metastorm Process Orchestrator for .NET.
- Describe how .NET assembly methods, imported into an Integration Wizard collection library by the .NET Activator, can be accessed.
- Describe how custom extensions can be developed and executed in response to Metastorm BPM process events.
- Explain Metastorm BPM Visual Studio 2005 .NET Integration - Process Events

1.1 Acronyms

The following table lists the acronyms used in this guide:

<i>Term</i>	<i>Meaning</i>
DLL	Dynamic Link Library
ECL	Metastorm Enterprise Component Library
GAC	Global Assembly Cache

IDE	Integrated Development Environment
IIS	Internet Information Services
XEL	Metastorm XML Library file extension
XML	eXtensible Markup Language

Table 1: Acronyms

1.2 Terminology

The following table lists terms that have specific meanings for this guide:

Term	Meaning
.NET Activator	A tool that allows a System Integrator to create a Metastorm XML Library (XEL) file which contains an Integration Wizard Collection and a script to expose the functions contained within the Integration Wizard Collection to the Process Designer.
Metastorm Enterprise Component Library for .NET	Class library that exposes a .NET component based interface to Metastorm functionality.
Metastorm Code Behind Project	A Visual Studio .NET 2005 project that integrates code behind to a Metastorm Designer procedure. The project is created using Process Events. Refer to section 6 Creating Process Events in Visual Studio .NET
Metastorm Client	Application used to access Metastorm processes.
Metastorm Designer	Application used by a Process Designer to design Metastorm processes.
Metastorm Engine	A single Metastorm server process.
Integration Wizard collection library	A library that contains an Integration Wizard collection. The Integration Wizard collection contains additional Integration Wizard function definitions and can be viewed via Library Properties. For further information, refer to the Designer User Manual.
Integration Wizard Collection	A Set of Integration Wizard functions that are defined in a library for use in a Metastorm procedure.
.NET Developer	The person who imports a Metastorm BPM process and uses the extended Visual Studio functionality to create .NET assemblies containing code to be invoked in response to Metastorm BPM Process Events.
Process Designer	The person who uses the Metastorm Designer to create a process. The Process Designer can use functions provided by the System Integrator and delegate process events.
System Integrator	The person who uses the Activator to create a set of Integration Wizard Collection functions that will be available to a Process Designer.

Table 2: Terminology

1.3 Getting Further Information

This manual does not cover, in detail, the Metastorm .NET Activator, the Metastorm Enterprise Component Library (ECL) for .NET, the ASP.NET Web Parts, the Integration Wizard component

of the Metastorm Designer or server-side scripting with Jscript .NET. The following table lists where to find detailed information on these topics:

<i>For Information on</i>	<i>See</i>
Metastorm .NET Activator	Metastorm .NET Activator Help (accessible from the .NET Activator)
Metastorm Enterprise Component Library for .NET	Metastorm Enterprise Component Library for .NET Usage Guide
Integration Wizard	Designer User Manual
Process Events	This document and the Metastorm Process Events Help (accessible from the Process Events wizard).
Server-side scripting with Jscript .NET	Scripting Developer Guide

Table 3: Getting further information

2 WHAT IS THE METASTORM PROCESS ORCHESTRATOR FOR .NET?

The Metastorm Process Orchestrator for .NET provides interoperability between Metastorm processes and services implemented as .NET components.

The Process Orchestrator for .NET can be used by a Visual Studio .NET developer to:

- Develop application components as .NET Assemblies. The Metastorm .NET Activator enables methods from .NET Assemblies to be stored in an Integration Wizard collection library which, when the library is attached to a procedure, updates the Integration Wizard in the Metastorm Designer with the methods. These methods can then be invoked by Metastorm procedures.
- Delegate certain process events from within the Designer. Then create assemblies using the Metastorm BPM Process Code-Behind Project template in Visual Studio to handle the delegated process events.

In addition, .NET developers can use the Metastorm Enterprise Component Library for .NET to access Metastorm functionality via a component based interface.

For further information on the Metastorm Enterprise Component Library for .NET, refer to the Metastorm Enterprise Component Library for .NET Usage Guide.

Web developers can also use the Web Client ASP.NET web parts to build websites using Visual Studio that contain Metastorm functionality.

For further information, refer to the Web Client ASP.NET Web Parts Developer Guide.

3 USING THE PROCESS ORCHESTRATOR FOR .NET

3.1 Importing Methods via the Integration Wizard


The Process Orchestrator for .NET is used in the following way:


1. A Visual Studio .NET Developer develops application components as .NET Assemblies.
2. A System Integrator uses the Metastorm .NET Activator to make selected .NET classes available as functions within the Metastorm Designer's Integration Wizard, as follows:
 - i. The System Integrator runs the Metastorm .NET Activator.
 - ii. The System Integrator uses the .NET Activator to specify a .NET Assembly Cache (GAC), Zap Cache or downloaded Cache, browsing to another PC if necessary.
 - iii. The .NET Activator uses Reflection to interrogate the list of .NET components hosted in the Cache and provides the System Integrator with a list of the classes and methods exposed by each assembly.
 - iv. The System Integrator selects which methods they want to be available as Integration Wizard functions in the Metastorm Designer.
 - v. The .NET Activator then creates a Metastorm library containing a script with these functions.
 - vi. The .NET Activator saves the library in .XEL format.

For information on using the Metastorm .NET Activator to import methods from .NET assemblies into the Metastorm Designer, refer to the .NET Activator Help. This can be accessed from the .NET Activator.

3. A Process Designer publishes the library.
4. A Process Designer can use functions exposed by .NET Assembly components from within Metastorm procedures by including the created library within their procedures. If the .NET Assemblies are:

- Stateless, the Integration Wizard can be used.
- Stateful, Process Designers can use these assemblies directly from the Metastorm Designer Script Editor. A skeleton script including class declaration and instantiation is available, but scripting of the method calls may need to be modified manually.

 *Stateless assemblies are those in which a single method can be called without requiring any previous methods to be called.*

 *Stateful assemblies are those which require a number of method calls. An example would be if you were to receive a result back from a rules engine, pass the value into the assembly and then perform other actions using different functions in the assembly.*

3.2 Visual Studio .NET Integration

A Visual Studio .NET developer writes process events which are triggered by events exposed in the Designer.

 *Refer to section 6 Creating Process Events in Visual Studio .NET.*

4 SETTING UP THE PROCESS ORCHESTRATOR FOR .NET

4.1 Supported Environments and Installation Prerequisites

4.1.1 .NET Activator

- Microsoft .NET framework version 2.0.
- Microsoft .NET 2.0 Software Development Kit

4.1.2 Enterprise Component Library for .NET Server Components

- Metastorm Engine 7.5 or later.
- DCOM Access permissions granted to the installing user.
- Microsoft Internet Information Services, when HTTP hosting is configured.
- Microsoft .NET framework version 2.0.

4.1.3 Enterprise Component Library for .NET Sample Client

- Microsoft Internet Information Services.
- ASP.NET 2.0.
- Microsoft .NET framework version 2.0.

4.1.4 Process Events

- Microsoft .NET framework version 2.0.
- Metastorm Process Metadata Service

The Process Metadata Service is a web service which queries the Metastorm repository and provides calling applications with specific information about published processes. For example, it provides a list of the available process maps. For a particular map, it can provide a list of all the available actions within the map, together with form field definitions etc. The Metastorm Process

Metadata Service can be installed from the Metastorm BPM CD. It can be found under the “Metastorm Process Engine” component.

 *The Process Metadata Service is not installed by default.*

4.1.5 Complex Types


- Metastorm BPM Designer
- Engine support for Process Orchestrators (if Metastorm Process Engine is installed)


4.1.6 Engine support for Process Orchestrators

- Metastorm Engine 7.5 or later.

4.1.7 ASP.NET Web Parts

- Visual Studio 2005
- SQL Server 2005 Express Edition

 *In order to install the .NET framework version 2.0, Windows Installer version 3.0 must be installed. This is installed as part of Windows Server 2003 or can be installed separately.*

 *For further details of supported environments and installation prerequisites for Metastorm BPM, refer to the Supported Environments guide and Installation Prerequisites guide provided with the main product.*

4.2 Installation

To install the Process Orchestrator for .NET:

1. Insert the Process Orchestrator for .NET CD into your CD drive.

If the drive is:

- Configured to autorun, the installation procedure starts automatically
- Not configured to autorun:
 - i. Access Windows Explorer and browse to the files on the CD.
 - ii. Double-click the file **Autorun.exe** to start the installation.

The autorun screen is displayed.

2. To proceed with the installation, click on the **Process Orchestrator for .NET** link.
 - You see the initial screen of the Process Orchestrator for .NET installation:

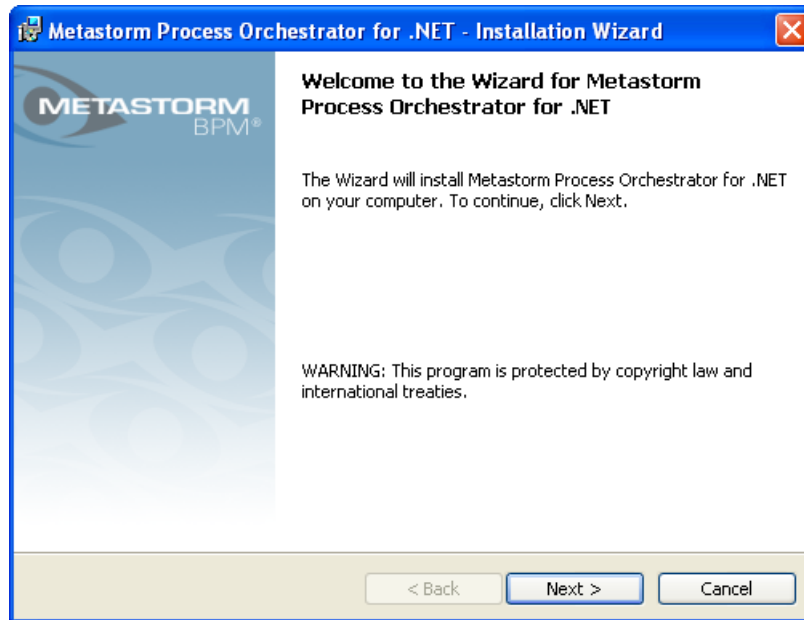


Figure 1: Metastorm Process Orchestrator Installation Welcome Screen

3. Click on the **Next** button.
 - The License Agreement screen of the Process Orchestrator for .NET installation is displayed:

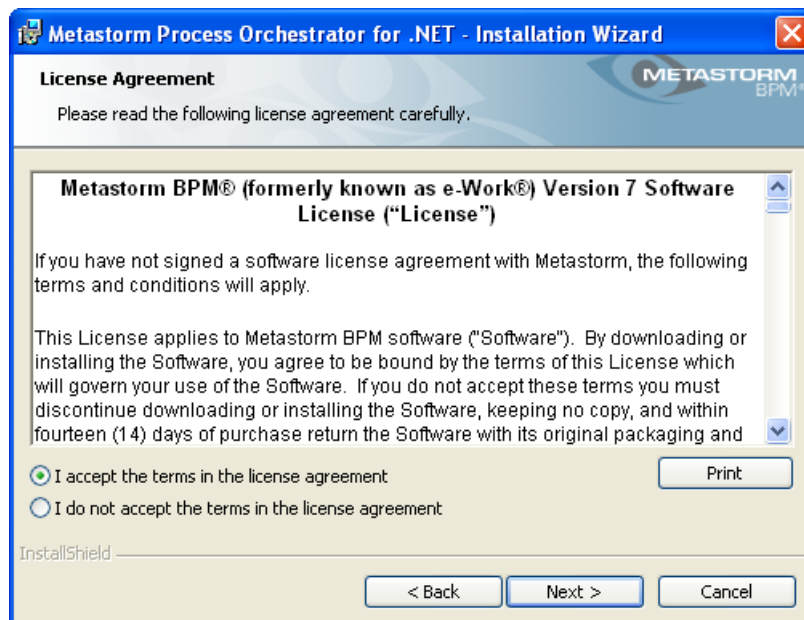


Figure 2: Metastorm Process Orchestrator License Agreement Screen

4. Click **Print** to print the installation and startup information.
5. Select the **I accept the terms in the license agreement** radio button.

6. Click on the **Next** button.
 - The Custom Setup screen of the Process Orchestrator for .NET installation is displayed.

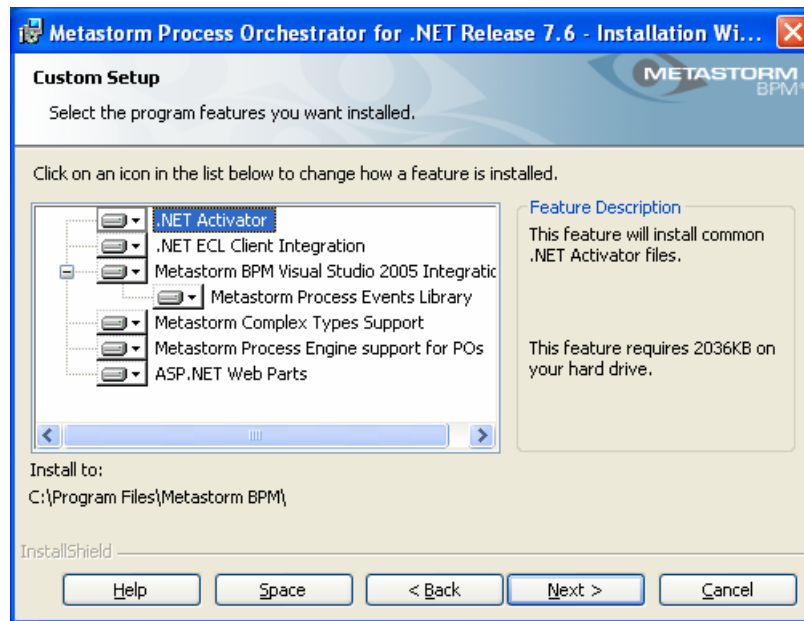





Figure 3: Metastorm Process Orchestrator Custom Setup Screen

-  The options that appear on this screen depend on the versions of the .NET framework that are installed.
-  The feature “Metastorm Process Engine support for POs” can only be installed if the Metastorm BPM Process Engine is installed. Selecting “.NET Activator” requires the installation of “Metastorm Process Engine support for POs”. If “Complex Types” is selected, this feature is only required if the Engine is installed on the same machine.
-  For further information on Process Orchestrator for .NET component information, refer to the .NET Activator Help (accessible from .NET Activator) and Enterprise Component Library for .NET Usage Guide and Web Client ASP.NET Web Parts Developer Guide. Details on Complex Types support can be found in the Designer User Manual.

