AVEVA
Batch Management

COM Technical Reference Guide

July 2019

Publication date: Wednesday, July 17, 2019

Contact Information

AVEVA Group plc
High Cross
Madingley Road
Cambridge
CB3 0HB. UK

https://sw.aveva.com/

For information on how to contact sales, customer training, and technical support, see https://sw.aveva.com/contact.

Contents

CHAPTER 1

Introduction

This guide describes using the Batch Management COM technology and provides practical examples of its use.

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.

In This Chapter

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.

CHAPTER 2

Batch Management COM Technology

The Batch Management control system provides automation extensibility by way of Microsoft® Component Object Model (COM) technology. COM is a collection of services that allow software components to interoperate within a networked environment. The COM-based technologies supported by the batch control system consist of ActiveX controls, automation servers, and type libraries.

In This Chapter

ActiveX Controls

The Batch Management control system includes two ActiveX controls (InBatchSFC and OcxBatch) that you can use to develop custom applications for batch scheduling, monitoring, and control. You can use these controls within any application that supports ActiveX controls (such as Visual Basic.NET, C++, and C#). The ActiveX controls are available for use only after you have installed an Batch Management Runtime Client.

SFC ActiveX Control

The SFC ActiveX control enables you to develop custom applications that provide visual representation (sequential flow charts) of batch processing.

For more information on the SFC ActiveX control, see *Chapter 3, SFC ActiveX Control*.

Batch Management ActiveX Control

The Batch Management ActiveX control enables you to develop custom applications for batch scheduling and batch management functions. The control provides the functionality of the Batch Scheduler and Batch Display applications.

For more information on the Batch Management ActiveX control, see Chapter 4, *Batch Management ActiveX Control* on page 43.

Automation Servers

The Batch Management control system includes two automation servers that provide access to the material and the recipe databases. Each server is comprised of a set of object classes that contain a variety of methods and properties. You can use these to develop custom applications within COM-based environments such as Visual Basic, Visual C++, and Visual C#.

Material Database Automation Server

The Material Database Automation Server (MaterialSrv.exe) provides read and write access to the materials database. You use the server to develop custom applications that provide the following functionality:

- Add, change, and delete materials (such as ingredients and intermediates)

- Define default characteristics for a material
- Query and assign available units to a material
- Add and remove lot tracking information for a material assigned to a unit
- Query material lot tracking information
- Define actual characteristic values for a specific lot of material
- Find the location of a material
- Query the contents of a unit
- Query the total quantity of a material

For more information on the Material Database Automation Server, see Chapter 5, "Material Database Automation Server."

Recipe Database Automation Server

The Recipe Database Automation Server (RecipeEdit.exe) provides read and write access to the recipe database. You use the server to develop custom applications that provide the following functionality:

- Add, change, and delete recipes
- Query and change recipe header information
- Query and change recipe equipment requirements
- Query formula inputs defined for a recipe
- Query formula outputs defined for a recipe
- Define and modify the formula for a recipe
- Define a recipe procedure

For more information on the Recipe Database Automation Server, see Chapter 5, "Recipe Database Automation Server."

Batch Function Interface Type Libraries

The Batch Management control system includes two type libraries that define an interface from which users can create an in-process server (.dll) to interact with the batch function interface. Each type library is comprised of a set of object classes that contain a variety of methods and properties. You can use these to develop custom servers with COM-based environments such as Visual Basic.NET, C++, and C#.

For more information on the batch function interface type libraries, see Chapter 6, "Batch Function Interface Type Libraries."

Batch Hooks Type Library

The Batch Hooks Type Library (batchvbserver.dll) includes functions and subroutines that can be used to access the batch function interface. The batch function interface consists of several hooks into the processing of Batch Manager. By adding logic to these hooks, you can extend the capabilities of Batch Manager.

The batch function interface consists of the following hooks:

- Batch Initialization – When a batch is initialized from the Batch Scheduler
- Batch Prepare – At the start of a batch
- Batch Complete – At the completion of a batch

- Unit Procedure Prepare – At the start of a unit procedure
- Unit Procedure Complete – At the end of a unit procedure
- Operation Prepare – At the start of an operation
- Operation Complete – At the completion of an operation
- Phase Prepare – At the start of a phase
- Phase Complete – At the completion of a phase
- Evaluate Unit for Allocation – When unit allocation is required
- Evaluate Connection for Allocation – When connection allocation is required
- Allocation Changes – When a unit or connection is allocated or released
- Log Equipment Status Changes – When the status of a unit or segment changes

Batch Objects Type Library

The Batch Hook Type Library (batchobjsrv.dll) provides objects that contain the appropriate batch, phase, parameter, and equipment data available and modifiable within the hooks.

The type library provides you with access to the following objects:

- Batch Object – All batch schedule data.
- Phase Object – All phase configuration data
- Parameter Object – All phase formula parameter data.
- Equipment Object – All batch equipment data
- Equipment Status Object – All equipment status data

