Wonderware

MES Middleware Extensibility Hooks

1/8/13

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200

http://www.wonderware.com

For comments or suggestions about the product documentation, send an e-mail message to ProductDocumentationComments@invensys.com.

Contents

Documentation Conventions 5
Technical Support 6

Chapter 1 Getting Started 7
Using This Guide 7
Definitions 7
Purpose 8
Custom Action Scenarios Using Hooks 9

Chapter 2 Functional Overview 11
Execution Flow Diagram 12
Transaction Control 13
Stored Procedure 13
Assembly 14

Chapter 3 Configuring Pre-Hooks and Post-Hooks 15
Installation Location 15
Opening the Middleware Configuration Editor 15
Middleware SP Name 16
Pre-Hook/Post-Hook Name 17
Stored Procedure 17

Assembly 18

Assembly Is Not Added to the GAC 18

Assembly Is Added to the GAC 18

Pre-Hook and Post-Hook Type 19

Middleware Flow (Pre-Hook/Post-Hook) 19

Stop 19

Stop on Error 20

Continue Even on Error 22

Loading the Configuration and Hooks 23

Examples of Middleware Hook Configurations and Executions 23

Custom Stored Procedure Hook 23

Content of the Custom Stored Procedure sp_I_Uom_Log 24

Action 24

Custom Stored Procedure Hook Raises an Error (Middleware Flow: Stop) 25

Content of the Custom Stored Procedure sp_I_Uom_Log 25

Action 26

Custom Assembly That Is Not in the GAC 26

Content of the Code in the TestHooksClass.cs 26

Action 28

Custom Assembly Is Added to the GAC 28

Content of the Code in the TestHooksClass.cs 28

Action 29

Chapter 4 Configuring Extensibility Hooks on Multiple Middleware Servers 31

Chapter 5 Frequently Asked Questions 33

Welcome

This guide describes how to use the Wonderware MES Middleware Extensibility Hooks mechanism to invoke an assembly or a stored procedure to perform custom actions before and/or after a specific middleware event is executed.

You can view this document online or you can print it, in part or whole, by using the print feature in Adobe Acrobat Reader.

This guide assumes you know how to use Microsoft Windows, including navigating menus, moving from application to application, and moving objects on the screen. If you need help with these tasks, see the *Microsoft Online Help*.

Documentation Conventions

This documentation uses the following conventions:

Convention	Used for
Initial Capitals	Paths and file names.
Bold	Menus, commands, dialog box names, and dialog box options.
`Monospace`	Code samples and display text.

Technical Support

Wonderware Technical Support offers a variety of support options to answer any questions on Wonderware products and their implementation.

Before you contact Technical Support, refer to the relevant section(s) in this documentation for a possible solution to the problem. If you need to contact technical support for help, have the following information ready:

- The type and version of the operating system you are using.
- Details of how to recreate the problem.
- The exact wording of the error messages you saw.
- Any relevant output listing from the Log Viewer or any other diagnostic applications.
- Details of what you did to try to solve the problem(s) and your results.
- If known, the Wonderware Technical Support case number assigned to your problem, if this is an ongoing problem.

Chapter 1

Getting Started

The Wonderware MES Middleware Extensibility Hooks is a mechanism for invoking an assembly or a stored procedure to perform custom actions before and/or after a specific middleware event is executed.

Using This Guide

The purpose of this guide is to help you use the Wonderware MES Middleware Extensibility Hooks mechanism.

This chapter introduces and defines concepts used in the manual, and describes the purpose for Middleware Extensibility Hooks.

Subsequent chapters describe how the hooks function with the middleware, explain how to configure hooks, and provide configuration and execution examples.

Definitions

- A *middleware event* is a method exposed by the MES middleware and that method can be invoked by the client layer (e.g., stateless API, etc.) to perform some action in the MES database.
- A *hook* represents a custom functionality implemented by the end user in the form of a stored procedure, assembly, workflow, etc. A pre-hook represents a custom functionality implemented by the end user to execute that functionality before the middleware event is executed. A post-hook represents a custom functionality implemented by the end user to execute that functionality after the middleware event is processed.

Purpose

The purpose of the Middleware Extensibility Hooks is to enable an end user to execute custom functionalities, such as invoking an ArchestrA workflow, calling a custom stored procedure, or calling a method in an assembly, before or after processing the middleware event. This allows the end user to use their custom functionalities to check for specific conditions or perform certain processes beyond those supported by standard Wonderware MES functionality.

The Extensibility Hooks configuration is stored in the middleware folder, and it is loaded into the Wonderware MES middleware host's memory to execute the hooks when a corresponding middleware event is made to the MES middleware.

The middleware can be configured for a set of one or more "hooks" against a single middleware event. That is, when the Extensibility Hooks are configured for a middleware event, the MES middleware executes the pre-hook before the middleware event is processed, processes the middleware event, and then executes the post-hook after the middleware event. The hook configurations provide an option to cancel the entire middleware event, including the executions of pre- and post-hook in case of an error. The pre-hook configuration also provides an option for an end user to stop processing the middleware event, if the standard middleware functionality is embedded inside the custom implementation.

Custom Action Scenarios Using Hooks

There are many scenarios where the extensibility hooks can be used to check for specific conditions, process custom actions to control the flow of middleware event, and so on. The examples listed in the table below explain how the hooks can be used to perform custom actions.