7. Select the components of the Process Orchestrator for .NET that you want to install.
8. Click on the **Next** button.
 - The Destination Folder screen of the Process Orchestrator for .NET installation is displayed:

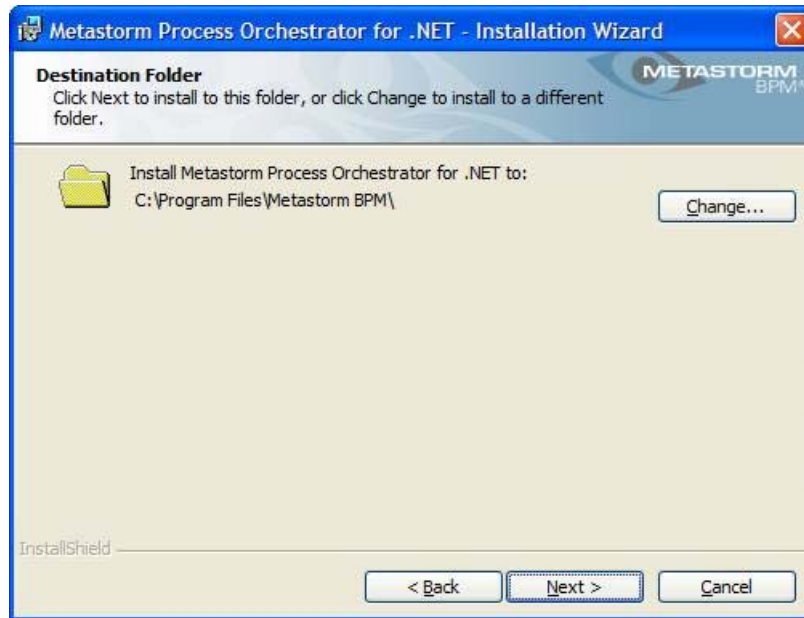


Figure 4: Metastorm Process Orchestrator Destination Folder Screen

9. If you want to install the Process Orchestrator for .NET to a location other than the default location, click on the **Change** button and browse to a new location.

 The Destination Folder screen is only displayed if Metastorm BPM has not been installed.

10. Click on the **Next** button.
 - The Connection Setup screen of the Process Orchestrator for .NET installation is displayed:

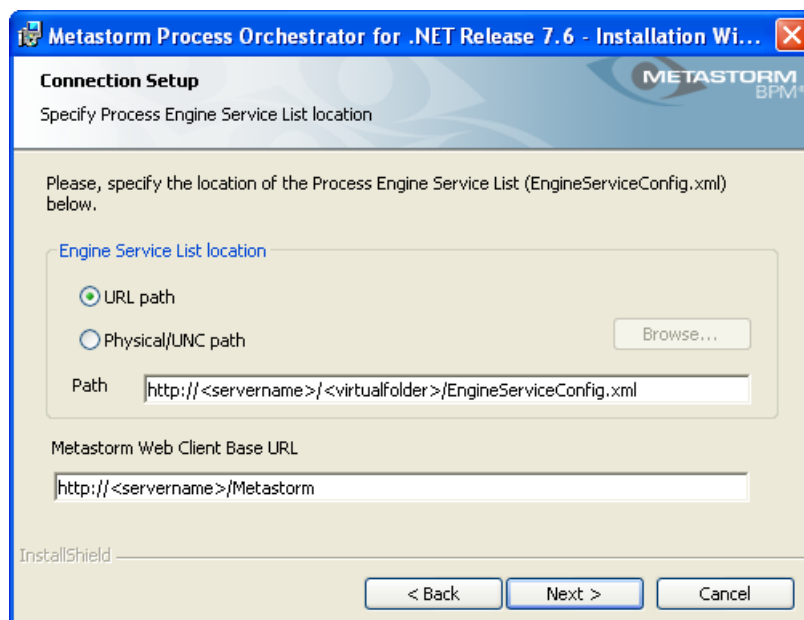




Figure 5: Metastorm Process Orchestrator Connection Setup Screen

11. Set the radio buttons to select the link to the location of the Process Engine Service List to either a Physical/UNC path or a URL.

- If the Engine is to be installed on the same machine the path defaults to the local Engine location.
- If the Engine is installed on another machine then enter the path or browse to the location if the Engine is already installed. Expose the folder containing EngineServiceConfig.xml on that machine as a network share with the appropriate permissions to access via UNC path.
- If the connection is over HTTP enter the URL in the format:
http://<enginecomputer>/escripts/EngineServiceconfig.xml.


 The engine service file EngineServiceConfig.xml can be configured to contain multiple services.

 For further information on customizing EngineServiceConfig.xml refer to the Metastorm Administration Guide.

12. Set the Web Client Base URL.

- The Metastorm Web Client Base URL is defined as:

http://<Web Server Name>/<Metastorm Virtual Folder Name>

 This option is only available is the user selected ASP.NET Web Parts in the Installation Screen.

13. Click on the **Next** button.

- The Database Setup screen is displayed:

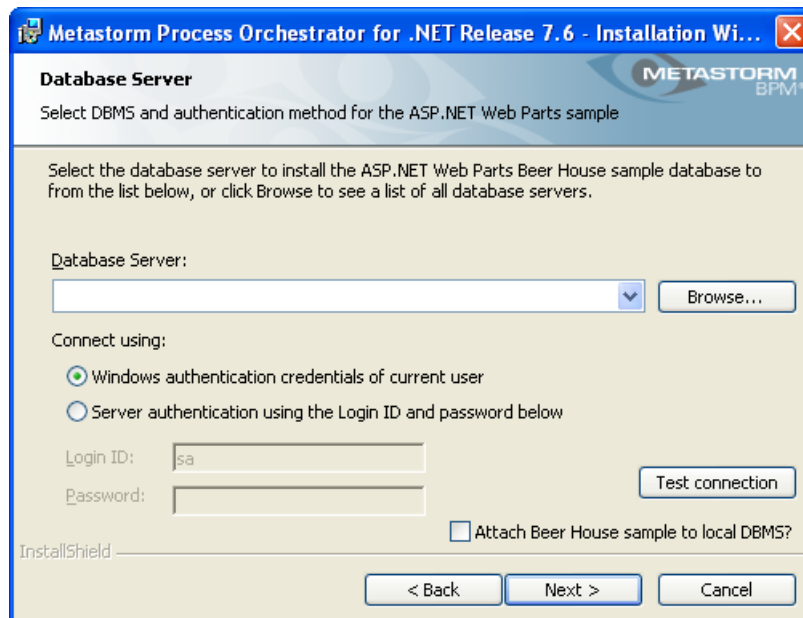


Figure 6: Metastorm Process Orchestrator Database Server Screen

14. Click on the **Next** button.

- The Ready to Install screen of the Process Orchestrator for .NET installation is displayed:*

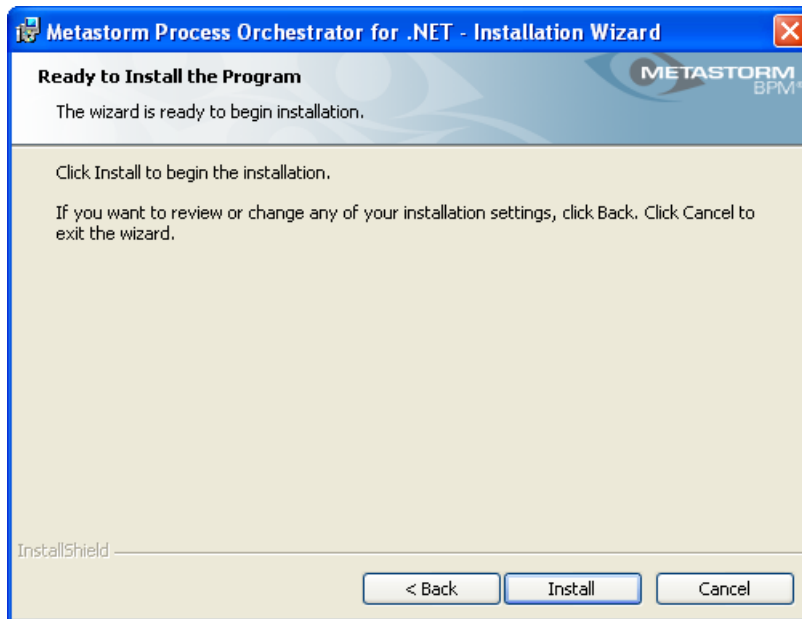


Figure 7: Metastorm Process Orchestrator Ready to Install Screen

15. Click the **Install** button.
 - The Process Orchestrator for .NET is installed.
16. Click the **Finish** button.
17. Start the Engine service.
18. Start the ECL service.

4.2.1 Troubleshooting Installation

Unable to access the BPMEngine.Net URL

When a client using the Engine's .NET interface attempts to connect to an engine on another machine, the following error may be reported:

Access denied 401 error.

To resolve this problem, open the Internet Information Services administration tool, and change the BPMEngine.Net virtual folder to turn on Integrated Windows Authentication (IWA). Connect to the service using the machine with the web service installed via the ECL test client.

5 CALLING IMPORTED METHODS VIA THE INTEGRATION WIZARD

You can use the .NET Activator to import functions exposed by .NET Assemblies into a Metastorm Integration Wizard collection library.

For details of how to import functions exposed by .NET, refer to the .NET Activator Help (accessible from the .NET Activator).

To use functions, exposed by .NET Assemblies, in a Metastorm procedure, you must:

1. Deploy the assemblies in both the designer\dotnetbin and the engine\dotnetbin directories using Visual Studio.
2. Publish the library.
3. Associate the library with a procedure.
4. Call imported methods via the Metastorm Designer Integration Wizard.
5. Publish the procedure.

The following subsections describe these steps in more detail.

5.1 Publishing the Library

The .NET Activator produces a Metastorm library containing an Integration Wizard collection.

To publish the library:

1. Access the Metastorm Designer.
2. Open the library by selecting the File menu then the Open menu item.
3. View the Integration Wizard collection, if required, as follows:

- i. Select the File menu, then the Library Properties menu item.

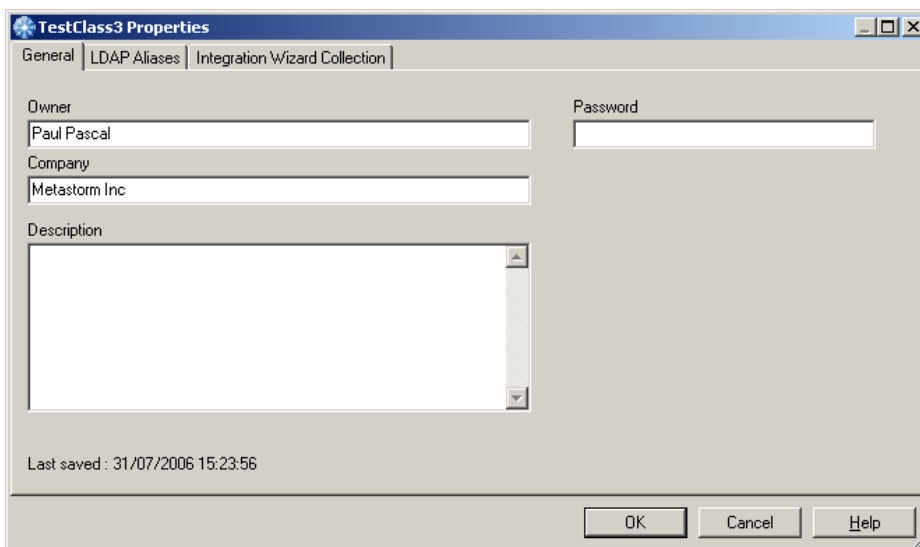


Figure 8: Library Properties

- ii. Click on the Integration Wizard Collection tab.

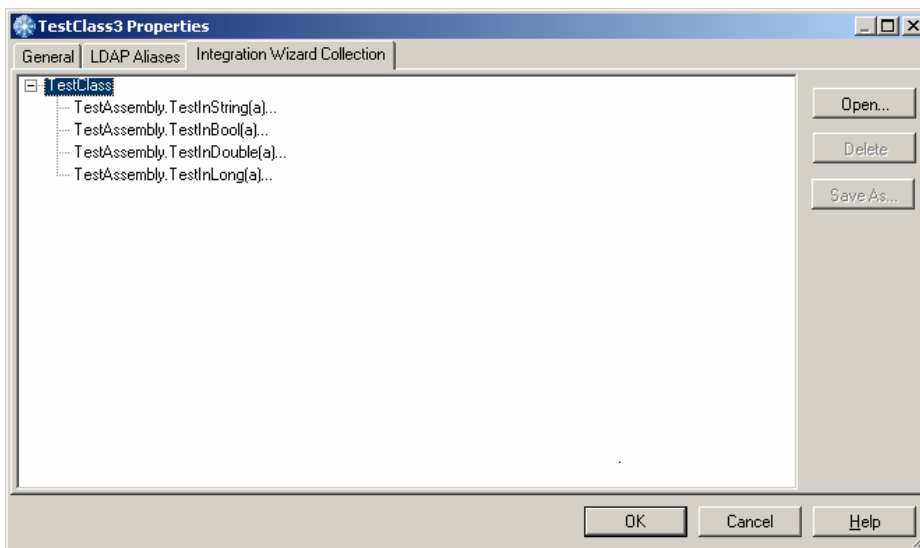


Figure 9: Integration Wizard Collection

- iii. Click on the OK button.
4. View or edit the scripts containing the imported functions, if required, as follows:
 - i. In the main Designer window, select the View menu, then the Scripts menu option.
 - ii. Click on the Server tab.
 - iii. Edit a script, if required, by selecting the script then clicking on the Edit button.
 - iv. Click on the Close button.
5. Publish the library by selecting the File menu, then the Publish menu option.

5.2 Associating the Library with a Procedure

To access the new Integration Wizard items that correspond to the imported .NET methods, you must associate the new library with a procedure via Procedure Properties.

For further information on associating a library with a procedure, refer to the Designer User Manual.