CHAPTER 3

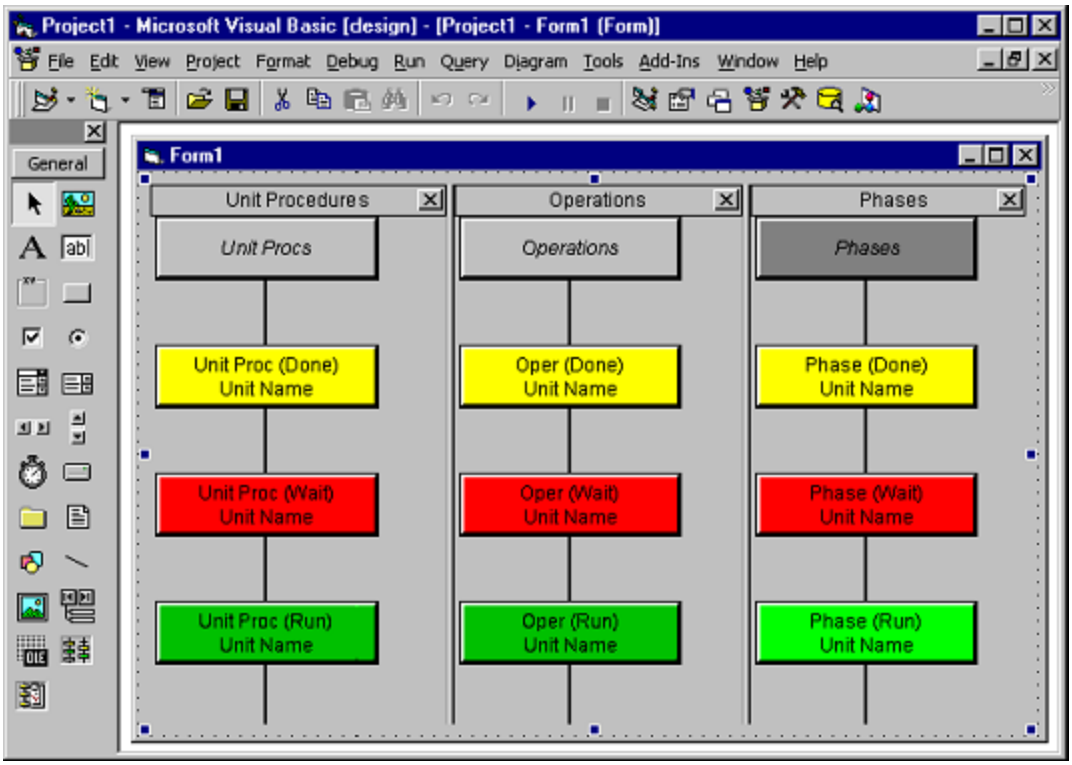
SFC ActiveX Control

The SFC ActiveX control (InBatchSFC.ocx) provides a visual representation of a running batch. You can use this control within any application that supports ActiveX controls (such as Visual Basic.NET, C++, and C#).

To access to the SFC ActiveX control, you must install the appropriate Batch Client software and enable any appropriate licensing.

Refer to the appropriate COM-based environment documentation for specific details on installing ActiveX controls within that environment.

The following figure shows the SFC ActiveX control graphics display.

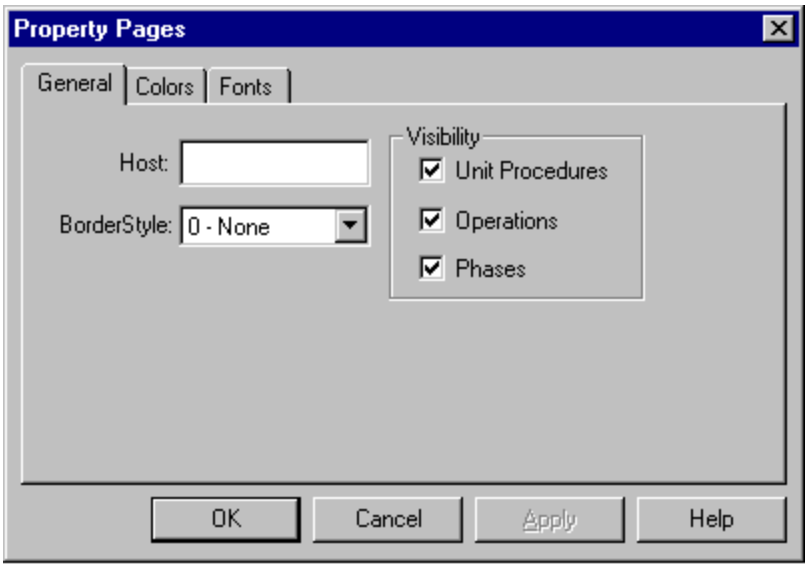


In This Chapter

Properties

Each SFC object has a Control Name and other properties available for configuration.

The following figure shows the SFC Property Pages dialog box.



The following properties are available for the SFC ActiveX control.

ActiveUnitGridColumnHeaders

This property is a comma-separated string that specifies the label displayed in each column of the Active UnitProc Parameter Grid. Each position in the comma-delimited string maps to a position in the base set of column elements, even if the columns are re-ordered.

Data Type: String

Access: RW

Base Element Positions

Parameter, Parameter Type, Target Value, Actual Value, Material ID, Material Name, UOM, High Deviation, Low Deviation, High Limit, Low Limit, Lot Code, and Preact.

Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default label to be used.

Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live labels in the Active grid.

ActiveUnitGridColumnPositions

This property is a comma-separated string that specifies the column positions at which column elements appear in the Active UnitProc Parameter Grid.

Each position in the comma-delimited string maps to a position in the base set of column elements. This property changes the position from the base.

Data Type: String

Access: RW

Base Element Positions

Parameter, Parameter Type, Target Value, Actual Value, Material ID, Material Name, UOM, High Deviation, Low Deviation, High Limit, Low Limit, Lot Code, and Preact.

Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default label to be used.

The results are undefined if you try to specify the same position for multiple elements.

Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live positions in the Active grid.

ActiveUnitGridColumnWidths

This property is a comma-separated string that specifies the column width (in pixels) that is used in each column of the Active UnitProc Parameter Grid.

Each position in the comma delimited string maps to a position in the base set of column elements—even if the columns are re-ordered.

Data Type: String

Access: RW
Base Element Positions

Parameter, Parameter Type, Target Value, Actual Value, Material ID, Material Name, UOM, High Deviation, Low Deviation, High Limit, Low Limit, Lot Code, and Preact.

A value of 0 for an entry specifies that the column is hidden.
Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default width to be used.
Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live widths in the Active grid.

ActiveUnitGridSortedAscending

This property sets or gets the current sorted direction of the Active UnitProc Parameter Grid. This value is valid only if one of the columns of the grid is sorted.

0 = descending sort order

1 = ascending sort order

Data Type: Boolean

Access: RW
Writing to the Property

Use at design time and run time.
Reading from the Property

At design time the property returns any value you may have set or the default.

At run time the property shows the current live sort direction of the Active grid.

ActiveUnitGridSortedColumn

This property sets or gets the current sorted column position in the Active UnitProc Parameter Grid. The position represents a position in the base set of column elements, even if the columns are re-ordered.

Data Type: Short

Access: RW

Base Element Positions

Parameter, Parameter Type, Target Value, Actual Value, Material ID, Material Name, UOM, High Deviation, Low Deviation, High Limit, Low Limit, Lot Code, and Preact.

A value of –1 signifies no sorting.

Writing to the Property

Use at design time and run time.

Reading from the Property

At design time, the property returns any value you may have set or the default.

At run time, the property shows the current live positions in the Active grid.

BackColor

This property sets the fill color of the background of the control.

Data Type: Color

Access: RW

BorderStyle

This property determines the type of border that is drawn around the control:

0 = None

1 = Fixed Single

Data Type: Short

Access: RW

CloseButtons

This property enables or disables the buttons to close the Unit Procedure, Operation, and Phase panes:

0 = Buttons disabled

1 = Buttons are available for selection

Default = Buttons disabled

Data Type: Short

Access: RW

ForeColor

This property sets the foreground color used for all lines in the control.

Data Type: Color

Access: RW

Host

This property sets the name of the computer running Batch Manager. You must set the host before you can call Init method.

Data Type: String

Access: RW

InActiveUnitGridColumnHeaders

This property is a comma-separated string that specifies the label displayed in each column of the InActive UnitProc Parameter Grid.