Hook Intercept	Condition/Process	Hook Type	Example Scenarios
sp_U_Process_CheckIn (Process.CheckIn)	Process Status changed from or to Certified. Changing the status of a process from or to Certified requires a user to have special permission/privilege.	Pre-Hook	A pre-hook can intercept the incoming middleware call (e.g., Process.CheckIn) to validate whether the user has a special permission to change the process status from/to certified, and act based on the outcome from the custom validation.
sp_I_Cert_User_Link (Cert_User_Link.Add)	A visual inspection of user's certification is required before the certification is attached to the user.	Pre-Hook	A pre-hook can intercept the incoming middleware call (e.g., Cert_User_Link.Add) to send an email (using workflow) to the factory supervisor for authorization before the certification can be attached to the user.
sp_U_Job_Exec_StrtDataEntryJob (Job_Exec.StartDataEntryJob)	Creating a new work order that is not instantiated from a process requires authorization from the supervisor.	Pre-Hook	A pre-hook can intercept the incoming middleware call (e.g., Job_Exec.StartDataEntryJob) to initiate a workflow for the supervisor to authorize the transaction before the data entry job is created in the MES database.
sp_U_Storage_Exec_AddInv (Storage_Exec.AddInv)	Push the inventory quantities to the ERP.	Post-Hook	A post-hook can do the post processing on the Storage_Exec.AddInv call to push the inventory quantities to an external application/database.

Chapter 2

Functional Overview

A pre-hook and the post-hook can be configured for a single middleware event to invoke one or more stored procedures (if more than one stored procedure call is embedded inside the single custom stored procedure configured into an Extensibility Hook; see the second note below for more details) or they can be configured to invoke a .NET assembly to handle more complex activities. When invoking a stored procedure, the custom (hook) stored procedure must exist in the MES database targeted by the middleware server. Similarly, when invoking a .NET assembly, the assembly must exist in the specified path (see "Assembly" on page 18 for more details about assembly configuration).

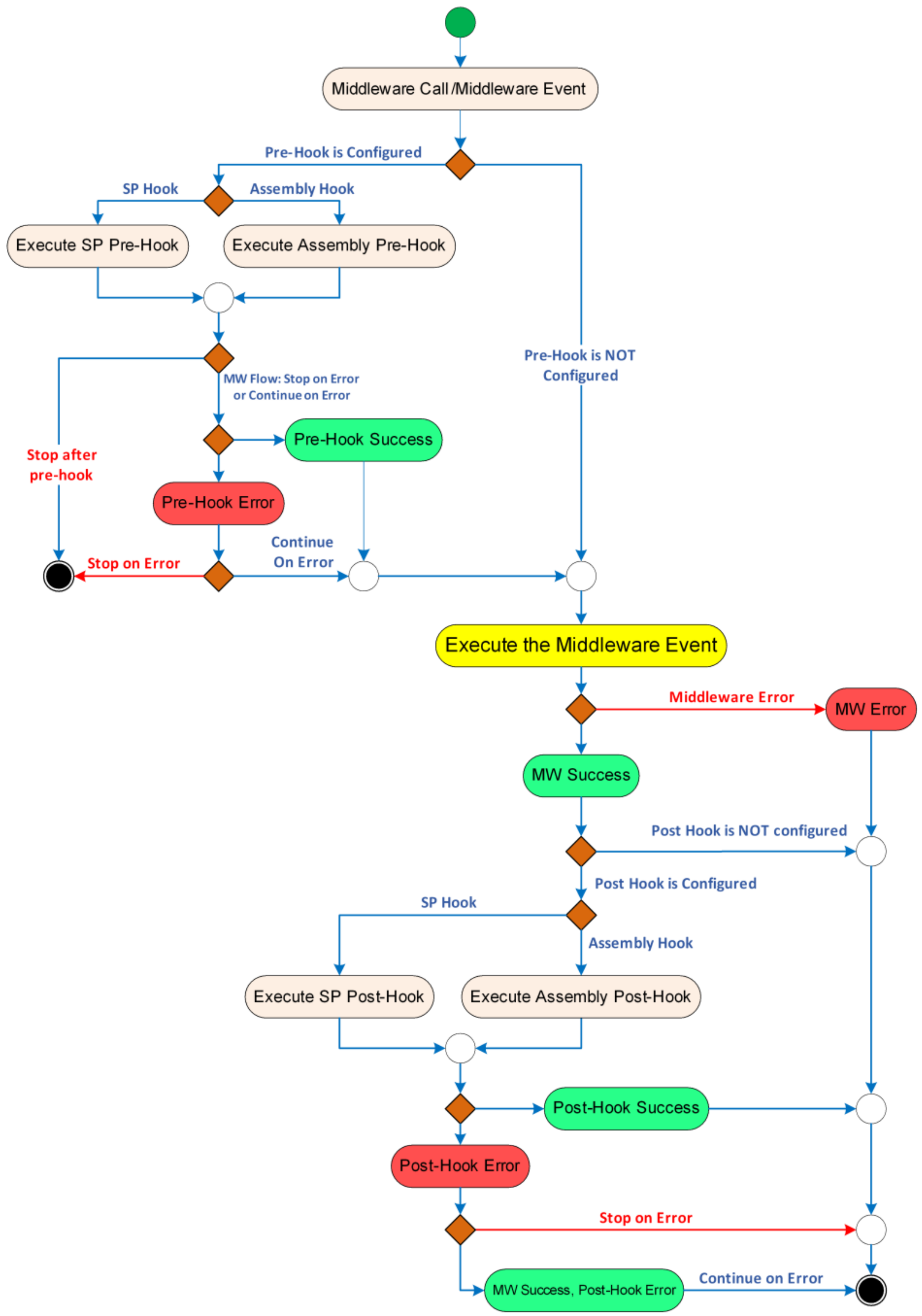
Both pre- and post-hooks can be configured to either continue or stop if an error is encountered. Pre-hooks can also be configured to stop after executing and not allow the normal stored procedure to be called, even in the absence of an error. See "Middleware Flow (Pre-Hook/Post-Hook)" on page 19 for more information.