To associate the library with a procedure:

1. Create a new procedure by selecting the File menu then the New menu option, or open an existing procedure by selecting the File menu then the Open menu option.
2. Add the library to the procedure, by selecting the File menu then the Procedure Properties menu option.
3. Click on the Used Libraries tab.

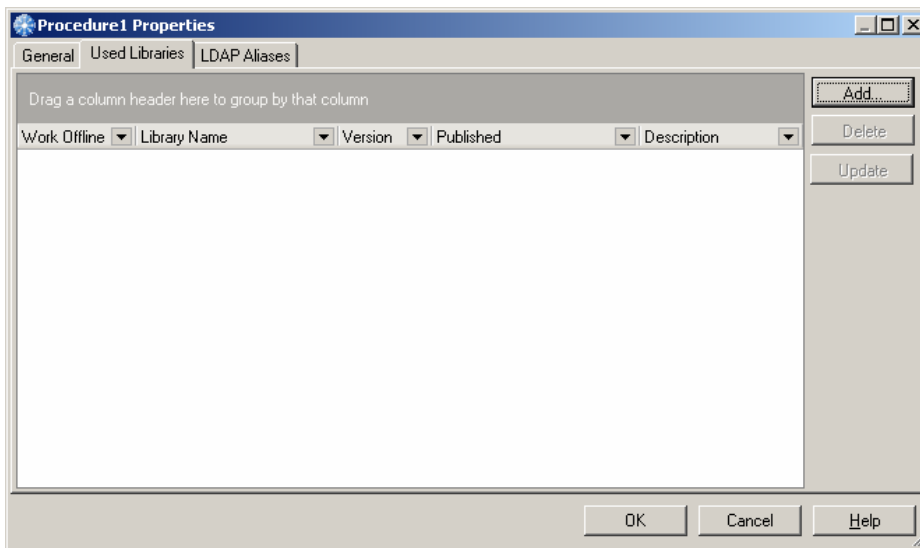


Figure 10: Used Libraries Tab

4. Click on the Add button and browse for the required library.
5. Click on the OK button.

5.3 Calling Imported Methods via the Integration Wizard

Once the library has been published and associated with a procedure, the imported methods are available for selection in the Integration Wizard when that procedure is open in the Designer.

To incorporate .NET Assembly functions into a procedure:

1. For any event where you want to incorporate any of the required functions, use the Integration Wizard to access the functions, under the specified category.

2. Enter any values required, by the function, as parameters.
3. Click on the Next button.

For further information on using the Integration Wizard, refer to the Designer User Manual.

5.4 Publishing the Procedure

To publish the procedure:

1. If the .NET assembly with the imported functions is in a DLL, ensure the DLL is in the following location:

```
<Metastorm Installation Directory>\Engine\Dotnetbin
```

2. Select the View menu then the Options menu option.
3. On the Publisher tab, ensure the Enable versioning option is checked.
4. Click on the OK button.
5. Publish the procedure by selecting the File menu then the Publish menu option.
 - The procedure can now be accessed from a Metastorm Client.

5.5 Full Name Binding

If a file is referenced from the GAC, it requires a full name binding, including a version and a public key token, as well as the assembly name, where as before only the assembly name was required.

For example, in an earlier Version of Metastorm BPM, the following call to an assembly method in the GAC would be valid:

```
%resGet_FolderID:=%ScriptEval(JScript.NET, ,%Procedure.Name,%M
apName, "eWork.Activator.TestClass.Invoker.Activate", "TestClas
s", "TestClass.eWorkObject", "get_FolderID", "Public")
```

Now, the function must be to the following format:

```
%resGet_FolderID:=%ScriptEval(JScript.NET, ,%Procedure.Name,%M
apName, "eWork.Activator.TestClass.Invoker.Activate", "TestClas
s, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=0fa3cc64eebf4c8b", "TestClass.eWorkObject", "get
_FolderID", "Public")
```

6 CREATING PROCESS EVENTS IN VISUAL STUDIO .NET

Process Events Integration is a method of separating Process Design from event implementation.

Process events are external events that are created via a wizard that connects Metastorm BPM processes to Visual Studio .NET. This separates the design and coding processes.

Processes are setup in Designer and the process event handlers are created using the Designer via the Integration Wizard and Visual Studio 2005 .NET.

Process events setup using Visual Studio 2005 enable a .NET developer to develop customized events that respond to Metastorm BPM process events. The customized events can be written using C# or Metastorm BPM language.

The people involved in process events are:

- Process Designer - the participant who uses the Metastorm BPM Designer to create process maps and delegates .NET event handler assembly functionality from a Metastorm BPM process.
- .NET Developer - the participant who imports a Metastorm BPM process and uses the extended Visual Studio functionality to create .NET assemblies containing code to be invoked in response to Metastorm BPM process events.

This section goes through the steps required to create external process events in .NET:

1. The Process Designer publishes the Process Events Library using the Designer.
2. The Process Designer creates and publishes procedures setting event handler options which call .NET events.
3. The .NET Developer creates a Metastorm BPM Process Code-Behind Project which contains the skeleton code for creating process events.

4. The .NET Developer creates C# source code inside the process events skeleton using Visual Studio.
5. The .NET Developer builds and deploys the assemblies in the engine\dotnetbin or engine\dotnetlib folder.
6. The Process Participant is now able to fill in the relevant forms using a browser.

In addition to the above steps, this section also explains what to do when stages, maps or events that may have been renamed, deleted or added.

6.1 Architecture

The exposure of the Process Events has been designed as shown in the following diagram:

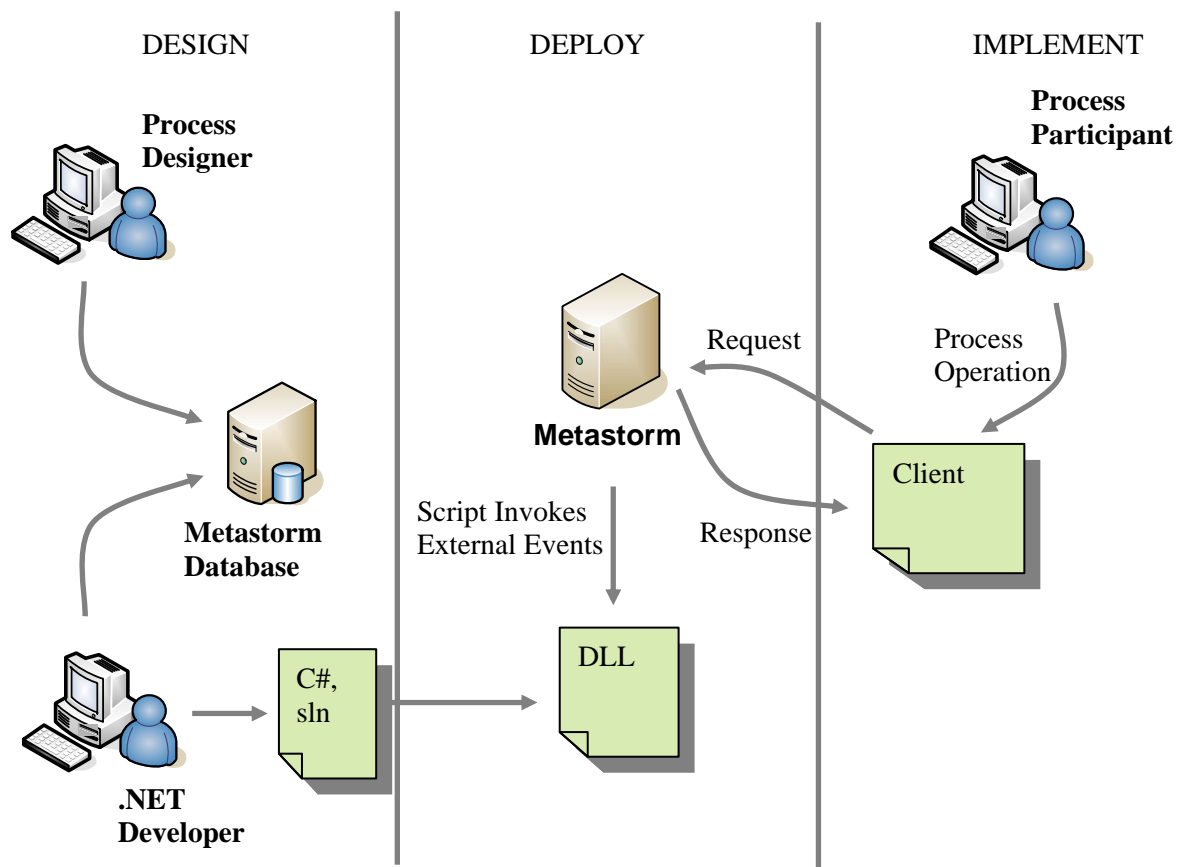



Figure 11: Process Events Architecture

6.2 Publishing the Process Events Library

The Process Events Library contains a script which is required for calling an external event handler defined in Visual Studio 2005.

To use the Process Events Library the Process Designer:

1. Opens the Process Events Library
2. Publishes the Process Events Library
3. Associates the Process Events Library with a new or existing process.
4. The Event Handler category is now available in the Integration Wizard.

 For further information, refer to 6.3.2 Exposing Process Events in Designer.

6.3 Creating and Publishing Procedures using the Designer


6.3.1 Creating a Procedure


A Metastorm procedure is created using the Designer to automate business processes using maps, folders, stages and actions.

The Metastorm process is represented by maps, stages, forms and actions in the Designer. The Process Designer can insert formula and script code in the Designer to execute events. The events can be created in Designer or the events can be exposed using the Integration Wizard. The exposed events call external event handlers created in .NET.

Events are found in the **Do This** properties tab and typically are:

- When Action Started
- When Action Completed
- When Stage Started
- When Stage Completed
- When Form Loaded
- When Form Completed

 Form control events, for example buttons and text fields, are not supported.

 For further information on creating procedures refer to the Designer User Manual.

6.3.2 Exposing Process Events in Designer

Each process in a procedure can have its event exposed to Visual Studio .NET solutions. Process events can be exposed:

- Locally
- Globally

In order to use exposed process events and customized extensions, `Process Events Library.xel` should be published and associated with the procedure. This library contains the External Event Handler Integration Wizard function which contains a generic event for for local and global exposed processes.

Locally

The events can be exposed process by process by placing a call to a formula which inserted manually or using the Integration Wizard.

The Integration Wizards Collections Library contains a `%ExecuteExtensionEval` call to invoke a JScript.NET to handle process events. The External Event Handlers item identifies the Procedure Name and invokes the correct action, form or stage event.

Each action, form and stage can have process events exposed as required. These are exposed as follows:

1. Associate `Process Events Library.xel` with the procedure.
2. Select the action, form or stage.
3. Select the **Do This** tab in the Properties window.
4. Select the event, for example, **When Action Started**.
5. Click the Integration Wizard button
6. Select the **Event Handlers** category and the **External Event Handler** item
7. Click **Finish**
8. The following syntax is automatically created:

```
%ExecuteExtensionEval(JScript.NET, %Procedure.Name, %MapName,  
"Metastorm.ProcessEvents.HandleExternalEvents.DelegateEvents"  
, )
```

9. The process can then be published.

 *This script will run the relevant .NET code. This function call should not be modified.*

Globally

All Process Events can be exposed using the **Map** property tab of the map. This tab contains two options which globally expose process events:

- Delegate all external events for map
- Perform delegated events before local event

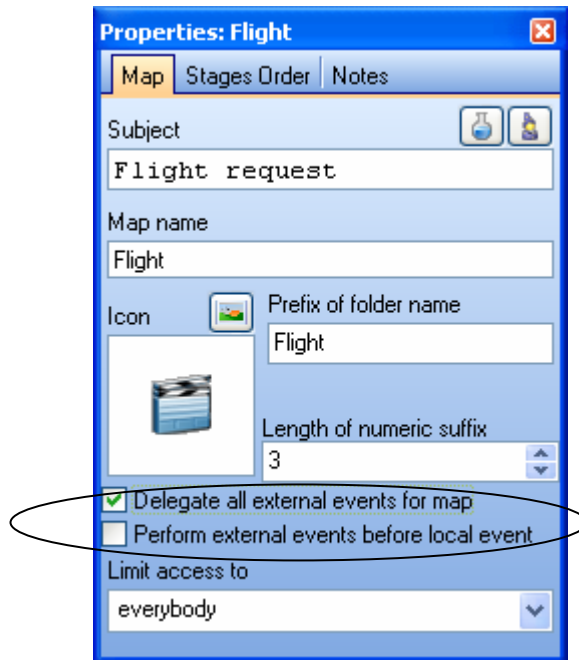





Figure 12: Map Properties highlighting Process Events options

Delegate all external events for map

This is a check box property which defines whether all the events of a process will be delegated to an external .NET assembly. Local events can still be defined if required. When checked, this option implies that each event has the External Event Handler script applied to it.

Perform delegated events before local event

This defines when the process engine will invoke the delegated events when all processed events are delegated. The default value (unchecked) performs delegated events after the local event. The property is only active if the **Delegate all external events for map** is checked.

-  *The `Process Events Library.xel` needs to be associated with the procedure for these options to have an effect.*
-  *If the user has also delegated external events in a local reference, the external event will be fired twice.*
-  *Admin Forms are not included by the **Delegate all external events for map** option. To include Admin Form, for each Admin Form apply local events to the forms (**When Form Loaded** and **When Form Saved**) for them to be included.*

6.4 Creating a Metastorm BPM Process Code-Behind Project

Metastorm BPM provides a Process Events Wizard which is a custom Visual Studio project wizard. It allows the developer to create a new solution based on data retrieved from a Metastorm database via a user selected Web Service. The solution will contain one or more projects with

each project corresponding to each procedure retrieved from the database and selected by the user. For each project a class library is automatically generated by the wizard.