Each position in the comma-delimited string maps to a position in the base set of column elements, even if the columns are re-ordered.

Data Type: String

Access: RW
Base Element Positions

Parameter, Parameter Type, Value, Material ID, Material Name, and UOM.
Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default label to be used.
Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live labels in the **InActive** grid.

InActiveUnitGridColumnPositions

This property is a comma-separated string that specifies the column positions at which column elements display in the Active UnitProc Parameter Grid.

Each position in the comma-delimited string maps to a position in the base set of column elements—this property changes the position from the base.

Data Type: String

Access: RW
Base Element Positions

Parameter, Parameter Type, Value, Material ID, Material Name, and UOM.
Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default position to be used.

The results are undefined if you try to specify the same position for multiple elements.
Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live positions in the **Active** grid.

InActiveUnitGridColumnWidths

This property is a comma-separated string that specifies the column width (in pixels) that is used in each column of the InActive UnitProc Parameter Grid.

Each position in the comma-delimited string maps to a position in the base set of column elements, even if the columns are re-ordered.

Data Type: String

Access: RW
Base Element Positions

Parameter, Parameter Type, Value, Material ID, Material Name, and UOM

A value of 0 for an entry specifies that the column is hidden.
Writing to the Property

Use at design time and run time.

A blank or missing entry in the delimited string causes the default width to be used.

The results are undefined if you try to specify the same position for multiple elements.
Reading from the Property

At design time the property returns any value you may have set or blank if not set.

At run time the property shows the current live widths in the **Active** grid.

InActiveUnitGridSortedAscending

This property sets or gets the current sorted direction of the InActive UnitProc Parameter Grid.

This value is valid only if one of the columns of the grid is sorted.

0 = descending sort order

1 = ascending sort order

Data Type: Boolean

Access: RW
Writing to the Property

Use at design time and run time.
Reading from the Property

At design time the property returns any value you may have set or the default.

At run time the property shows the current live sort direction of the In**Active** grid.

InActiveUnitGridSortedColumn

This property sets or gets the current sorted column position in the InActive UnitProc Parameter Grid. The position represents a position in the base set of column elements, even if the columns are re-ordered.

Data Type: Short

Access: RW
Base Element Positions

Parameter, Parameter Type, Value, Material ID, Material Name, UOM

A value of –1 signifies no sorting.
Writing to the Property

Use at design time and run time.
Reading from the Property

At design time the property returns any value you may have set or the default.

At run time the property shows the current live positions in the In**Active** grid.

OperationsVisible

This property sets a flag to show or hide the **Operations** pane of the SFC:

0 = Hide Operations pane

1 = Show Operations pane

Data Type: Short

Access: RW

OperBackColor

This property sets the background color for operation objects in the Ready or Interlocked state.

Data Type: Color

Access: RW

OperContinueBackColor

This property sets the background color for operation objects in which one or more phases are running with the continue mode property enabled.

Data Type: Color

Access: RW

OperContinueTextColor

This property sets the text color for operation objects in which one or more phases are running with the continue mode property enabled.

Data Type: Color

Access: RW

OperDoneBackColor

This property sets the background color for operation objects in the Done or Aborted state.

Data Type: Color

Access: RW

OperDoneTextColor

This property sets the text color for operation objects in the Done or Aborted state.

Data Type: Color

Access: RW

OperFont

This property sets the font style for the text on operations.

Data Type: Font

Access: RW

OperRunBackColor

This property sets the background color for operation objects in the Run state.

Data Type: Color

Access: RW

OperRunTextColor

This property sets the text color for operation objects in the Run state.

Data Type: Color

Access: RW

OperTextColor

This property sets the text color for operation objects in the Ready or Interlocked state.

Data Type: Color

Access: RW

OperWaitBackColor

This property sets the background color for operation objects in the Held or Wait state.

Data Type: Color

Access: RW

OperWaitTextColor

This property sets the text color for operation objects in the Held or Wait state.

Data Type: Color

Access: RW

PhaseBackColor

This property sets the background color for phase objects in the Ready or Interlocked state.

Data Type: Color

Access: RW

PhaseDoneBackColor

This property sets the background color for phase objects in the Done or Aborted state.

Data Type: Color

Access: RW

PhaseDoneTextColor

This property sets the text color for phase objects in the Done or Aborted state.

Data Type: Color

Access: RW

PhaseFont

This property sets the font style for the text on phases.

Data Type: Font

Access: RW

PhaseRunBackColor

This property sets the background color for phase objects in the Run state.

Data Type: Color

Access: RW

PhaseRunTextColor

This property sets the text color for phase objects in the Run state.

Data Type: Color

Access: RW

PhasesVisible

This property sets a flag to show or hide the **Phases** pane of the SFC:

0 = Hide Phases pane

1 = Show Phases pane

Data Type: Short

Access: RW

PhaseTextColor

This property sets the text color for phase objects in the Ready or Interlocked state.

Data Type: Color

Access: RW

PhaseWaitBackColor

This property sets the background color for phase objects in the Held or Wait state.

Data Type: Color

Access: RW

PhaseWaitTextColor

This property sets the text color for phase objects in the Held or Wait state.

Data Type: Color

Access: RW

ProcBackColor

This property sets the background color for the procedure cell.

Data Type: Color

Access: RW

ProcFont

This property sets the font style for the text on the procedure cell.

Data Type: Font

Access: RW

ProcTextColor

This property sets the color for the text on the procedure cell.

Data Type: Color

Access: RW

UnitProcBackColor

This property sets the background color for operation objects in the Ready state.

Data Type: Color

Access: RW

UnitProcContinueBackColor

This property sets the background color for unit procedure objects in which one or more phases are running with the continue mode property enabled.

Data Type: Color

Access: RW

UnitProcContinueTextColor

This property sets the text color for unit procedure objects in which one or more phases are running with the continue mode property enabled.

Data Type: Color

Access: RW

UnitProcDoneBackColor

This property sets the background color for unit procedure objects in the Done or Aborted state.

Data Type: Color

Access: RW

UnitProcDoneTextColor

This property sets the text color for unit procedure objects in the Done or Aborted state.

Data Type: Color

Access: RW

UnitProceduresVisible

This property sets a flag to show or hide the **Unit Procedure**s pane of the SFC:

0 = Hide Unit Procedures pane

1 = Show Unit Procedures pane

Data Type: Short

Access: RW

UnitProcFont

This property sets the font style for the text on unit procedures.

Data Type: Font

Access: RW

UnitProcRunBackColor

This property sets the background color for unit procedure objects in the Run state.

Data Type: Color

Access: RW

UnitProcRunTextColor

This property sets the text color for unit procedure objects in the Run state.