Notes

- A single middleware event can have only one pre-hook and one post-hook.
- To execute more than one stored procedure for a single middleware event, all the custom stored procedures must be invoked from inside the custom stored procedure named in the hook. Similarly, to invoke more than one assembly for a single middleware event, all the intended assemblies must be invoked inside that single custom assembly named in the hook.
- If a pre-hook or a post-hook is not linked to a middleware event, then that middleware event is not affected by any custom hook implementations inside or outside the MES database.

Execution Flow Diagram

The following diagram illustrates the flow of the pre-hook, middleware event, and the post-hook.



Transaction Control

If pre- and/or post-hooks are configured, then the flow of the middleware event (i.e., whether to commit or rollback, stop or continue with the flow) is controlled by the Middleware Flow parameter [see "Middleware Flow (Pre-Hook/Post-Hook)" on page 19 for more information]. If an exception is raised from the pre- or post-hooks or from the middleware event itself, and Middleware Flow is set to Stop on Error, then the current transaction is rolled back, including the transactions from pre- and post-hooks. The end user is responsible for determining when to raise an error from the stored procedure/assembly and for controlling the flow of the middleware event.

The stored procedures/assemblies involved for the pre- and post-hooks are executed as a part of the middleware transaction that corresponds to the middleware event. In general, the stored procedures/assemblies involved for the pre- and post-hooks are executed as a part of the middleware transaction that corresponds to the middleware event. The total transaction time allotted for the middleware event is split among the pre-hook, middleware event, and the post-hook. However, if the Middleware Flow parameter is set to Continue Even on Error, then each event (i.e. pre-hook, normal middleware event, post-hook) is executed in a separate transaction. In this case, the transaction time-out that is configured applies to each transaction.

Stored Procedure

The transaction that corresponds to the middleware event needs to complete executing the custom stored procedure configured for the pre-hook, complete executing the middleware stored procedure, and complete executing the custom stored procedure configured for the post-hook. All three of these stored procedures must be executed within the allotted transaction time; otherwise, the middleware raises a timeout exception to the caller.

It is not recommended to have an autonomous transaction or nested transactions inside custom stored procedures, because that might affect the transaction flow built inside the Wonderware MES product.

Assembly

A transaction started by the middleware event needs to be completed within the allotted time. The amount of time allotted for the middleware transaction time includes the time the middleware takes to execute custom pre-hook assembly, custom post-hook assembly, and the middleware call itself. Therefore, end users must be cautious when writing custom assembly hooks such that they do not cause any performance bottlenecks when executing the actual middleware call. If the code in the assembly involves processing the files or executing long running transactions, then it is recommended to execute those processes on a background thread, so that the pre- or the post-hook processes do not block the main thread. However, the transactions started on a separate thread (e.g., background thread) are not part of the transaction started on the main thread (i.e., Middleware Event thread). Therefore, the success or failure on the secondary thread does not affect the transaction on the main thread as long as they do not create deadlocks.

Chapter 3

Configuring Pre-Hooks and Post-Hooks

Installation Location

The Middleware Extensibility Hooks mechanism is installed as a part of standard Wonderware MES installation on machines where the MES middleware is installed. If default location was used for the MES installation, this application/editor (MWDBMappingEditor.exe) can be found in the following folder on the machine that is hosting the MES host (i.e., the machine that is hosting the middleware server):

C:\Program Files\Wonderware\MES\Middleware\

However, the installation path can vary if the MES middleware (MES Host) is installed on a different drive or in a different folder. Therefore, look for this application in your corresponding installation path, which typically ends with **\MES\Middleware\**.