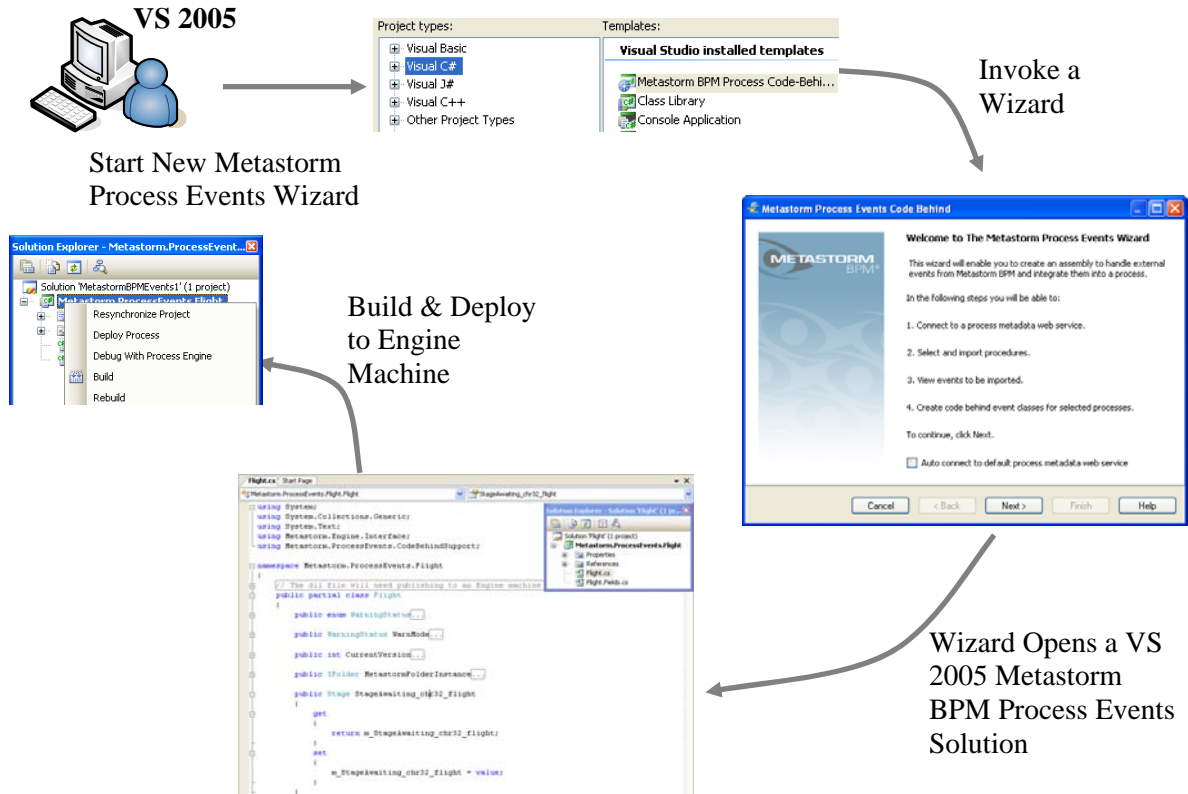


Figure 13: Process Events Workflow

The Process Events Wizard takes the user through a set of serial steps via a standard ‘wizard’ interface, a sequence of dialog windows, where information will be entered and/or approved by the user before proceeding to the next step. Help is provided from the Wizard.

Please be aware that the Process Events Help may be displayed behind Visual Studio when the Help button is clicked in the Wizard.

The steps for creating a Process Events project are:

1. Open Visual Studio .NET.
2. Select **File | New | Project** from the menu.
3. Select Project Type **Visual C#**.
4. Select **Metastorm BPM Process Code-Behind Project** template.

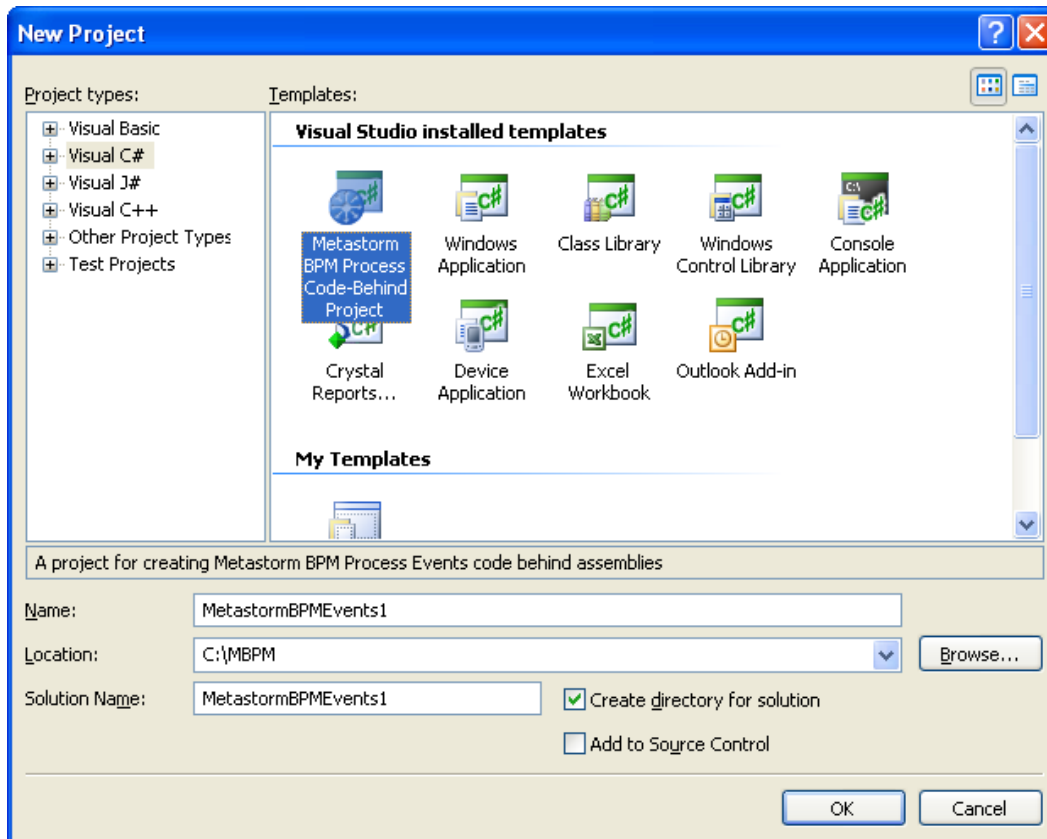




Figure 14: Metastorm BPM Process Code-Behind Wizard – New Project

5. Enter the location and solution name.

 Any project name entered will be overwritten by the wizard.

 The location name of the project plus the file name should not exceed 248 characters. It is recommended that the .NET Developer does not use the default Visual Studio path as the dynamically created map classes may exceed this limit.

6. Click **OK**.
7. An introductory window is displayed to the user, explaining the wizard functionality.

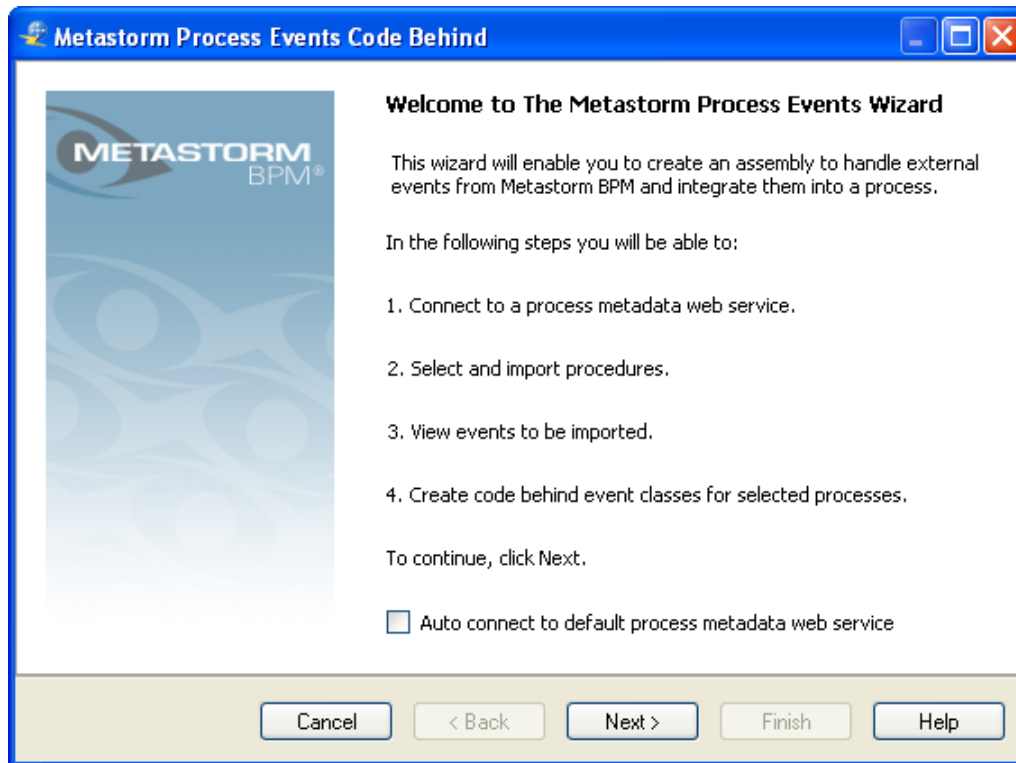



Figure 15: Metastorm Process Events Code Behind Wizard – Welcome

8. Check **Auto connect to default process metadata web service** if you have already have a default process metadata web service. The default configuration is set in the Metadata Web Service Connection Details. The default configuration is automatically moved to the top of the service list.

 *Checking this option, skips steps 10 and 11.*

9. Click the **Next** button.
10. Choose the desired Metadata service to connect to by using the drop down, filling in the URL and authentication type if required.

Metastorm Process Events Code Behind

Process Metadata Web Service Connection Details

Provide valid credentials to the metadata service, in order to retrieve event enabled processes

Alias:

URL:

Type:

User name:

Password:

Domain:

Default Configuration

Proxy Details

Use Proxy

Type:


Host:


User name:

Password:

Bypass if local address

Figure 16: Metastorm Process Events Code Behind Wizard - Web Service Connection Details

 The Metastorm Process Metadata Service should be installed on the same machine as the engine and it is only used in a development environment.

 For configuration details refer to section 6.7 Configuring Process Metadata Web Service Connection Details.

11. Click **Next**.

12. Select and import procedures from the displayed list. All procedures in the list contain exposed events.

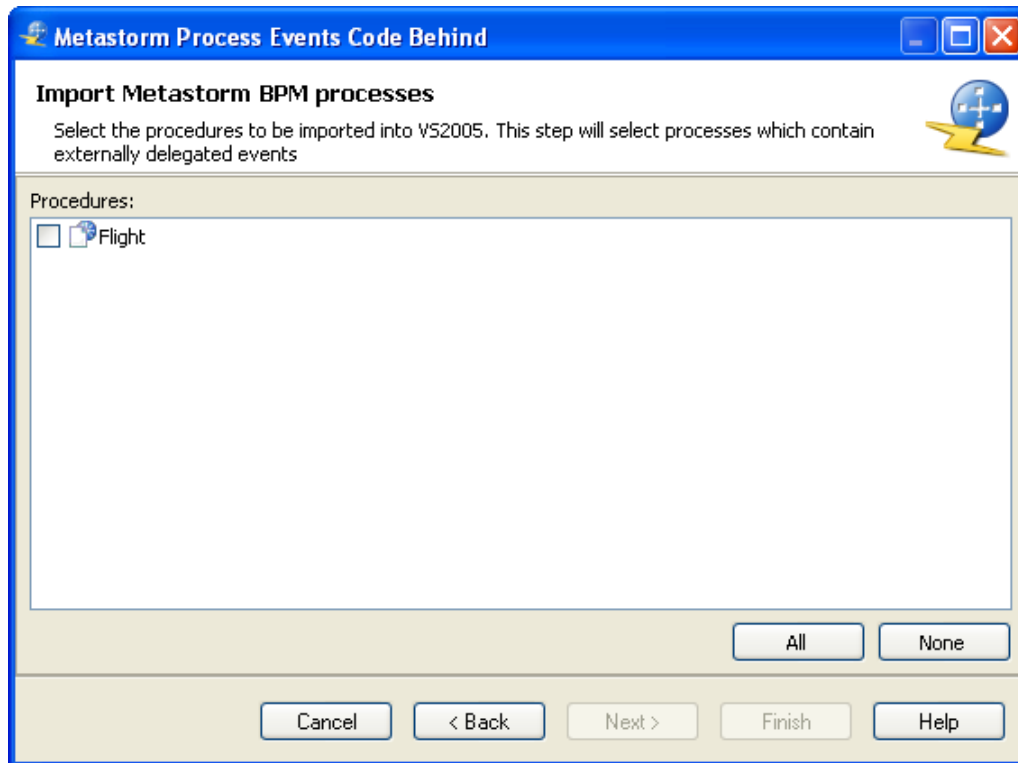


Figure 17: Metastorm BPM Process Events Code Behind Wizard – Import Procedures

13. Click **Next**.
14. A hierarchical 'tree view' of the selected procedures is displayed.

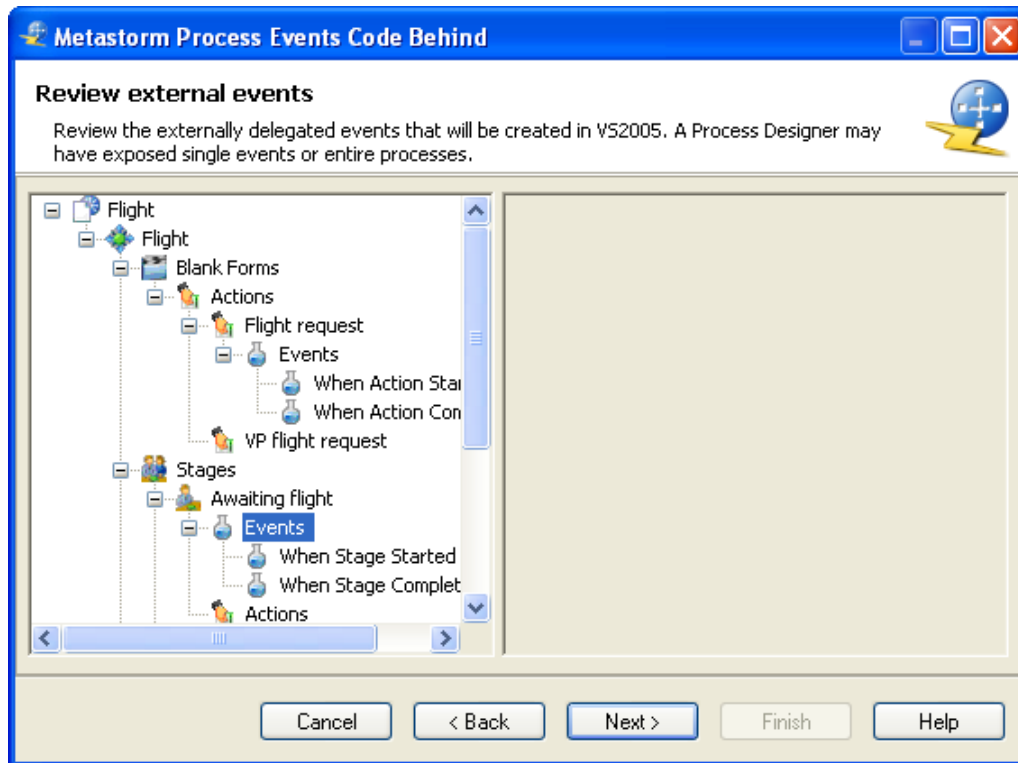


Figure 18: Metastorm BPM Process Events Code Behind Wizard – Review External Events