Data Type: Color

Access: RW

UnitProcTextColor

This property sets the text color for unit procedure objects in the Ready or Interlocked state.

Data Type: Color

Access: RW

UnitProcWaitBackColor

This property sets the background color for unit procedure objects in the Held or Wait state.

Data Type: Color

Access: RW

UnitProcWaitTextColor

This property sets the text color for unit procedure objects in the Held or Wait state.

Data Type: Color

Access: RW

Methods

The following methods are available for the SFC ActiveX control.

EnableBranchSelection()

This method enables the selection of branches and transitions in the SFC control. This must be enabled if branches are to be selected when performing run-time procedure jumps.

Syntax

```
BatchSFCVar.EnableBranchSelection (bEnable)
```

Parameters

bEnable
Data Type: Boolean

Flag to signal branch selection state:
True = Branch selection enabled
False = Branch selection disabled

GetJumpTokenId()

This method queries the identification of the SFC token to which the user has selected to jump. A Long return value provides the jump identification value. This value can then be used in the BatchSetJumpToken method available in the Batch ActiveX Control to move procedural control to the new location.

Syntax

```
ReturnValue = BatchSFCVar.GetJumpTokenId ()
```

Init()

This method initializes communication with Batch Manager. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.Init ()
```

MiscSet2LevelLabels()

This method sets the phase header labels on the **Phases** pane of the SFC to match the unit procedure names or the operation names depending on whether three or two recipe levels have been defined. The MiscGet2Levels method available with the Batch ActiveX Control is used to determine the number of recipe levels defined for the system and is required to automatically set the parameter within this method. If this method is not used and only two recipe levels are defined, the name of the hidden operation is displayed on the top of the **Phases** pane. In addition, if only two recipe levels are defined, the OperationsVisible property should be disabled.

For more information on the Batch ActiveX control, see Chapter 4, *Batch Management ActiveX Control* on page 43.

Syntax

```
BatchSFCVar.MiscSet2LevelLabels (b2Levels)
```

Parameters

b2Levels
Data Type: Boolean
Flag to configure the control for two or three recipe levels:
True = Two recipe levels defined
False = Three recipe levels defined

OperationZoomIn()

This method increases the size of objects in the **Operations** pane.

Syntax

```
BatchSFCVar.OperationZoomIn ()
```

OperationZoomOut()

This method decreases the size of objects in the **Operations** pane.

Syntax

```
BatchSFCVar.OperationZoomOut ()
```

PhaseZoomIn()

This method increases the size of objects in the **Phases** pane.

Syntax

```
BatchSFCVar.PhaseZoomIn ()
```

PhaseZoomOut()

This method decreases the size of objects in the **Phases** pane.

Syntax

```
BatchSFCVar.PhaseZoomOut ()
```

Refresh()

This method forces the control to be redrawn.

Syntax

```
BatchSFCVar.Refresh ()
```

SetCLBFocus()

This method specifies the Campaign, Lot and Batch displayed by the SFC. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.SetCLBFocus (CLB)
```

Parameters

CLB
Data Type: String
Composed of the campaign ID, lot ID and batch ID separated by slashes. For example, the string C1/L1/B1 represents campaign C1, lot L1 and batch B1.

SetOperationFocus()

This method simulates a mouse click on an operation object. The row parameter represents the row of the operation to select. The column parameter represents the column of the operation to select. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.SetOperationFocus (Row, Column)
```

Parameters

Row
Data Type: Integer
Row of operation selected. All operation objects, transition objects, branches, and vertical lines that connect operations are included in the row count.

Column
Data Type: Integer
Column of operation selected. All operation objects and branches are included in the column count.

SetPhaseFocus()

This method simulates a mouse click on a phase object. The row parameter represents the row of the phase to select. The column parameter represents the column of the phase to select. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.SetPhaseFocus (Row, Column)
```

Parameters

Row
Data Type: Integer
Row of phase selected. All phase objects, transition objects, branches, and vertical lines that connect phases are included in the row count.

Column
Data Type: Integer
Column of phase selected. All phase objects and branches are included in the column count.

SetUnitProcedureFocus()

This method simulates a mouse click on a unit procedure object. The row parameter represents the row of the unit procedure to select. The column parameter represents the column of the unit procedure to select. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.SetUnitProcedureFocus (Row, Column)
```

Parameters

Row
Data Type: Integer
Row of unit procedure selected. All unit procedure objects, transition objects, branches, and vertical lines that connect unit procedures are included in the row count.

Column
Data Type: Integer
Column of unit procedure selected. All unit procedure objects and branches are included in the column count.

StartManualOperation()

Note: For advanced system debugging use only.

This method puts the SFC control into the manual operation debug mode. There is no way to reset the SFC to normal mode. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
ReturnCode = BatchSFCVar.StartManualOperation ()
```

Term()

This method terminates communication with Batch Manager. The return code is a short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 51*.

Syntax

```
returncode = batchsfcvar.term ()
```

UnitProcedureZoomIn()

This method increases the size of objects in the **Unit Procedures** pane.

Syntax

```
BatchSFCVar.UnitProcedureZoomIn ()
```

UnitProcedureZoomOut()

This method decreases the size of objects in the **Unit Procedures** pane.

Syntax

```
BatchSFCVar.UnitProcedureZoomOut ()
```

Events

The following events are available for the SFC ActiveX control.

BranchClicked

This event is called when a user selects a branch in the SFC control. To select branches in the SFC control, you must enable Branch selection using the EnableBranchSelection method.

Syntax

```
BatchSFCVar_BranchClicked (campaign, lot, batch, Row, Column, clicked)
```

Parameters

campaign
Data Type: String
Campaign Id of selected batch.

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

Row
Data Type: Short
Row of branch selected. All recipe procedure elements and vertical lines connecting recipe procedure elements are included in the row count.

Column
Data Type: Short
Column of branch selected. All recipe procedure elements and branches are included in the column count.

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

OperationClicked

This event is called when a user selects an object in the **Operations** pane.

Syntax

```
BatchSFCVar_OperationClicked (campaign, lot, batch, Row, Column, clicked)
```

Parameters

campaign
Data Type: String
Campaign Id of selected batch.

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

Row
Data Type: Integer
Row of operation selected. All operation objects, transition objects, branches, and vertical lines that connect operations are included in the row count.

Column
Data Type: Integer
Column of operation selected. All operation objects and branches are included in the column count.

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

PhaseClicked

This event is called when a user selects an object in the **Phases** pane.

Syntax

BatchSFCVar_PhaseClicked (campaign, lot, batch, label, status, clicked)

Parameters

campaign
Data Type: String
Campaign Id of selected batch.