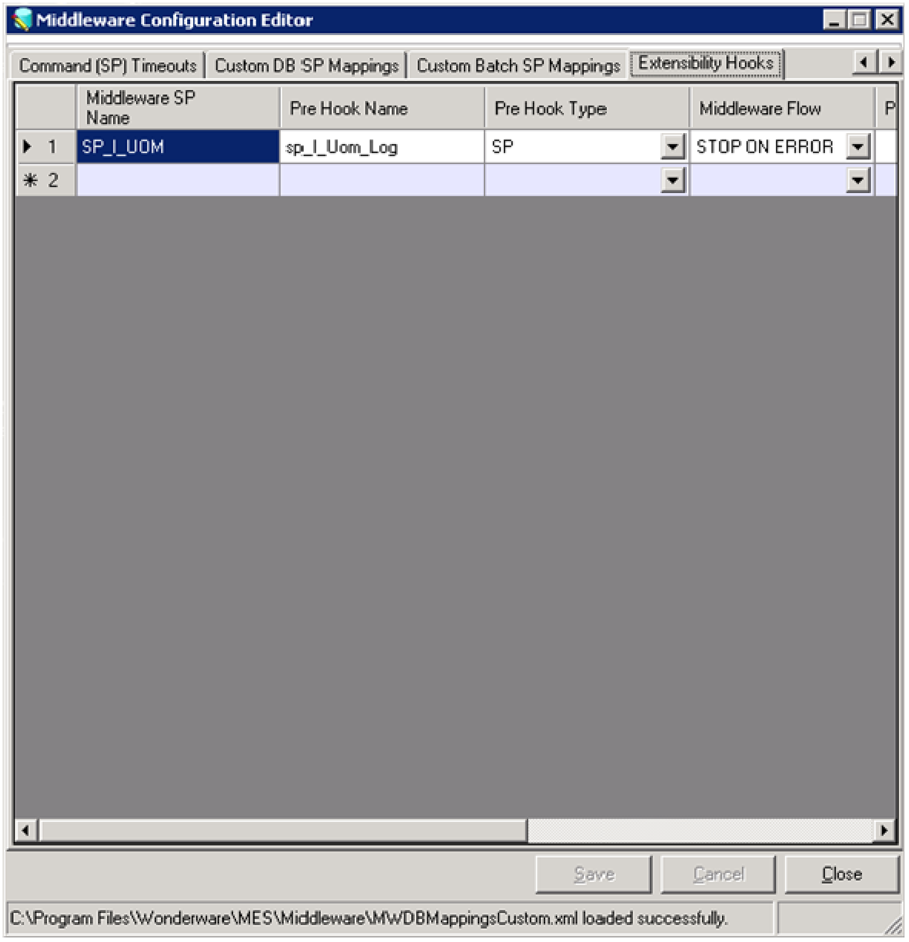
Opening the Middleware Configuration Editor

To open the Middleware Configuration Editor

In the Start menu, click:

All Programs > Wonderware > MES > Utilities > Middleware Configuration Editor

The Middleware Configuration Editor appears.



	Middleware SP Name	Pre Hook Name	Pre Hook Type	Middleware Flow	P
▶ 1	SP_I_UOM	sp_I_Uom_Log	SP	STOP ON ERROR	
✳ 2					

C:\Program Files\Wonderware\MES\Middleware\MWDBMappingsCustom.xml loaded successfully.

The configuration information entered in the Configuration Editor is stored in the file MWDBMappingsCustom.xml file specified in "Installation Location" above.

Middleware SP Name

This column in the Middleware Configuration Editor represents a unique stored procedure name for which a post-hook or a pre-hook can be configured.

Note: No stored procedure name can be used more than once.

The stored procedure name is the name of a valid stored procedure that exists in the MES database to which the MES middleware connects. The end user is responsible for finding the correct stored procedure name by manually looking at the stored procedure names in the MES database. If the middleware connects to an Oracle database, the stored procedure name must be entered in all capital letters.

Whenever this stored procedure (configured in this column) is invoked, by virtue of calling an equivalent middleware method from the client, the middleware determines whether there are any pre- or post-hooks configured for this stored procedure and execute them according to the Extensibility Hook configuration.

The hooks configured for a stored procedure can be removed or deleted by highlighting the row in the grid.

Pre-Hook/Post-Hook Name

The name of the Extensibility Hook is configured in this column.

Note: If the name is left blank (or empty), the middleware assumes that the Extensibility Hook is not configured.

Stored Procedure

A valid stored procedure name from the MES database, if the type of Extensibility Hook is stored procedure.

The parameter names, data types, and presence of default values in the custom (hook) stored procedure must match exactly those of the parameter names, data types, and presence of default values in the middleware stored procedure (stored procedures configured in the Middleware SP Name column). However, there can be additional parameters in the custom (hook) stored procedure that might not be matching the parameters in the middleware stored procedure, but those additional parameters must be defaulted to NULL in the custom stored procedure. If they are not defaulted to NULL, an exception might be raised by the database layer for not supplying enough values for those additional parameters. It is recommended that the list of stored procedure parameters, data types, and default values (if any) be copied from the middleware stored procedure to the custom stored procedure to ensure all these properties are identical between them. However, the default values for the custom (hook) stored procedure do not have to exactly match the default values in the middleware stored procedure.

The values supplied for the middleware stored procedure (configured in the Middleware SP Name column) are the same values supplied to the custom (hook) stored procedure.

Assembly

The hook method in the assembly has a single ‘string’ parameter that accepts the XML message from the middleware event. The XML message is the exact XML generated for the middleware event to process. The end user is responsible for parsing the XML message and extracting the relevant data from the XML message. See the Stateless API help for more details about the XML structure. The middleware calls impersonate the ArchestrA Network user; therefore, the custom (hook) assembly is invoked as that user.

The assembly name can be configured in two ways:

- If the assembly is added to the GAC, then the detailed information about the assembly version, public key token, etc. must be specified.
- However, if the assembly is not added to the GAC, then the physical location of the assembly must be specified.

In both of these cases, the class name and the method name must be specified by separating them by semi-colon (;).

Note: The configured Extensibility Hook method can accept only one string parameter.