15. Click **Next**.
16. The wizard generates a new Visual Studio solution based on the procedures selected by the user.

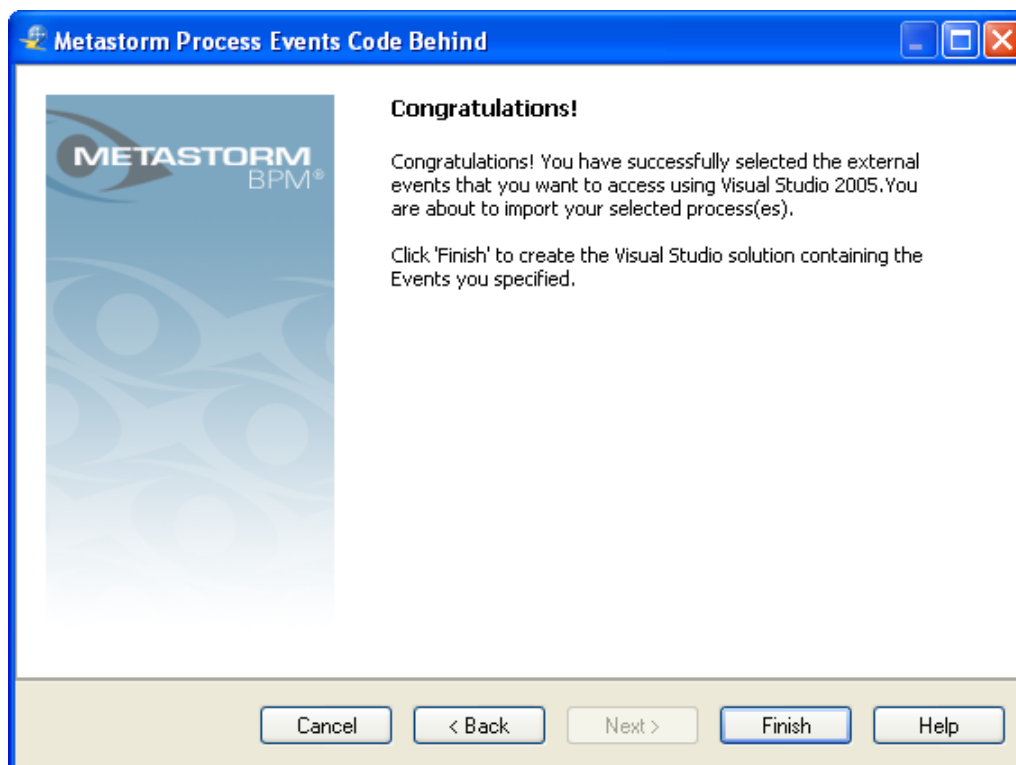


Figure 19: Metastorm Process Events Code Behind Wizard - End

17. Click **Finish**.

The default project created has the name `Metastorm.ProcessEvents.<ProcedureName>`. The solution name is defined by the user in the first screen of the wizard.

Each project consists of C# files relating to each Map and each Admin Form group containing External Events.

*✎ Non alphanumeric characters (except `_`) that are accepted by Designer in Stage, Form, Actions or Event names, are replaced with the corresponding character code. For example a Stage named **Process & Review** in Designer will be amended to **Process_ chr32_ chr38_ chr32_ Review** in Visual Studio, where `chr32` =space and `chr38`=ampersand.*

6.5 Using a Metastorm BPM Process Code-Behind Project

A project created using the Metastorm BPM Process Code-Behind Project template consists of:

- The solution name, defined in the Process Events Wizard.
- The project name which is `Metastorm.ProcessEvents.<ProcedureName>`
- The files within the project. A single component is created for each Map and each Admin Form which contains two partial classes for each process. The files are named `<MapName>.cs` and `<MapName>.Fields.cs`

- A tree view of the processes including action, stages, forms, custom variables and events. The tree view is for informational usage only so that the developer can determine the process context of the events.
- Events skeletons which are created using the procedure, map, form, action and stage names defined in Designer.
- Resynchronization option.


The code skeleton has two dependencies:

- `Metastorm.Engine.Interface75`
- `Metastorm.ProcessEvents.CodeBehindSupport`

The first dependency gives access to the `IFolder` Interface. The second provides support for the auto generated code in the skeleton.

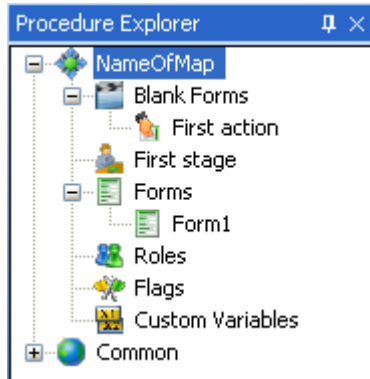
The `IFolder` Interface provides access to functions exposed by the Metastorm BPM language.

Metastorm BPM Code-Behind Project comes with IntelliSense to assist the .NET developer.

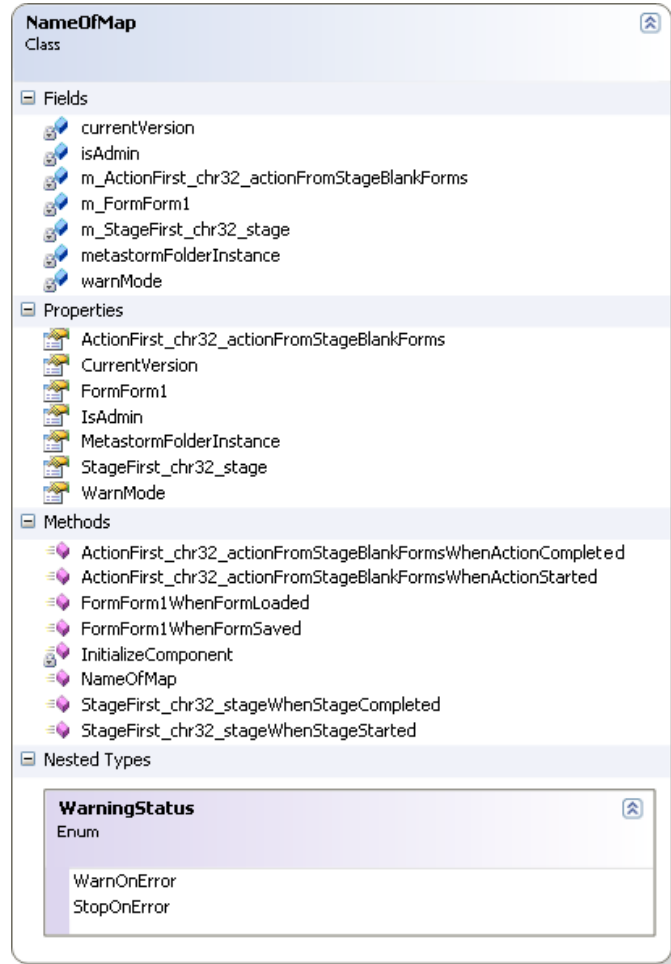
 Refer also to **When Formulas are evaluated** in the Designer User Manual for a description of the order in which process events are fired.

6.5.1 Process Events Class Diagram

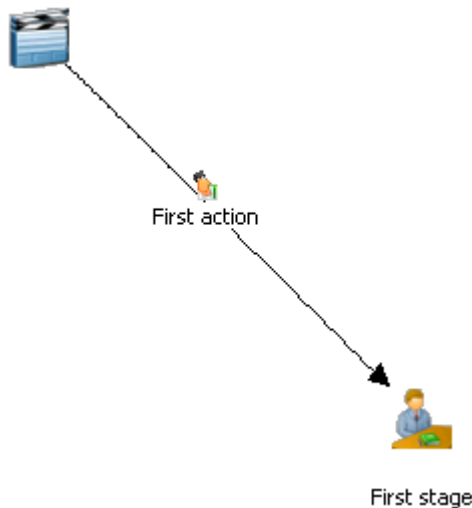
The Process Events Wizard generates a skeleton for the code. The diagrams below show a published procedure with the corresponding .NET code. The **Delegate all external events for map** property has been checked in Designer.



Procedure Explorer in Designer



Class Diagram in Visual Studio



Main Pane in Designer

Figure 20: Procedure in Designer and Visual Studio 2005

*Non alphanumeric characters accepted by Designer in names are replaced with the corresponding character code. For example a Stage named **First Stage** in Designer will be amended to **First_chr32_Stage** in Visual Studio, where chr32 =space.*


6.5.2 Procedure


Each map and each Admin Form in a procedure is split into two partial classes.

- <MapName>.Fields.cs which contains:
 - Private members of the map class
 - The initialization of the map owned objects.
 - Actions

- Forms
- Stages
- CurrentVersion, metastormFolderInstance, WarnMode, IsAdmin
- <MapName>.cs which contains:
 - Properties: - CurrentVersion, metastormFolderInstance, WarnMode, IsAdmin
 - Methods – Events, Initialize Component, MapName
 - Nested Types – Warning Status.
 - Public properties of the map class.
 - Initialization of user configurable values.

The structure is created mirroring the Designer naming conventions and structure.

 *Map Segments are integrated into the parent map so that the segment belongs to the parent's map's class file when added to a Code-Behind project.*

 *Stages, Forms and Action do not provide access to the Metastorm Object Model via `metastormFolderInstance`, this means that the developer is able to use `metastormFolderInstance.Action.StartsStage` but unable to use `ActionName.StartsStage`.*

6.5.3 Stages

Each stage has private variable and public accessor methods. Private stage names are prefixed with “m_”. Variable Names are prefixed with the object type.

Stage Names

```
private Stage m_<ObjectType:Stage><StageName>;
public Stage <ObjectType:Stage><StageName>
    {
        get
        {
            return m_<ObjectType:Stage><StageName>;
        }
        set
        {
            m_<ObjectType:Stage><StageName>= value;
        }
    }
}
```

Events related to stages are in the format:

Start Event


```
public bool <ObjectType:Stage><stageName>WhenStageStarted()
{
    return true;
}
```


Finish Event

```

public bool <ObjectType:Stage><stageName>WhenStageCompleted()
{
    return true;
}

```

 *Non alphanumeric characters that are accepted by Designer in Stage names that are accepted in Designer are replaced with the corresponding character code. For example a Stage named **Process & Review** in Designer will be amended to **Process_chr32_chr38_chr32_Review** in Visual Studio, where chr32 =space and chr38=ampersand.*

 *Rule stages: OnInvokeRule is executed via the preceding **When Action Completed**.*

Common Stages

Common stages are not included by name in a Code-Behind Project. Instead, all actions from a common stage are assigned to the corresponding stage.

For example, there is a common stage called “MyCommonStage” which has a common action “MyCommonAction”. The “MyCommonStage” is applied to “MyFirstStage” and “MySecondStage” resulting in the following code:

```

public Action ActionMyCommonActionFromStageMyFirstStage
{
    get
    {
        return m_ActionMyCommonActionFromStageMyFirstStage;
    }
    set
    {
        m_ActionMyCommonActionFromStageMyFirstStage = value;
    }
}

public bool
ActionMyCommonActionFromStageMyFirstStageWhenActionStarted()
{
    return (true);
}

public bool
ActionMyCommonActionFromStageMyFirstStageWhenActionCompleted(
)
{
    return (true);
}

public Action ActionMyCommonActionFromStageMySecondStage
{
    get
    {
        return m_ActionMyCommonActionFromStageMySecondStage;
    }
}

```

```
    }
    set
    {
        m_ActionMyCommonActionFromStageMySecondStage = value;
    }
}

public bool
ActionMyCommonActionFromStageMySecondStageWhenActionStarted()
{
    return (true);
}

public bool
ActionMyCommonActionFromStageMySecondStageWhenActionCompleted
()
{
    return (true);
}
```

6.5.4 Forms

- Each form will have Private variable and public accessor methods. Private form names are prefixed with “m_”.
- Variable Names are prefixed with the object type so that the variable is not overloaded.

Form names

```
private Form m_<ObjectType:Form><FormName>;
public Form <ObjectType:Form><FormName>
{
    get
    {
        return m_<ObjectType:Form><FormName>;
    }
    set
    {
        m_<ObjectType:Form><FormName>= value;
    }
}
```


Events related to forms will be in the format:

Start Event

```
public bool <ObjectType:Form><FormName>WhenFormLoaded()
{
    return true;
}
```

Finish Event

```
public bool <ObjectType:Form><formName>WhenFormSaved()
{
    return true;
}
```

 Non alphanumeric characters that are accepted by Designer in Form names are replaced with the corresponding character code. For example a Form named **Process & Review** in Designer will be amended to **Process_chr32_chr38_chr32_Review** in Visual Studio, where chr32 =space and chr38=ampersand.

6.5.5 Admin Forms

- Admin Forms are created with their own partial classes with isAdmin set to True.
- Each form will have Private variable and public accessor methods. Private form names are prefixed with “m_”.
- Variable Names are prefixed with the object type so that the variable is not overloaded.

Form names

```
private Form m_<ObjectType:Form><FormName>;
public Form <ObjectType:Form><FormName>
{
    get
    {
        return m_<ObjectType:Form><FormName>;
    }
    set
    {
        m_<ObjectType:Form><FormName>= value;
    }
}
```

Events related to forms will be in the format:

Start Event

```
public bool <ObjectType:Form><FormName>WhenFormLoaded()
{
    return true;
}
```

Finish Event

```
public bool <ObjectType:Form><formName>WhenFormSaved()
{
    return true;
}
```

6.5.6 Actions

- Each action has private variable and public accessor methods. Private form names are prefixed with “m_”.
- Variable Names are prefixed with the object type.
- Creation actions do not have preceding stage names and are differentiated from blank or admin forms.