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

label
Data Type: String
Label of selected batch

status
Data Type: Integer
Status of selected phase where:

0 = Open
1 = Ready
2 = Wait
3 = Run
4 = Done
5 = Interlocked
6 = Aborted
7 = Held
8 = Aborting

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

ProcedureClicked

This event is called when a user selects the procedure symbol in the SFC **Unit Procedures** pane. The procedure object is generally the first object in the unit procedure, operation, and phase sequences.

Syntax

BatchSFCVar_ProcedureClicked (campaign, lot, batch, Row, Column, clicked)

Parameters

campaign
Data Type: String
Campaign Id of selected batch

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

Row
Data Type: Short
Row of procedure symbol location

Column
Data Type: Short
Column of procedure symbol location

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

SystemShuttingDown

This event is called when Batch Manager is shutting down. All applications should handle this event by calling the Term function.

Syntax

```
BatchSFCVar_SystemShuttingDown ()
```

TransitionClicked

This event is called when a user clicks on a transition object.

Syntax

```
BatchSFCVar_PhaseClicked (campaign, lot, batch, label, status, clicked)
```

Parameters

campaign
Data Type: String
Campaign Id of selected batch

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

label
Data Type: String
Label of selected batch

status
Data Type: Integer
Status of selected phase where:
0 = Open
1 = Ready
2 = Wait
3 = Run
4 = Done
5 = Interlocked
6 = Aborted
7 = Held
8 = Aborting

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

UnitProcedureClicked

This event is called when a user selects a unit procedure in the **Unit Procedures** pane.

Syntax

```
BatchSFCVar_UnitProcedureClicked (campaign, lot, batch, Row, Column, clicked)
```

Parameters

campaign
Dta Type: String
Cmpaign Id of selected batch

lot
Data Type: String
Lot Id of selected batch

batch
Data Type: String
Batch Id of selected batch

Row
Data Type: Short

Row of branch selected. All unit procedure objects, transition objects, branches, and vertical lines that connect unit procedures are included in the row count.

Column
Data Type: Short
Column of unit procedure selected. All unit procedure objects and branches are included in the column count.

clicked
Data Type: Boolean
Click type where:
0 = Single-click
1 = Double-click

Error Return Values

The following error return values are available for the SFC ActiveX control.

Value	Description
100	The String CLB was not formatted correctly.
170	The OCX has not been successfully initialized.
171	The OCX could not allocate memory for initialization.
173	The connection to Environment Manager or Batch Manager failed.
174	The OCX is already initialized.
175	The OCX could not locate a valid batch license file.

Examples

The following sample code illustrates how to design a form with the SFC control. In this example, three recipe levels have been defined and the control has been configured, initialized and focused on a specific batch. If only two recipe levels are defined, the **OperationsVisible** property should be disabled and the **MiscSet2LevelLabels** method must be configured appropriately. The control is responsible for automatically displaying the operations and phases when a unit procedure is selected. This example assumes that the SFC ActiveX control has been properly added to the application.

```
private void Form_Load(object sender, EventArgs e)
{
      // Define required local variable
      string CLB = null;
      // Set SFC control properties
      BatchSFCVar.Host = "BATCHSERVER";
      BatchSFCVar.UnitProceduresVisible = 1;
      BatchSFCVar.OperationsVisible = 1; //zero for two recipe levels
      BatchSFCVar.PhasesVisible = 1;
      BatchSFCVar.MiscSet2LevelLabels (false); //true for two recipe levels
      BatchSFCVar.EnableBranchSelection (true);
      // Initialize communication wiht the batch server
      BatchSFCVar.Init();
      //Set focus to campaign "C1", lot "L1:, batch B1
      CLB = "C1/L1/B1";
```

```
      BatchSFCVar.SetCLBFocus (CLB);
}
```

The following example is slightly more complex and uses the Batch ActiveX control to automatically determine the number of recipe levels defined and sets the SFC control accordingly. This example assumes that both the SFC and the Batch ActiveX Controls have been properly added to the application.

For more information on the Batch ActiveX control, see Chapter 4, *Batch Management ActiveX Control* on page 43.

```
private void Form_Load(object sender, EventArgs e)
{
      // Define required local variables
      string CLB = null;
      int TwoRecipeLevels = 0;
      // Query number of recipe levels from the Batch ActiveX control
      TwoRecipeLevels = BatchOcxVar.MiscGet2Levels();
      // Set SFC control properties
      BatchSFCVar.Host = "BATCHSERVER";
      BatchSFCVar.UnitProceduresVisible = 1;
      if (TwoRecipeLevels == 1)
      {
            // Configure the SFC control for two recipe levels
            BatchSFCVar.OperationsVisible = 0;
            BatchSFCVar.MiscSet2LevelLabels (true);
      }
      else
      {
            // Configure the SFC control for three recipe levels
            BatchSFCVar.OperationsVisible = 1;
            BatchSFCVar.MiscSet2LevelLabels (false);
      }
      BatchSFCVar.PhasesVisible = 1;
      BatchSFCVar.EnableBranchSelection (true);
      // Initialize Communication with Batch Server
      BatchSFCVar.Init();
      // Set focus to campaign "C1", lot "L1", batch "B1"
      CLB = "C1/L1/B1";
      BatchSFCVar.SetCLBFocus (CLB);
}
```