Assembly Is Not Added to the GAC

If the assembly is not currently added to the GAC, the user is required to specify the physical location of the assembly itself to load the assembly from the specified path.

For example,

> C:\Temp\TestHooks\TestHooks\bin\Debug\TestHooks.dll;TestHooksNamespace.TestHooksClass;TestHooksMethod

represents the assembly name as “TestHooks.dll”, located at the path **C:\Temp\TestHooks\TestHooks\bin\Debug\**, and that has the namespace TestHooksNamespace, class name TestHooksClass, and the method name TestHooksMethod.

Assembly Is Added to the GAC

If the assembly is currently added to the GAC, the user is required to specify the public key token, assembly version, and so on, to load the right assembly from the GAC.

Note: The assembly must be signed with a strong name.

For example,

> TestHooks,version=1.0.0.0,culture=neutral,processorarchitecture=MSIL,PublicKeyToken=23106a86e706d0ae;TestHooksNamespace.TestHooksClass;TestHooksMethod

represents the assembly name as TestHooks.dll, version as 1.0.0.0, public key token as 23106a86e706d0ae, and that has the namespace TestHooksNamespace, class name TestHooksClass, and the method name TestHooksMethod.

Pre-Hook and Post-Hook Type

A single middleware event can either execute a stored procedure or invoke an assembly for a pre-hook and a post-hook. No other types besides the following are allowed in the current release.

- **Stored procedure.** Indicates that the configured hook type is a stored procedure. In other words, the name configured in the corresponding Pre-Hook Name or Post-Hook Name column is considered to be a stored procedure. Therefore, the configured custom stored procedure in the MES database is invoked before or after the middleware call depending upon the pre-hook or post-hook configuration.
- **Assembly.** Indicates the configured hook type is an assembly. In other words, the name configured in the corresponding Pre-Hook Name or Post-Hook Name column is considered to be an assembly. Therefore, the configured custom assembly is invoked before or after the middleware call depending upon the pre-hook or post-hook configuration.

Middleware Flow (Pre-Hook/Post-Hook)

This column in the middleware Configuration Editor represents the type of flow that is desired while processing the middleware event. The following sub-sections explain the different types of middleware flow supported by the MES middleware.

Stop

This option indicates that the flow of the middleware execution will stop after completing the current execution, regardless of whether the current execution succeeds or not. In other words, when the pre- or the post-hook is configured with this option, the execution stops with the current call and returns the result to the caller, regardless of whether the current transaction was successful or not (i.e., committed or rolled back). If an error occurs, the error message is returned to the caller.

The table below describes the flow of the middleware call after the pre-hook or post-hook execution is completed with the Middleware Flow parameter set to Stop.

Hook Type	Hook Success	Hook Error
Pre-Hook	Stop. The pre-hook call is completed successfully. **Note:** The data modified on tables for the pre-hook call is committed. The normal middleware call will never be executed.	Stop. The pre-hook call resulted in an error. The error message is returned to the end user or the error message is logged in the logger. **Note:** The data modified on table for the pre-hook call is rolled back. The normal middleware call will never be executed.
Post-Hook	Not applicable; this option cannot be configured for post-hooks.	Not applicable; this option cannot be configured for post-hooks.

Stop on Error

This option indicates that the flow of the middleware execution will stop only if it encounters an error while processing the current execution. As long as there is no error while executing the pre-hook, post-hook, or the middleware event, the execution continues.
The table below describes the flow of the middleware call after the pre-hook or post-hook execution is completed with the Middleware Flow parameter set to Stop on Error.

Hook Type	Middleware Execution Status	Hook Success	Hook Error
Pre-Hook	Not applicable.	The pre-hook call completed successfully. **Note:** The data modified on tables for the pre-hook call is committed. The middleware continues to execute the normal middleware call.	Stop. The pre-hook call resulted in an error. The error message is returned to the caller. **Note:** The data modified on tables for the pre-hook call is rolled back. The normal middleware call will never be executed.

Hook Type	Middleware Execution Status	Hook Success	Hook Error
Post-Hook	The normal middleware call/transaction has succeeded.	The data modified on tables for the normal middleware call and for the post-hook are committed.	The data modified for the normal middleware call and for the post-hook are both rolled back. The error message is returned to the caller.
Post-Hook	Stop. The normal middleware call/transaction has failed or rolled back. The error message is returned to the caller.	Not applicable. **Note:** The post-hook will never be executed, because the normal middleware call/event/transaction has failed or rolled back.	Not applicable. **Note:** The post-hook will never be executed, because the normal middleware call/event/transaction has failed or rolled back.

Continue Even on Error

This option indicates that the flow of the middleware execution will continue even when an error is encountered in any part of the execution sequence.

The table below describes the flow of the middleware call after the pre- or post-hook execution is completed with the Middleware Flow parameter set to Continue Even on Error.

Hook Type	Middleware Execution Status	Hook Success	Hook Error
Pre-Hook	Not applicable.	The pre-hook call is completed successfully. **Note:** The data modified on tables for the pre-hook call is committed. The middleware continues to execute the normal middleware call.	Continue even on error. The pre-hook call resulted in an error. The generated error message is ignored by the middleware. **Note:** The data modified on tables for the pre-hook call is rolled back. The middleware continues to execute the normal middleware call.
Post-Hook	The normal middleware call/transaction has succeeded.	The database transactions started for the normal middleware call and for the post-hook call are committed.	The data modified on tables for the normal middleware call is committed, but the data modified on tables for the post-hook is rolled back. The error message generated by the post-hook transaction is ignored by the middleware.
Post-Hook	The normal middleware call/transaction has failed or rolled back.	Stop. The error message is returned to the caller. **Note:** The post-hook will never be executed, because the normal middleware call/event/transaction has failed or rolled back.	Not applicable. **Note:** The post-hook will never be executed, because the normal middleware call/event/transaction has failed or rolled back.