Start Events


```
public void
<ObjectType:Action><actionName>FromStage<[StageName]
/[BlankForms, AdminForm]>WhenActionStarted()
{
    return true;
}

public void
<ObjectType:Action><actionName>FromStage<[StageName]
/[BlankForms, AdminForm]>OnlyStartActionIf()
{
    return true;
}
```

Finish Event

```
public void
<ObjectType:Action><actionName>FromStage<[StageName]
/[BlankForms, AdminForm]>WhenActionCompleted()
{
    return true;
}
```

 *Action names will be a combination of the StartsWithStageName and the action name*

 *Non alphanumeric characters that are accepted by Designer in Actions names are replaced with the corresponding character code. For example a Stage named **Processing & Reviewing** in Designer will be amended to **Processing_chr32_chr38_chr32_Reviewing** in Visual Studio, where chr32 =space and chr38=ampersand.*

6.5.7 Dependencies

The code skeleton is dependent on:

- `Metastorm.Engine.Interface75` which gives access to the `IFolder` Interface.
- `Metastorm.ProcessEvents.CodeBehindSupport` which provides support for the auto generated code in the skeleton.

The `IFolder` Interface provides access to functions exposed by the Metastorm BPM language.

6.5.8 CurrentVersion

The CurrentVersion is updated when no changes have been made to the procedure and the procedure is re-published in the Designer and resynchronized in Visual Studio 2005.

For example:

A procedure is published with **Delegate all external events for map** checked. The Process Events Integration Wizard is run in Visual Studio 2005 to create code behind. A constructor is automatically created:

```
public <MapName>()
{
    InitializeComponent();
    currentVersion = 1;
    // Set warn mode for version conflicts
    // WarnOnError
    // IgnoreOnError
    // StopOnError
    warnMode = WarningStatus.WarnOnError;
}
```

The procedure is published with no changes and in Visual Studio the code is resynchronized. The code is updated:

```
public <MapName>()
{
    InitializeComponent();
    currentVersion = 2;
    // Set warn mode for version conflicts
    // WarnOnError
    // IgnoreOnError
    // StopOnError
    warnMode = WarningStatus.WarnOnError;
}
```

The current version is checked by the invoker at runtime.

The warn mode control the behavior. For example, this may be a warning in the Designer log, or `StopOnError` may throw an actual exception or add an error.

6.5.9 Private member declarations & Event Handler Initializations

This section shows examples of C# code (`<MapName>.Fields.cs`) automatically generated through the Process Events wizard.

Forms, stages and action methods.

```
this.m_FormForm1 = new Form();
this.m_FormForm1.Name = "Form1";
```

```
this.m_StageFirst_chr32_stage = new Stage();  
this.m_StageFirst_chr32_stage.Name = "First_stage";
```

```
this.m_ActionFirst_chr32_actionFromStageBlankForms = new  
Action();  
this.m_ActionFirst_chr32_actionFromStageBlankForms =  
"First_chr32_action";
```

When Action Completed

```
this.m_ActionFirst_chr32_actionFromStageBlankForms.WhenAction  
Completed +=  
    new  
    BaseObject.FinishEventHandler(this.ActionFirst_chr32_actionFr  
omStageBlankFormsWhenActionCompleted);
```

When Action Started

```
this.m_ActionFirst_chr32_actionFromStageBlankForms.WhenAction  
Started +=  
    new  
    BaseObject.StartEventHandler(this.ActionFirst_chr32_actionFro  
mStageBlankFormsWhenActionStarted);
```

When Stage Completed

```
this.m_StageFirst_chr32_stage.WhenStageCompleted +=  
    new  
    BaseObject.FinishEventHandler(this.StageFirst_chr32_stageWhen  
StageCompleted);
```

When Stage Started

```
this.m_StageFirst_chr32_stage.WhenStageStarted +=  
    new  
    BaseObject.StartEventHandler(this.StageFirst_chr32_stageWhenS  
tageStarted);
```

When Form Loaded

```
this.m_FormForm1.WhenFormLoaded +=  
    new  
    BaseObject.StartEventHandler(this.FormForm1WhenFormLoaded);
```

When Form Saved

```
this.m_FormForm1.WhenFormSaved +=  
    new  
    BaseObject.FinishEventHandler(this.FormForm1WhenFormSaved);
```

6.5.10 Public Accessor Methods / Properties

```
public bool StageFirst_chr32_stageWhenStageStarted()
{
    return true;
}

public bool StageFirst_chr32_stageWhenStageCompleted()
{
    return true;
}

public bool
ActionFirst_chr32_actionFromStageBlankFormsWhenActionStarted(
)
{
    return true;
}

public bool
ActionFirst_chr32_actionFromStageBlankFormsWhenActionComplete
d()
{
    return true;
}

public Stage StageFirst_chr32_stage
{
    get
    {
        return m_StageFirst_chr32_stage;
    }
    set
    {
        m_StageFirst_chr32_stage = value;
    }
}

public bool StageFirst_chr32_stageWhenStageStarted()
{
    return true;
}

public bool StageFirst_chr32_stageWhenStageCompleted()
{
    return true;
}

public Form FormForm1
{
```

```

        get
        {
            return m_FormForm1;
        }
        set
        {
            m_FormForm1 = value;
        }
    }

    public bool FormForm1WhenFormLoaded()
    {
        return true;
    }

    public bool FormForm1WhenFormSaved()
    {
        return true;
    }

```

6.5.11 IntelliSense

To assist the .NET Developer, Metastorm BPM Code-Behind Project has IntelliSense for its properties and methods.

For example:

```
metastormFolderInstance.|
```

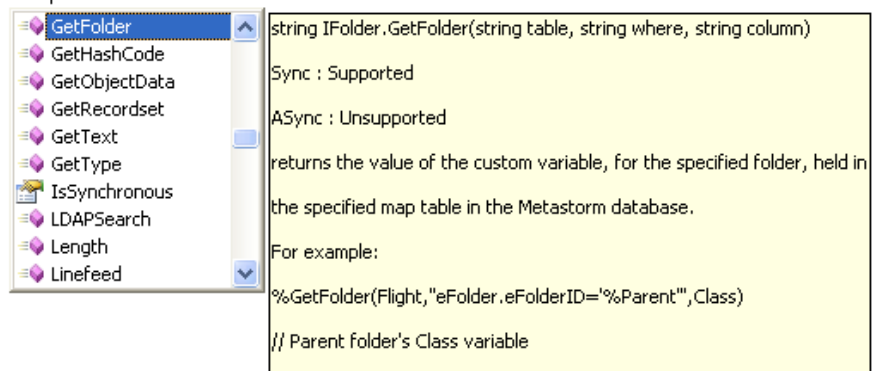



Figure 21: IntelliSense example

 All Integration Wizard functions are available in C# except functions which run VBScript or JScript.NET. These functions can be used in Designer in conjunction with `%ExecuteExtensionEval`.

6.6 Metastorm BPM Process Code Behind Example

The following worked example shows how to populate a dropdown is created in the Designer which contains a list of names.

1. Associate **Process Events Library.xel** with procedure.
2. Set up text custom variables:
 - txtNames
 - txtListOptions
3. Add a dropdown to a form and set the following properties

Component	Property Tab	Properties	Value
Drop-down	Drop-down	Variable	txtNames
Drop-down	Options	List options	%txtListOptions
Form1	Do This	When user loads form	In the Integration Wizard select: Event Handlers and External Event Handler .

Table 4: Component Property Settings

4. Publish the procedure
5. In Visual Studio 2005, create a Metastorm BPM Process Code-Behind Project using the published procedure.
6. Find `public bool FormForm1WhenFormLoaded()`
7. Amend as follows:


```
public bool FormForm1WhenFormLoaded()
{
    metastormFolderInstance.CustomVariable("txtListOptions",
    "Richard,Katherine,Pedro,Bami");
    return true;
}
```
8. Build the project
9. Deploy the project
10. In Internet Explorer, view the form and the dropdown is populated with the list defined in Visual Studio .NET.

6.7 Configuring Process Metadata Web Service Connection Details

Internet Information Services (IIS) is configured to include the Process Metadata Web Service called **Metastorm.Common.ProcessMetadataService** which sets its default authentication method as **Integrated Windows Authentication**. The URL is defined in a Visual Studio

Integration configuration files. The Visual Studio Integration requires the Metastorm Process Metadata Service.

Before the Process Metadata Web Service can be configured the following need to be set:

- Security
- Process Engine Database connection settings

6.7.1 Security

The Visual Studio 2005 .NET Developer requires access to the Metastorm Database (SQL or Oracle) using normal authentication methods.

The web service that is invoked has to have the necessary Internet Information Services (IIS) security set by an IIS administrator.

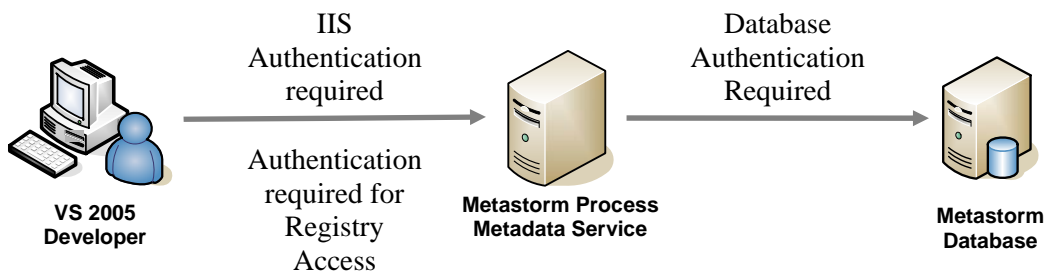


Figure 22: Metastorm Process Metadata Service Security

When the Visual Studio 2005 .NET developer deploys the dll to the engine machine, the developer will need to have the necessary NTFS permissions to deploy an assembly in relevant folder.

6.7.2 Process Engine Database Connection Settings

The Metastorm Process Metadata Service uses the Process Engine's database connection settings. For the service to have access to the configuration information, certain user accounts need to have read permission to the following registry keys:

- HKLM\Software\Metastorm\Engine
- HKLM\Software\Metastorm\Engine\Database
- HKLM\Software\Metastorm\Engine\Database Connectors & sub keys.

By default, the service uses Integrated Windows Authentication. This means that the service impersonates the currently logged on user, so all Visual Studio users must have read access to these keys.

6.7.3 Authentication


When installing the service the currently logged on user is added by default. In development environments where there is more than one person accessing the service each developer will need to have read permissions for the registry. The best practice would be to create a user group, such as Metastorm Developers and grant permissions to this group. Further users can be added to this group as required.

Alternatively, the service can be accessed using Anonymous authentication. This can be changed in the IIS administration console. In this case permissions are given to the Anonymous user only. This typically is the I_USR<machinename user>.

There are some issues to be aware with regard to service and database configurations:

1. The service authentication type (Integrated Windows Authentication, Basic and Anonymous) is not important, when access to the Metastorm database is configured as SQL Authentication.
2. When accessing a SQL Server database using trusted connections (Integrated Authentication) the following apply:
 - i. Integrated Windows Authentication – all users accessing the service must be authenticated by SQL Server.
 - ii. Anonymous – You will need to make sure the I_USR<machinename user> can be authenticated by SQL Server
 - iii. Basic Authentication - prompts the user for their credentials.
3. SQL Authentication (nominating a user id and password to connect to the database) is the only mode supported when accessing Oracle databases from the Metastorm Process Metadata Service.

 *Basic Authentication credentials are sent via clear text.*

 *The Metastorm Process Metadata Service is required during development and is not used in production. It is recommended that customers test their development on non-production machines.*

6.7.4 Process Metadata Web Service Configuration Details Dialog

The Metastorm Process Metadata Service can be accessed from the Metastorm Process Events Code-Behind Wizard or through the Process Events add-in in Visual Studio.

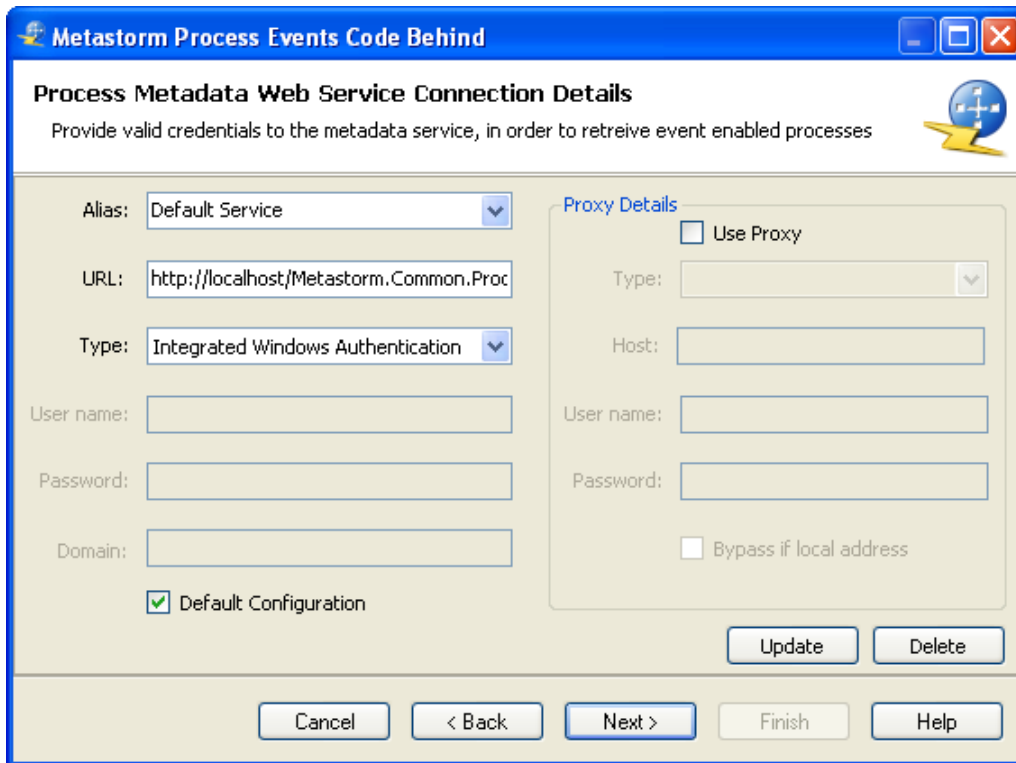


Figure 23: Metastorm Process Metadata Service Configuration Dialog in the Wizard

In Visual Studio:

1. **Tools | Add-in Manager**
2. Select **Process Events Solution Support**
3. Click OK
4. **View | Metastorm Process Events Web Service Details**

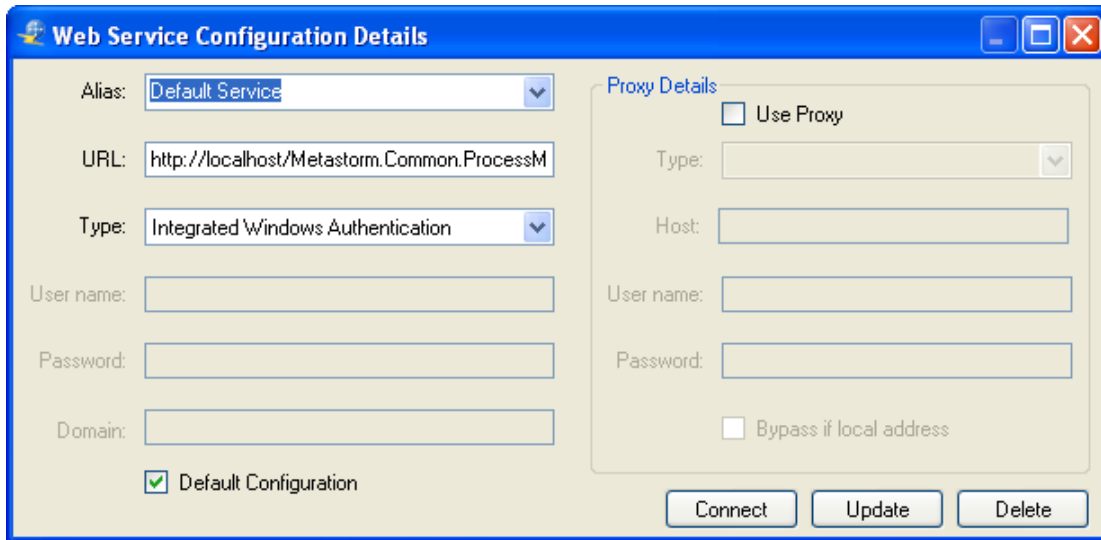


Figure 24: Web Service Configuration Details accessed from the menu


Connection Parameters

Option	Summary	Remarks
Alias (string)	This is a user defined name for the Web Service.	
URL (string)	The location of the Process Metadata Service. The default service name is displayed. It is obtained from the Process Activator's and VS Integration configuration files.	The Metastorm Process Metadata Service can run on a separate engine machine. If this is the case, the URL will reference the engine machine.
Type	The authentication used by the Metastorm Process Metadata Service.	The following authentication methods are supported: <ul style="list-style-type: none"> Anonymous - a dedicated local account is used to server the request. Basic Authentication (requires User name, Password and Domain) Integrated Windows Authentication, the default. The service impersonates the credentials of the user accessing the service. Refer to 6.7 Configuring Process Metadata Web Service Connection Details for more details.
User name (string)	Sets the user name used for basic authentication.	
Password (string)	Sets the password used for basic authentication.	
Domain (string)	Sets the domain used for basic authentication.	Remember that password information is sent in plain text over the network when using basic authentication.
Default Configuration (boolean)	Sets the displayed Web Service as the default Process Metadata Web Service. The alias name of the default configuration is moved to the top of the Alias dropdown.	

Proxy Details

Option	Summary	Remarks
Use Proxy (boolean)	Sets the information for the proxy server used for all web services calls.	
Type	Sets the proxy type.	<ul style="list-style-type: none"> The proxy server can be configured into two ways: CURRENT_USER - the current user's IE settings are used to determine the proxy. host:port -the proxy at host:port is used (for example. http://93.13.17.2:8080). These settings override the IE settings.
Host	The host port of the proxy. For example: http://93.13.17.2:8080.	Http must be used in front of definition.
User name (string)	Sets the username used when accessing the proxy server.	
Password (string)	Sets the password used when accessing the proxy server.	
Bypass if local address (boolean)	Determines whether the proxy server is bypassed when calling local resources.	<p>If checked, requests to local Internet resources do not use the proxy server. Local requests are identified by the lack of a period (.) in the URI, as in http://webserver/ or access the local server, including http://localhost, http://loopback, or http://127.0.0.1 .</p> <p>If not checked, all Internet requests are made through the proxy server.</p>

Buttons


Options	Remarks
Connect	<p>This button checks if the defined web service exists and uses this connection. An error message is displayed if the web service fails to connect.</p> <p> <i>The Connect button is only available when using the Process Events Solution Support Add-in. In the wizard, the web service connection is tested when the user clicks Next.</i></p>
Update	This button updates the displayed web service configuration details.
Delete	This button deletes the displayed web service configuration details.

6.8 Add-in

Process Events is distributed with a Process Events Solution Support Add-in. The Add-in provides Visual Studio 2005 support for Metastorm BPM Code Behind Process Events Solutions and contains the following menu options:

- Metastorm Web Service Configuration Details.
- Deploy Process.
- Debug With Process

- Resynchronizing a Project
- Solution Browser.

 When *Process Events* is installed the Add-in is stored in the current user's *My Documents\Visual Studio 2005\AddIns* folder. If a different user uses *Process Events*, the file *Metastorm.ProcessEvents.SolutionSupportAddin.AddIn* will need to be copied to the current user's *My Documents\Visual Studio 2005\AddIns* folder.

6.8.1 Using the Add-in:

1. Select **Tools | Add-in Manager**
2. Select **Process Events Solution Support**
3. Click **OK**.
4. The **View** menu contains two new options:
 - i. **Browse Process Events** – this option displays the Solution Browser depicting Designer's Procedure Explorer in graphical format.
 - ii. **Metastorm Process Events Web Service Details** – this option displays the Metastorm Web Service Configuration Details which set the location of the Web Service.
5. Solution Explorer contains two new options in the project context menu:
 - i. **Deploy Process** – this option copies a built project's assemblies (DLL's) to a location (by default *engine\dotnetbin*) for use with the published procedure.
 - ii. **Resynchronizing a Project** – this option resynchronizes the .NET process events code with the published procedure.
 - iii. **Debug With Process** – this option enables a .NET developer to debug a process events project.

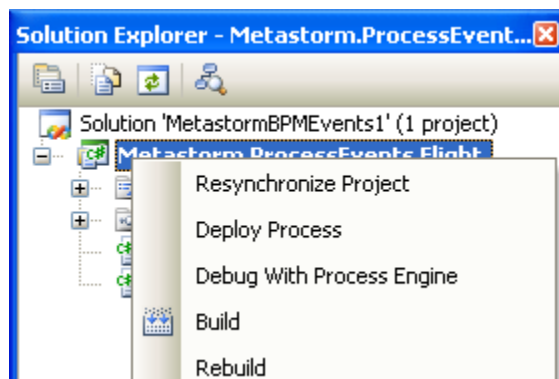


Figure 25: Process Events Menu Options

6.8.2 Metastorm Web Service Configuration Details

Each of the features requires a connection to the process metadata service. The solution Add-in attempts to auto connect to a process metadata xml file, which is loaded on first attempt. If none of the features work, you can reconfigure the web service to a valid connection.

To view the Web Service Configuration details:

1. From the **View** menu select **Metastorm Process Events Web Service Details**.
2. The **Web Service Configuration Details** dialog is displayed.

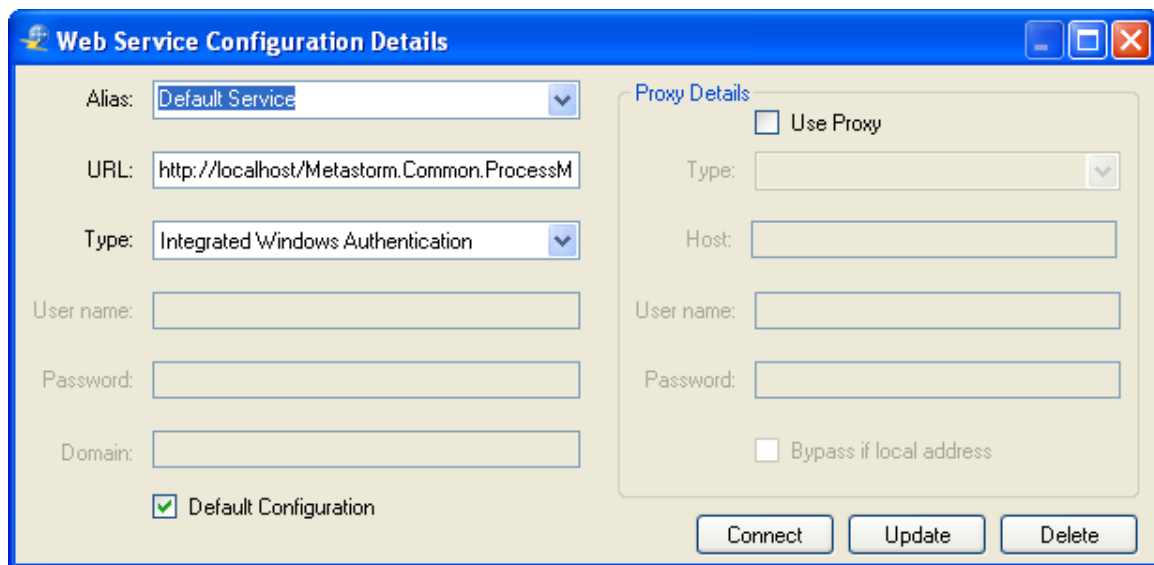


Figure 26: Web Service Configuration Details Dialog

This screen displays a list of available Metastorm Process Metadata Services.


It is assumed that the web service is installed on the same machine as the engine.

6.8.3 Deploy Process

The Deploy option requires the project to be built. The assembly files are then copied from the project's bin\build folder to the Deployment Target Path. The default paths are C:\Program Files\Metastorm BPM\Engine\donetbin or C:\Program Files\Metastorm BPM\Engine\donetlib.

The assembly cache uses the engine defined in the Metadata Service Configuration file.

The Deploy option, stops and restarts the engine if it running. An "Engine Status" label indicating progress is displayed in the Deploy Project Files dialog. The engine is restarted if a procedure has been run before and the assembly is loaded in the script hosts appdomain assembly cache. The cache is flushed by stopping the engine, deploying the new assembly and restarting the engine.

 The Deploy option is recommended for use on a developers machine and not in a production environment.

To deploy a project:

1. Load the Process Events Add-in
2. Open Solution Explorer.
3. Right click the Project Name.
4. Select **Build**.
5. Right click the Project Name.
6. Select **Deploy Process**.
7. The Deploy Process dialog is displayed.
8. Check the **Files to Deploy** and the **Deployment Target Path**.
9. Click **Deploy**.

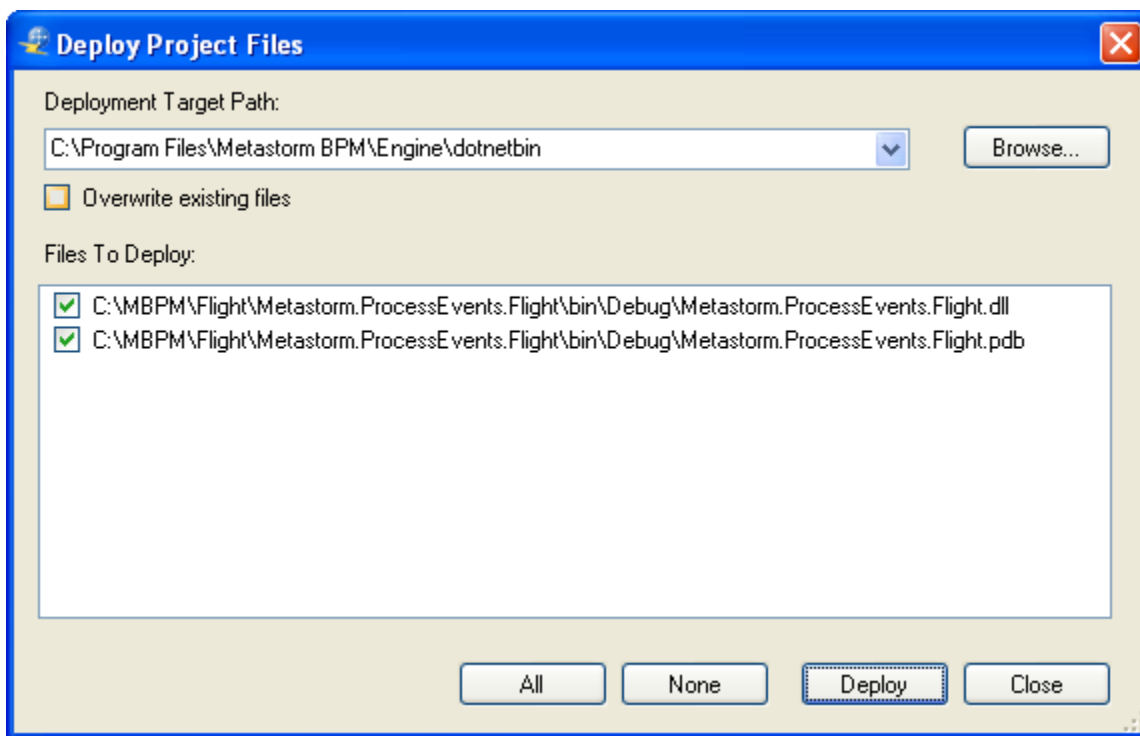


Figure 27: Deploy Project Files dialog

Deploy Project Files

Deployment Target Paths

The target path of the assembly files which are copied from the project's bin\build folder to the defined Deployment Target Path. The default paths are

C:\Program Files\Metastorm BPM\Engine\donetbin or

C:\Program Files\Metastorm BPM\Engine\donetlib.

Overwrite existing files

Overwrites existing files in the Deployment Target Path folder.

Files to deploy

The files needed to deploy Process Events are the dll and pdb file for the appropriate project.

Buttons

All

This button selects all files in **Files to deploy**.