CHAPTER 4

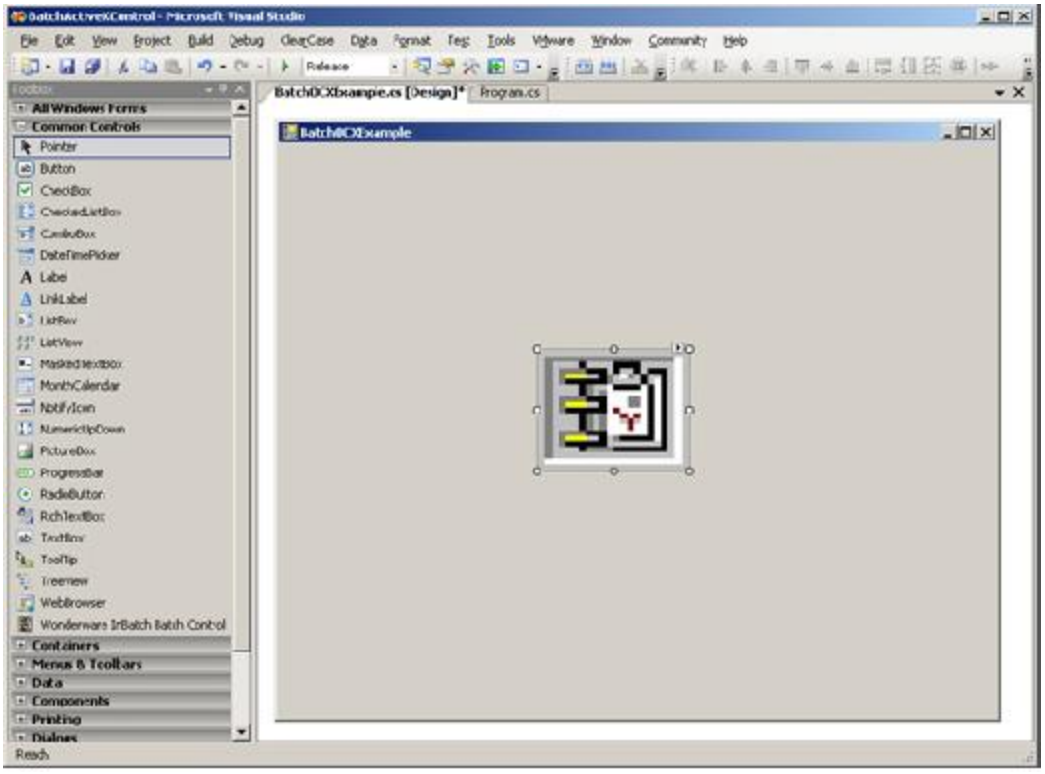
Batch Management ActiveX Control

The Batch Management ActiveX control (OcxBatch.ocx) provides an interface that you can use to schedule, monitor, and control batches. You can use this control within any application that supports ActiveX controls (such as Visual Basic.NET, C++, and C#).

NOTE: To access the Batch Management ActiveX control, you must install the appropriate batch run-time client software and enable any appropriate licensing.

For specific details on installing ActiveX controls within that environment, refer to the appropriate COM-based environment documentation.

The following figure shows an example Batch Management ActiveX dialog box.



Batch Management Control Functional Areas

OcxBatch breaks the Batch Management control operations down into various functional areas. The control method and event naming conventions reflect these functional areas. Most likely, there is interest in only a few functional areas of the Batch Management control, and therefore the application only needs to use and support a small subset of the methods and events that are provided.

The basic functional areas of OcxBatch are:

- Batch scheduling (Schedule)
- Batch processing (Batch)
- Phase processing (Phase)

- Answer questions (Question)
- Enter batch comment (Comment)
- Equipment allocation (Equipment)
- Allocation queue (AllocQueue)
- List batch messages (Messages)
- Edit phase (EditPhase)
- View transition (ViewTransition)
- Save recipe (SaveRecipe)
- Equipment selection (Select)
- List errors (Error)
- Miscellaneous (Misc)

In This Chapter

Working with Lists

OcxBatch is a non-GUI control that relies on an ActiveX container programming and scripting interface to make use of its capabilities. Most of the functional areas are based on presenting data in a list. You interact with these lists by displaying them in a GUI control or simply extracting data.

The lists typically have methods to perform the following tasks:

- Querying the number of items in the list
- Querying a preformatted list item based on a given index
- Querying individual fields of a list item based on a given index
- Setting the selected item in the list

The lists also typically have events for the following notifications:

- When the entire list has possibly become invalid.
- When to add an item
- When to change an item
- When to delete an item
- When to select an item
- When a list is busy being updated

You do not need all these methods and events when you work with a particular list. The necessary methods and events depend on the application requirements.

The most important aspect of working with the OcxBatch lists is that the control has complete command over them. The only authority you have is to tell the OcxBatch control to select a particular item in the list. Even then, the control fires an event back to the container to tell it to select the item in the list. Batch Manager automatically populates most of the lists, either automatically or after setting the appropriate focus.

Some of the lists are inter-related. That is, selecting an item in one (parent) list causes the views in other (child) lists to update. The container program responds to the events called by the lists of the control and instructs the control which item to select. Programs do not usually have to be concerned about any of the list inter-relationships.

The lists controlled by OcxBatch are:

- **Batch Scheduling Schedule List (ScheduleSched)**
  This list contains all batches in Batch Scheduler. This list is populated automatically.
- **Batch Scheduling Recipe List (ScheduleRecipe)**
  This list contains all recipes approved for production or approved for test in the Batch Scheduler. This list is populated when the ScheduleUpdateRecipeList method is run.
- **Batch Scheduling Formula List (ScheduleFormula)**
  This list contains all available formulas in the Batch Scheduler. This list is populated when **ScheduleUpdateFormulaList** method is run. It is also populated (or cleared) when a new Recipe is selected from the ScheduleRecipe list. The formula <default> will always be available.
- **Batch Scheduling Train List (ScheduleTrain)**
  This list contains all trains in the Batch Scheduler. This list is populated when the ScheduleUpdateTrainList method is run.
- **Batch Scheduling Recipe State List (ScheduleState)**
  This list contains all recipe states in the Batch Scheduler. This list is populated when the ScheduleUpdateStateList method is run.
- **Batch Scheduling Recipe Type List (ScheduleType)**
  This list contains all recipe types in the Batch Scheduler. This list is populated when the ScheduleUpdateTypeList method is run.
- **Batch Execution Schedule List (BatchSched)**
  This list contains all active batches in the Batch Display. This list is populated automatically. By default, all the available batches are displayed in this list. An application can also obtain a unit-centric view by providing the Batch Manager with a list of units. This is done with the BatchSetFocus method.
- **Phase Execution Phase List (PhasePhase)**
  This list contains all active phases for a batch or unit in the Batch Display. This list is populated when the RecipeSetCLBFocus method or the RecipeSetUnitFocus method is run.
- **Phase Execution Parameter List (PhaseParam)**
  This list contains all parameters for an active phase in the Batch Display. This list is populated when a phase is selected in the Phase Execution Phase List. This is a child list to the Phase Execution Phase List.
- **Phase Execution Interlock List (Phaseilock)**
  This list contains all phase interlocks in the Batch Display. This list is populated when a phase is selected in the Phase Execution Phase List. This is a child list to the Phase Execution Phase List.
- **Answer Questions Question List (QuestionQuest)**
  This list contains all active questions in the Batch Display. This list is populated when the RecipeSetCLBFocus method or the RecipeSetUnitFocus method is run.