Loading the Configuration and Hooks

After the Extensibility Hooks are configured and saved in the Middleware Connection Editor, the MES Host/MES Middleware must be restarted to reload all the configuration details and the Extensibility Hooks into its memory.

Examples of Middleware Hook Configurations and Executions

The examples in the following sections illustrate how the Extensibility Hooks can be configured and used using different options provided in the middleware Configuration Editor.

Custom Stored Procedure Hook

This example illustrates how a custom stored procedure can be configured as a hook that can be invoked when executing a standard stored procedure in the MES middleware.
The following example configuration is added to the middleware Configuration Editor.

Configuration Parameter	Example Entry	Description
Middleware SP Name	sp_I_UOM	The actual stored procedure name, supplied out-of-the box, from the MES database.
Pre-Hook Name	sp_I_Uom_Log	The custom stored procedure created by the end user in the MES database after installing the MES database.
Pre-Hook Type	SP	Indicates that the type of this hook is a stored procedure
Middleware Flow	STOP ON ERROR	Indicates to the middleware to stop the execution if there was an error while executing the sp_I_Uom_Log stored procedure.

Content of the Custom Stored Procedure sp_I_Uom_Log

```
CREATE PROCEDURE sp_I_Uom_Log
(@description nvarchar(40)
,@abbreviation nvarchar(20)
,@last_edit_comment nvarchar(254)
,@my_arg nvarchar(40) = null
,@last_edit_at datetime OUT
,@uom_id int OUT
) AS
BEGIN
    SET NOCOUNT ON
    DECLARE @logged_at DATETIME

        --Call other SPs if needed.

    SET @logged_at = GETDATE()
    INSERT INTO error_log (logged_at, severity, instance_info
        , object_name, object_section, description,
        system_msg)
    VALUES (@logged_at, 0, @abbreviation
        , @description, @description, @description,
        @description)

    SET @uom_id = SCOPE_IDENTITY()

        --Call other SPs if needed.

END
```

As can be seen from its content, the stored procedure sp_I_Uom_Log contains all the parameters and their properties that sp_I_Uom has, and an additional parameter containing more contextual information. The custom stored procedure can completely ignore the input parameters and the code inside the “begin and end” block can be customized for individual needs. The statements in red indicate that one or more custom stored procedures can be called from this custom stored procedure. The concept described above is also applicable to configuring the post-hook of type stored procedure.

Action

When a new unit of measurement (UOM) is added from the MES Client application or by using an ArchestrA script to add the UOM, the middleware typically uses the sp_I_Uom stored procedure to add the new unit of measurement. As explained in "Functional Overview" on page 11, the middleware first verifies whether any pre-hook is configured for the sp_I_Uom stored procedure. Since there is a pre-hook sp_I_Uom_Log configured for this case, the middleware first executes the stored procedure sp_I_Uom_Log and then evaluates the result before processing the standard middleware call sp_I_Uom to create a new unit of measurement in the MES database.

Custom Stored Procedure Hook Raises an Error (Middleware Flow: Stop)

This example illustrates how a custom stored procedure can be configured as a hook that can be invoked when executing a standard stored procedure in the MES middleware.
The following example configuration is added to the Middleware Configuration Editor.

Middleware SP Name	Pre-Hook Name	Pre-Hook Type	Middleware Flow
sp_I_UOM	sp_I_Uom_Log	SP	STOP

Content of the Custom Stored Procedure sp_I_Uom_Log

```
CREATE PROCEDURE sp_I_Uom_Log
(@description nvarchar(40)
,@abbreviation nvarchar(20)
,@last_edit_comment nvarchar(254) = NULL
,@my_arg nvarchar(40) = null
,@last_edit_at datetime OUT
,@uom_id int OUT
) AS
BEGIN
    SET NOCOUNT ON
    DECLARE @logged_at DATETIME

    --Call other SPs if needed.

    SET @logged_at = GETDATE()
    INSERT INTO error_log (logged_at, severity, instance_info
    , object_name, object_section, description, system_msg)
    VALUES (@logged_at, 0, @abbreviation
    , @description, @description, @description, @description)

    SET @uom_id = SCOPE_IDENTITY()

    --Call other SPs if needed.

    raiserror('Error message from the custom hook stored
    procedure - sp_I_Uom_Log', 16, 1)
END
GO
```

Action

In this case, when a new UOM is added from the MES Client application or by using an ArchestrA script to add the UOM, the middleware executes the stored procedure sp_I_Uom_Log, and evaluates the result before actually processing the standard middleware call sp_I_Uom to create a new unit of measurement in the MES database. Since the Middleware Flow parameter is set to Stop after executing the custom (pre-hook) stored procedure, the sequence of execution stops and returns the following error to the client:

> Error message from the custom hook stored procedure - sp_I_Uom_Log

Note that even if there are no errors in the custom hook stored procedure, the normal middleware call will never be processed in this case, because the the Middleware Flow parameter is set to Stop after executing the pre-hook call. That is, in this case, the middleware never adds the unit of measurement.