None

This button unchecks all selected files in **Files to deploy**.

Deploy

This button deploys the selected files to the defined Deployment Target Path.

Close

This button closes the dialog.


6.8.4 Debug With Process Engine


The **Debug With Process Engine** option enables a .NET developer to debug a process events project. If the engine is running the **Debug With Process Engine** menu option will attempt to attach the code behind the dll to the engine process. Users can then detach from the process using the **Detach All** menu option.

To debug the Process Engine:

1. Deploy the Project
2. In the Solution Browser, right click and select **Debug With Process Engine**

When the .NET developer has finished debugging they should select from the menu, **Debug | Detach All**.

 **Debug With Process Engine** is not supported with Split deployment.

 The **Debug With Process Engine** option is not accessible using Vista. In order to attach to an engine in Vista, you must attach to process using the standard Visual Studio **Attach to Process** option.

6.8.5 Resynchronizing a Project


When a process designer changes a procedure and republishes the procedure, the Visual Studio 2005 developer should resynchronize a project to include or exclude any methods that may have changed.

The Solution Add-in, Resynchronization obtains a data set from the database and uses it to amend an existing Visual Studio solution which was previously generated by the Process Events Wizard.

The resynchronization process displays the following information:


- An updated version of the project against the latest published data.
- New forms, actions or stages events (from local or global).
- Any stages, actions, forms or maps that have been removed or renamed in the published procedure are marked as obsolete. Obsolete stages are not removed from the Metastorm database as stages (folders) may be on Users' To Do / Watch list.

 A Metastorm Services Administrator can purge stages that are no longer being used, using the purge stages option on a map using the Metastorm Services Manager tool.

 For further information on purging stages, refer to the Administration Guide

There are three conditions that must be met to assume that a code element represents a data set for Stage, Form or Action entities, past or present.

- The code element must be a function whose name can be deconstructed into components.
- These components can be used to construct a variable name and property name, both of which must exist in the corresponding class.

 At invocation time, if the Visual Studio 2005 developer has set to `WarnMode.StopOnError` and there is a procedure that has not been resynchronized, the procedure will fail to execute.

Steps to Resynchronize a Project

To resynchronize a project:

1. Open **Solution Explorer**.
2. Right click the Project Name.
3. Select **Resynchronize Project**.
4. The Resynchronize Project dialog is displayed.

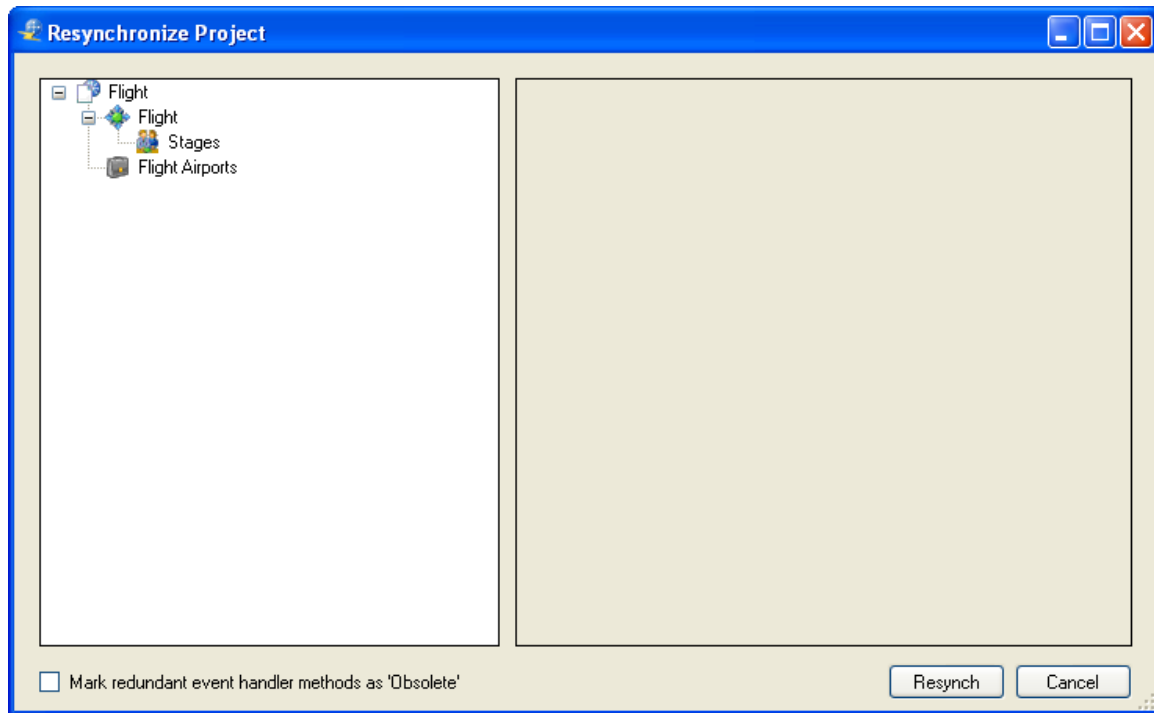






Figure 28: Resynchronize Project dialog

5. Check **Mark redundant event handler methods as 'Obsolete'** as required.
6. Click the **Resynch** button.
7. Rebuild the project.
8. Redeploy the project.

 *If the Procedure no longer exists, the .NET Developer will be warned that a project in a solution is no longer valid.*

 *If a Procedure is already in use it may be necessary to stop the process engine before redeploying the process.*

 *By default, clicking the **Resynch** button automatically saves the solution.*

 *The Start Page recent projects are not currently supported with resynchronization. The VS2005 developer will need to refresh their events after loading a project using the Start Page.*

Renaming a Map

If the map has been renamed, a new code behind unit is created.

Renaming a Stage or Action

When a project is resynchronized the option **Mark redundant event handler methods as 'Obsolete'** in the **Resynchronize Project** dialog can be used to mark any stages or actions that have been renamed. As the stage or action is no longer available to a process any related events are marked as obsolete in the code left behind and new stages and actions are added to the project..

The code behind class creates new events for the renamed form and places `Obsolete` around the existing code.

```
[Obsolete("Event Handler is obsolete, event no longer
valid")]
public bool
ActionAction1FromStageFirst_chr32_stageWhenActionStarted()
{
    return true;
}
```

The old stage, form or action events are not deleted, for backwards compatibility because a procedure may already be in use and therefore data may be held in the database against a specific stage or action.

Deleting a Stage

If a stage has been deleted from the procedure, the stage will remain in the Metastorm database if a User's To Do or Watch Lists is populated with the deleted stage unless the folder is purged using the Metastorm System Administrator.

Deleting an Action

If an action has been deleted from the procedure, the action will be marked as obsolete.

Creating a new Stage, Action or Form

When a project is resynchronized, any new stages, actions, forms and their associated events are added to the project.

6.8.6 Solution Browser

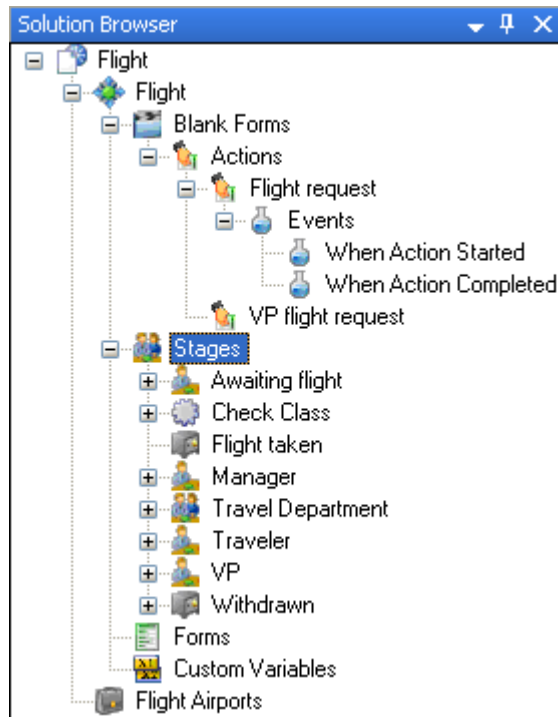
The solution browser provides an option to view a tree-view displaying a Metastorm BPM view of Procedures, Maps, Stages, Forms and Actions and events. A Visual Studio 2005 developer can navigate to a function by double clicking on an event.

The tree view also displays the available custom variables for each map (project), which the developer can access via the `metastormFolderInstance` member variable, for example:

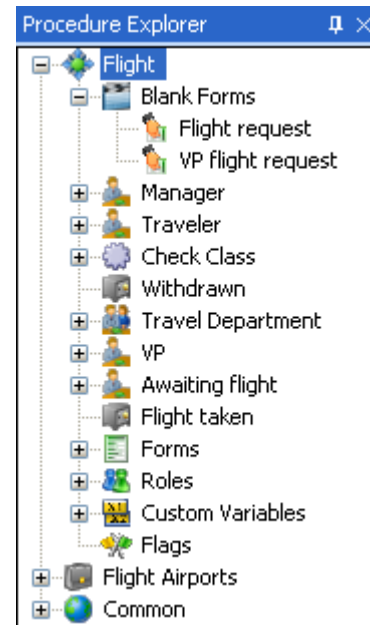
```
metastormFolderInstance.CustomVariable("txtCustomVariableName",
    "Text to set the variable");
```

To view the Solution Browser:

1. From the **View** menu select **Browse Process Events**.
2. The **Solution Browser** is displayed.




Solution Browser in Visual Studio .NET




Procedure Explorer in Designer

Figure 29: Tree hierarchy in Visual Studio .NET and Designer

 On resynchronizing a project the Solution Browser will refresh its window with the new populated Maps, Actions, Stages, Forms and Events added by the resynchronization.

Appendix A – Simple Type Mapping

The following table summarizes the simple type mapping that is applied when .NET functions are imported.

 At present, the Metastorm Process Orchestrator for .NET supports only simple types.

.NET Type	Metastorm Type
System.Int32 System.Int16	Integer
System.Decimal	Currency
System.Single System.Double	Real
System.Boolean	Check
System.DateTime	Date-time
System.Char	Text
System.String	Memo

Figure 30: Simple Type Mapping

Appendix B – Examples of Scripts Generated by .NET Activator

The .NET Activator creates 2 scripts containing the functions to access the assembly which are stored in a Metastorm library. The scripts are called:

- Activator [Assembly Name] - a class to invoke an assembly using data from a Metastorm process.
- AvailableFunctions [Assembly Name] - a proxy class containing all available functions, but commented out.

Activator [Assembly Name]

```
/* (C) 2005 Metastorm Inc
   JScript.NET Activation Script
*/

import System;
import eWork.Engine.ScriptObject;
import eWork.Engine.Activator;
import System.Reflection;

package eWork.Activator.TestClass
{
    public class Invoker
    {
        public static function Activate(
ework:SyncProcessData, args: Object[] ) : Object
        {
            var retval : Object = null;
            retval = ClassActivator.Activate(ework,
args);
            return retval;
        }
    }
}
```

AvailableFunctions [Assembly Name]

The commented code is intended to be a template for developers, containing the available functions of the class in JScript.NET. It is not intended to be used directly. Due to JScript.NET limitations, it may have to be altered to compile correctly.

The following are supported:

- Public Methods
- Property accessor & mutator methods
- Indexed properties

- Hidden constructors
- Multiple constructors
- Multiple Namespaces
- Delegates
- Static and non static functions and properties
- Depreciation
- Nested classes

The Activator makes use of the Metastorm synchronous process data object to activate .NET classes. Using the commented code, it is possible to use asynchronous scripts, if necessary.

This example of a generated script uses a complex type. The script is based on the following object:

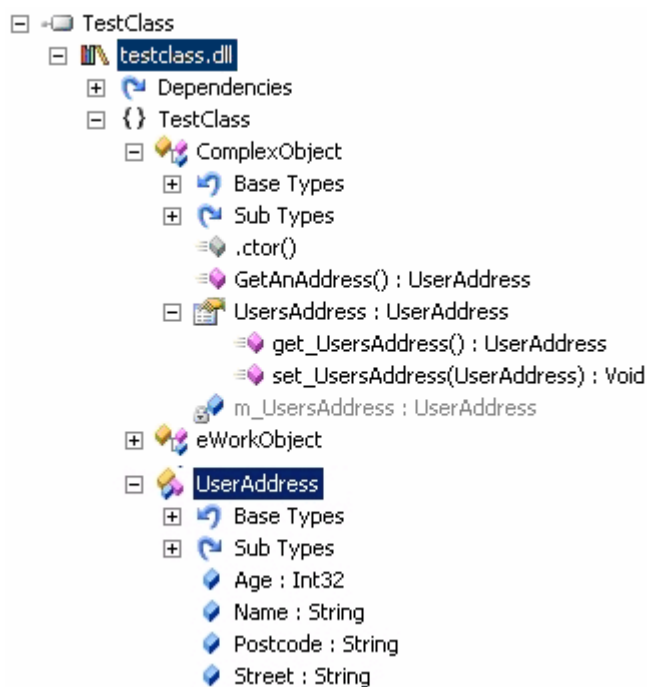


Figure 31: TestClass Object used in the Example Script

```
* (C) 2005 Metastorm Inc
JScript.NET Activation Script
import System;
import TestClass;
import System.Reflection;

package Activator.TestClass
{
    public class TestClassComplexObjectProxy
    {
```

```
        private var m_AssemblyComplexObject :
TestClass.ComplexObject;
        public function TestClassComplexObjectProxy()
        {
            m_AssemblyComplexObject = new
TestClass.ComplexObject()
        }
        public function GetHashCode(): Int32
        {
            return m_AssemblyComplexObject.GetHashCode();
        }

        public function Equals(obj : Object): Boolean
        {
            return m_AssemblyComplexObject.Equals(obj);
        }

        public function ToString(): String
        {
            return m_AssemblyComplexObject.ToString();
        }

        public function get UsersAddress():
TestClass.UserAddress
        {
            return m_AssemblyComplexObject.UsersAddress;
        }

        public function set UsersAddress(value :
TestClass.UserAddress)
        {
            m_AssemblyComplexObject.UsersAddress = value;
        }

        public function GetAnAddress(): TestClass.UserAddress
        {
            return m_AssemblyComplexObject.GetAnAddress();
        }

        public function GetType(): Type
        {
            return m_AssemblyComplexObject.GetType();
        }
    }
}
*/
```