- **Equipment Selection Instance List (SelectInst)**
  This list contains all process instances that require manual unit selection in the Batch Display. This list is populated when the RecipeSetCLBFocus method or the RecipeSetUnitFocus method is run.
- **Equipment Selection Equipment List (SelectEquip)**
  This list contains all units available for manual unit selection for a specific process instance in the Batch Display. This list is populated when an equipment instance is selected in the Equipment Selection Instance List. This is a child list to the Equipment Selection Instance List.
- **Equipment Allocation Equipment List (EquipmentEquip)**
  This list contains all equipment available for manual allocation in the Batch Display. This list is populated when the EquipmentSetCLBFocus method or the EquipmentSetUnitFocus method is run.
- **Equipment Allocation Instance List (EquipmentInst)**
  This list contains all process and transfer instances available for manual allocation in the Batch Display. This list is populated when an equipment item is selected in the Equipment Allocation Equipment List. This is a child list to the Equipment Allocation Equipment List.
- **Equipment Allocation Queue List (AllocQueue)**
  This list contains all active batches waiting for a unit or connection in the allocation queue. This list is populated when the EquipmentSetCLBFocus method or the EquipmentSetUnitFocus method is run.
- **Display Batch Messages Message List (MessageMsg)**
  This list contains all batch messages in the Batch Display. This list is populated when the MessageSetCLBFocus method or the MessageSetUnitFocus method is run.
- **Edit Phase Phase List (EditPhasePhase)**
  This list contains all recipe phases available for editing in the Batch Display. This list is populated when the EditPhaseSetCLBFocus method is run. There is no unit-centric view of this list.
- **Edit Phase Parameter List (EditPhaseParam)**
  This list contains all recipe phase parameters available for editing in the Batch Display. This list is populated when a phase is selected in the Edit Phase Phase List. This is a child list to the Edit Phase Phase List.
- **View Transition Transition List (ViewTransitionTrans)**
  This list contains all active transition logic expressions in the Batch Display. This list is populated when the ViewTransitionSetCLBFocus method is run. There is no unit-centric view of this list.
- **View Transition Tag List (ViewTransitionTag)**
  This list contains the tags that are used in an active transition logic expression in the Batch Display. This list is populated when an active transition is selected in the View Transition Transition List. This is a child list to the View Transition Transition List.
- **Display Errors Error List (ErrorErr)**
  This list contains all batch error messages in the Batch Display. This list is populated automatically by Batch Manager.

Tying the Functions Together (Setting Focus)

Many of the OcxBatch lists are inter-related. Selecting an item in one (parent) list may cause the view of other (child) lists to update. You can tie together several of the functional areas of the OcxBatch through a small amount of programming in the container. For example, when a batch is selected in the Batch Execution Schedule List, most of the other functional areas set their focus to correspond to the selected batch. Therefore, the Phase Execution Phase List can list the active phases for the batch selected in the Batch Execution Schedule List. This example is known as batch-centric focus setting. Functions can alternately set their focus based on a unit-centric model. In this case, there is less programming to connect the OcxBatch functional areas. Focus is set by using methods provided in each of the various functional areas.

Security Considerations

Many of the methods exposed by OcxBatch allow you to pass DoneBy, DoneByPassword, CheckBy, and CheckByPassword user information to the action. In most cases these arguments can simply be empty strings (""). If you do not provide information for these arguments, a security dialog box pops up at run time to enable an operator to enter the appropriate security information. If you do not want a run-time security dialog pop-up to appear, either disable security for the batch functions or provide security information through these arguments. One theoretical use of this feature is to allow the operator to log into a batch application once and have their security information passed to all of the batch control methods.

Programming Note

Most of the batch operations are run through an asynchronous interface to the Batch Server Batch Manager application. Therefore, data coming back in response to an action is not always available immediately. Instead, you must wait for the data to come back asynchronously. This can necessitate some complex code if you are designing a very synchronous application.

Method and Event Overview by Logical Groups

The following section lists methods and events by logical groups.

System-Level Methods and Events

The following list shows methods at the system level:

- Init()
- Term()
- MiscGet2Levels()
- MiscProductCheck().

  This method has two events:

  - SecurityPending
  - SystemShuttingDown

Batch Scheduler-Level Methods and Events

The following list shows methods and events at the batch scheduler level:

Defining Batches

Use these methods to schedule batches:

- ScheduleAddBatch()
- ScheduleAddBatchWC()
- ScheduleMultiAddBatch()
- ScheduleMultiAddBatchWC()
- ScheduleDeleteBatch()
- ScheduleDeleteBatchWC()
- ScheduleChangeBatch()
- ScheduleChangeBatchWC()

- ScheduleCleanup()
- ScheduleCleanupWC()
- ScheduleInitBatch()
- ScheduleInitBatchWC()
- ScheduleInitAll()
- ScheduleInitAllWC()
- ScheduleMoveBatchUp()
- ScheduleMoveBatchUpWC()
- ScheduleMoveBatchDown()
- ScheduleMoveBatchDownWC()
- ScheduleMoveBatchAfter()
- ScheduleMoveBatchAfterWC()
- ScheduleMoveBatchBefore()
- ScheduleMoveBatchBeforeWC()
- ScheduleSetExecInOrder()
- ScheduleSetExecInOrderWC()
- ScheduleGetExecInOrder()
- ScheduleSetAccessSecurity()
- ScheduleClearAccessSecurity()
- ScheduleHasValidAccessSecurity()
- ScheduleSchedGetSelected()
- ScheduleSchedSetSelected()
- ScheduleSchedGetNumItems()
- ScheduleSchedGetItem()
- ScheduleSchedGetCampaign()
- ScheduleSchedGetLot()
- ScheduleSchedGetBatch()
- ScheduleSchedGetRecipeId()
- ScheduleSchedGetSize()
- ScheduleSchedGetTrain()
- ScheduleSchedGetMode()
- ScheduleSchedGetFormula()
- ScheduleAddFormulaBatch()
- ScheduleAddFormulaBatch()
- ScheduleMultiAddFormulaBatch()
- ScheduleMultiAddFormulaBatchWC()

- ScheduleChangeFormulaBatch()
- ScheduleChangeFormulaBatchWC()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use these events with appropriate methods that schedule batches:

- ScheduleExecInOrder
- ScheduleSchedUpdate
- ScheduleSchedAdd
- ScheduleSchedDelete
- ScheduleSchedChange
- ScheduleSchedSelect
- ScheduleSchedBusy

Recipes

Use the following methods to view and select recipes:

- ScheduleUpdateRecipeList()
- ScheduleRecipeGetSelected()
- ScheduleRecipeSetSelected()
- ScheduleRecipeGetNumItem()
- ScheduleRecipeGetItem()
- ScheduleRecipeGetRecipeId()
- ScheduleRecipeGetRecipeName()
- ScheduleRecipeGetRecipeType()
- ScheduleRecipeGetRecipeState()

You can use the following events with methods that schedule recipes:

- ScheduleRecipeUpdate
- ScheduleRecipeSelect
- ScheduleRecipeBusy

Formula

Use the following methods to view and select recipe specific formulas:

- ScheduleUpdateFormulaList()
- ScheduleFormulaGetItem()
- ScheduleFormulaGetSelected()
- ScheduleFormulaSetSelected()
- ScheduleFormulaGetNumItems()