Custom Assembly That Is Not in the GAC

This example illustrates how a custom assembly that is not in the GAC can be configured as a hook and invoked when executing a standard stored procedure in the MES middleware.
The following example configuration is added to the Middleware Configuration Editor.

Middleware SP Name	**Pre-Hook Name**	**Pre-Hook Type**	**Middleware Flow**
sp_I_UOM	C:\Temp\TestHooks\TestHooks\bin\Debug\TestHooks.dll;TestHooksNamespace.TestHooksClass;TestHooksMethod	ASSEMBLY	STOP ON ERROR

The configuration in the pre-hook name can be interpreted as below:

> Assembly Name/DLL Name: TestHooks.dll
>
> Namespace: TestHooksNamespace
>
> Class Name: TestHooksClass
>
> Method Name: TestHooksMethod

Content of the Code in the TestHooksClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TestHooksNamespace
{
    public class TestHooksClass
    {
        public TestHooksClass()
        { }
        public void TestHooksMethod(string xmlSource)
        {
            // Compose a string that consists of three lines.
            string lines = string.Format("DateTime: {0}, XMLSource: {1}",
                DateTime.Now.ToString(), xmlSource);
            // Write the string to a file.
            using (System.IO.StreamWriter file = new
                System.IO.StreamWriter("c:\\Temp\\UomLog.txt", true))
            {
                file.WriteLine(lines);
            }
        }
    }
}
```

The value for the xmlSource parameter contains the XML message for the normal middleware event. The structure of the XML message used for this case is listed below:

```
<?xml version='1.0'?>
<request>
    <object>uom</object>
    <msgtype>exec</msgtype>
    <cmd>add</cmd>
    <uom>
        <session_id>119</session_id>
        <description>Unit_001</description>
        <abbreviation>ssss</abbreviation>
        <last_edit_comment></last_edit_comment>
    </uom>
    <validate>1</validate>
</request>
```

The text output is written to the text file UomLog.txt in the format below:

```
DateTime: 1/2/2013 11:34:44 AM, XMLSource:
<?xmlversion='1.0'?><request><object>uom</object>
<msgtype>exec</msgtype><cmd>add</cmd><uom><session_id>119
</session_id><description>Unit_001</description>
<abbreviation>ssss</abbreviation><last_edit_comment>
</last_edit_comment></uom><validate>1</validate></request>
```

The concept described above is also applicable to configuring the post-hook of type assembly.

Action

When a new UOM is added from the MES Client application or by using an ArchestrA script to add the UOM, the middleware typically uses the sp_I_Uom stored procedure to add the new unit of measurement. As explained in "Functional Overview" on page 11, the middleware first verifies whether any pre-hook is configured for the sp_I_Uom stored procedure. Since there is a pre-hook TestHooks.dll configured for this case, the middleware first executes the method TestMethod in the TestHooks.dll assembly and then evaluates the result before actually processing the standard middleware call sp_I_Uom to create a new unit of measurement in the MES database.

Custom Assembly Is Added to the GAC

This example illustrates how a custom assembly that has been added to the GAC can be configured as a hook and invoked when executing a standard stored procedure in the MES middleware. The expectation is that the assembly is signed with a strong-name.
The following example configuration is added to the Middleware Configuration Editor.

Middleware SP Name	Pre-Hook Name	Pre-Hook Type	Middleware Flow
sp_I_UOM	TestHooks,version=1.0.0.0,culture=neutral, processorarchitecture= MSIL,PublicKeyToken=23106a86e706d0ae; TestHooksNamespace.TestHooksClass; TestHooksMethod	ASSEMBLY	STOP ON ERROR

The configuration in the pre-hook name can be interpreted as below:

Assembly Name/DLL Name: TestHooks.dll

Namespace: TestHooksNamespace

Class Name: TestHooksClass

Method Name: TestHooksMethod

Content of the Code in the TestHooksClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TestHooksNamespace
{
   public class TestHooksClass
   {
      public TestHooksClass()
```

```
        { }
        public void TestHooksMethod(string xmlSource)
        {
            // Compose a string that consists of three lines.
            string lines = string.Format("DateTime: {0}, XMLSource: {1}",
                DateTime.Now.ToString(), xmlSource);
            // Write the string to a file.
            using (System.IO.StreamWriter file = new
                System.IO.StreamWriter("c:\\Temp\\UomLog.txt", true))
            {
                file.WriteLine(lines);
            }
        }
    }
}
```

The concept described above is also applicable to the post-hook of type assembly.

Action

When a new UOM is added from the MES Client application or by using an ArchestrA script to add the UOM, the middleware typically uses the sp_I_Uom stored procedure to add the new unit of measurement. As explained in "Functional Overview" on page 11, the middleware first verifies whether any pre-hook is configured for the sp_I_Uom stored procedure. Since there is a pre-hook TestHooks.dll configured for this case, the middleware first executes the method TestMethod in the TestHooks.dll assembly and then evaluates the result before actually processing the standard middleware call sp_I_Uom to create a new unit of measurement in the MES database.

Chapter 4

Configuring Extensibility Hooks on Multiple Middleware Servers

The Middleware Extensibility Hooks are configured in the Extensibility Hooks tab in the Middleware Configuration Editor, and the hooks configurations are stored in the same folder in the file MWDBMappingsCustom.xml. The MES middleware server loads the Extensibility Hooks only if the hook parameters are configured in the MWDBMappingsCustom.xml file in its folder. When there is more than one middleware server configured to connect to the same MES database, then the end user is responsible for managing the hook configurations and keeping them in sync across all the middleware servers.

For example, let's say Middleware Server 1 has a pre-hook configured for the sp_I_Uom (Uom.Add) to perform a pre-action before the actual sp_I_Uom is executed. Middleware Server 2 has a pre-hook configured for the sp_U_Job_Exec_StartBatchJobs (Job_Exec.StartJob) to perform a pre-action before the actual sp_U_Job_Exec_StartBatchJobs is executed. These two Middleware Servers are configured to connect to the same MES database, and these two are the only two hooks configured in their respective middleware directories.

The table below explains how the pre-hook commands are processed by both the middleware servers using the example above:

	Middleware Server1		Middleware Server2	
Middleware SP Name	**Pre-Hook Loaded**	**Execute Pre-Hook**	**Pre-Hook Loaded**	**Execute Pre-Hook**
sp_I_UOM	Yes	Yes	No	No
sp_U_Job_Exec_StartBatchJobs	No	No	Yes	Yes

The Middleware Server 1 processes the pre-hook for sp_I_UOM before calling the sp_I_Uom stored procedure on the MES database, but does not process the pre-hook for sp_U_Job_Exec_StartBatchJobs before calling this stored procedure, because Middleware Server 1 does not know anything about the pre-hook for sp_U_Job_Exec_StartBatchJobs, since the XML file in its folder does not have any information about the hook for sp_U_Job_Exec_StartbatchJobs.

Similarly, the Middleware Server 2 processes the pre-hook for sp_U_Job_Exec_StartBatchJobs before calling the sp_U_Job_Exec_StartBatchJobs stored procedure on the MES database, but does not process the pre-hook for sp_I_UOM before calling this stored procedure.

Note: If the desire is to have an Extensibility Hook processed for a middleware stored procedure, regardless of the Middleware Server that invokes the stored procedure, then the user must copy the configuration file or manually configure the Extensibility Hook configurations into all the MES Middleware servers, thereby keeping the configuration information in sync across the Middleware servers.

Chapter 5

Frequently Asked Questions

What are hooks? What is the purpose of having hooks configured in the MES middleware?

A hook represents a custom functionality implemented by the end user in the form of a stored procedure, assembly, workflow, and so on. The purpose of having a hook is to allow a custom functionality to be integrated as a part of standard MES event and execute that functionality as a part of standard MES event.

How do you determine the name of a stored procedure that corresponds to a specific middleware event?

In most cases, the name of the MES middleware event matches the name of the MES stored procedure. If not, they are configured in the MWDBMappings.xml file under the db_objects section. This XML file resides in the same folder where the MES server is installed on the host machine. It is recommended to first look in this file to find the name of the stored procedure that corresponds to the name of the middleware event. If such a configuration is not found in this file, then the stored procedure name is resolved automatically on-the-fly by the middleware server. For example, a middleware event Item.Add corresponds to the stored procedure sp_I_Item, Attr.Update corresponds to sp_U_Attr, Ent.GetAll corresponds to sp_SA_Ent, JobExec.DownloadSpecs corresponds to sp_U_Job_Exec_DownloadSpecs, and so on. See the “Middleware Configuration Editor” chapter in the *Wonderware MES Installation Guide* for more details about how a middleware method is mapped to a stored procedure.

Can stored procedure names configured in the custom database mappings tab in the Middleware Configuration Editor be used as part of hooks? How are those stored procedures related to hooks?

Yes, the stored procedures (names) configured in the custom database mappings can be used to configure the stored procedure hooks. The custom database mappings is a separate mechanism that allows to map a custom stored procedure created by an end user, and that allows an end user to call that custom stored procedure from the client applications using the standard XML structure used by the middleware.

Does the middleware server need to be restarted if a hook configuration is modified?

Yes, whenever a hook configuration is modified, the middleware server needs to be restarted to reload the changes in its memory. Otherwise, the changes are not effective until the middleware server is restarted.

If there is more than one middleware serve, does the hook information need to be copied to more than one middleware server?

Yes. It is the responsibility of the end user to copy and synchronize the hooks configurations on all the middleware servers.

Do the pre-hook or the post-hook mechanism support the UpdateSpecific() method in the middleware to perform pre- or post-process?

No. The UpdateSpecific() method in the middleware uses dynamic SQL (a SQL Update statement generated when needed and not pre-compiled) to update the data in the MES database. Since there are no stored procedures involved with the UpdateSpecific() method, pre- and post-hooks cannot be configured and executed against this method.

Under what user name the custom (hook) assembly is executed?

The middleware calls impersonate the ArchestrA Network user; therefore, the custom (hook) assembly is invoked as that user.

Do the extensibility hooks (i.e., custom stored procedures and custom DLLs) automatically upgrade when the standard MES product is upgraded?

No. The end user is responsible for upgrading the custom stored procedures and custom DLLs to synchronize the methods and parameters in the custom hooks with the methods and parameters in the standard MES product.