You can use the following events with methods that schedule trains:

- ScheduleFormulaUpdate

- ScheduleFormulaSelect
- ScheduleFormulaBusy

Trains

Use the following methods to view and select trains:

- ScheduleUpdateTrainList()
- ScheduleTrainGetSelected()
- ScheduleTrainSetSelected()
- ScheduleTrainGetNumItems()
- ScheduleTrainGetItem()
- ScheduleTrainGetTrain()

You can use the following events with methods that schedule trains:

- ScheduleTrainUpdate
- ScheduleTrainSelect
- ScheduleTrainBusy

Types

Use the following methods to view and select recipe types:

- ScheduleUpdateTypeList()
- ScheduleTypeGetSelected()
- ScheduleTypeSetSelected()
- ScheduleTypeGetNumItems()
- ScheduleTypeGetItem()
- ScheduleTypeGetType()

You can use the following events with methods that view and select recipe types:

- ScheduleTypeUpdate
- ScheduleTypeSelect
- ScheduleTypeBusy

States

Use the following methods to view and select recipe states:

- ScheduleUpdateStateList()
- ScheduleStateGetSelected()
- ScheduleStateSetSelected()
- ScheduleStateGetNumItems()
- ScheduleStateGetItem()
- ScheduleStateGetState()

You can use the following events with methods that view and select recipe states:

- ScheduleStateUpdate
- ScheduleStateSelect
- ScheduleStateBusy

Batch-Level Methods and Events

The following lists show methods and events at the batch level:

Batch Focus

Use the following methods to set focus:

- BatchSetFocus()
- RecipeSetUnitFocus()
- RecipeSetCLBFocus()

You can use the following event with methods that set focus:

BatchSchedFocusState

Batch Control

Use the following methods to control batches:

- BatchAbortBatch()
- BatchAbortBatchWC()
- BatchHoldBatch()
- BatchHoldBatchWC()
- BatchRestartBatch()
- BatchRestartBatchWC()
- BatchStartBatch()
- BatchStartBatchWC()
- BatchLockBatch()
- BatchLockBatchWC()
- BatchUnlockBatch()
- BatchUnlockBatchWC()
- BatchStartManualOperationWC()
- BatchSetAutomaticMode()
- BatchSetAutomaticModeWC()
- BatchSetSemiAutoMode()
- BatchSetSemiAutoModeWC()
- BatchSetManualMode()
- BatchSetManualModeWC()
- CommentEnterComment()
- CommentEnterCommentWC()

- SaveRecipeSave()
- SaveRecipeSaveWC()
- SaveRecipeRecipeExists()
- MiscGetRecipeDefBatchSize()
- MiscGetRecipeMinBatchSize()
- MiscGetRecipeMaxBatchSize()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

You can use the following events with appropriate methods that control batches:

- SaveRecipeAuthorChanged
- BatchMsgBoxMessage

Active Batch List Manipulation

Use the following methods to manipulate active batch lists:

- BatchSchedGetSelected()
- BatchSchedSetSelected()
- BatchSchedGetNumItems()
- BatchSchedGetItem()
- BatchSchedGetCLB()
- BatchSchedGetRecipeId()
- BatchSchedGetSize()
- BatchSchedGetTrain()
- BatchSchedGetMode()
- BatchSchedGetModeStr()
- BatchSchedGetStatus()
- BatchSchedGetStatusStr()
- BatchSchedGetEditMask()
- BatchSchedGetFormula()

You can use the following events with appropriate methods that manipulate batch lists:

- BatchSchedUpdate
- BatchSchedAdd
- BatchSchedDelete
- BatchSchedChange
- BatchSchedSelect
- BatchSchedBusy

Batch Questions

Use the following methods to obtain batch information:

- QuestAnswerQuest()
- QuestAnswerQuestWC()
- QuestionQuestGetSelected()
- QuestionQuestSetSelected()
- QuestionQuestGetNumItems()
- QuestionQuestGetItem()
- QuestionQuestGetQuestion()
- QuestionQuestGetType()
- QuestionQuestGetSecurity()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with appropriate methods that obtain batch information:

- QuestionQuestUpdate
- QuestionQuestAdd
- QuestionQuestDelete
- QuestionQuestSelect
- QuestionQuestBusy

Phase Parameter Editor Interface

Use the following methods to edit phase parameters:

- EditPhaseSetCLBFocus()
- EditPhaseChangePhase()
- EditPhaseChangePhaseWC()
- EditPhaseChangeParam()
- EditPhaseChangeParamWC()
- EditPhaseInstrGetInstr()
- EditPhasePhaseGetSelected()
- EditPhasePhaseSetSelected()
- EditPhasePhaseGetNumItems()
- EditPhasePhaseGetItem()
- EditPhasePhaseGetInstance()
- EditPhasePhaseGetUnitProcedure()
- EditPhasePhaseGetOperation()
- EditPhasePhaseGetLabel()
- EditPhasePhaseGetPhase()
- EditPhasePhaseGetDescription()
- EditPhasePhaseGetOperMask()

- EditPhaseParamGetSelected()
- EditPhaseParamSetSelected()
- EditPhaseParamGetNumItems()
- EditPhaseParamGetItem()
- EditPhaseParamGetParameter()
- EditPhaseParamGetType()
- EditPhaseParamGetMaterialId()
- EditPhaseParamGetMaterialName()
- EditPhaseParamGetUom()
- EditPhaseParamGetDataClass()
- EditPhaseParamGetSet()
- EditPhaseParamGetValue()
- EditPhaseParamGetDescription()
- MiscGetEnumName()
- MiscGetEnumNameByRow()
- MiscGetEnumValue()
- MiscGetNumEnums()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with appropriate methods that edit phase parameters:

- EditPhasePhaseUpdate
- EditPhasePhaseAdd
- EditPhasePhaseDelete
- EditPhasePhaseSelect
- EditPhasePhaseBusy
- EditPhaseParamUpdate
- EditPhaseParamChange
- EditPhaseParamSelect
- EditPhaseParamBusy
- EditPhaseInstrUpdate

Transition Logic Expressions

Use the following methods to view transition logic expressions:

- ViewTransitionSetCLBFocus()
- ViewTransitionForceTransition()
- ViewTransitionForceTransitionWC()
- ViewTransitionTransGetSelected()

- ViewTransitionTransSetSelected()
- ViewTransitionTransGetNumItems()
- ViewTransitionTransGetItem()
- ViewTransitionTransGetTrans()
- ViewTransitionTransGetLabel()
- ViewTransitionTransGetDesc()
- ViewTransitionExpGetExp()
- ViewTransitionTagGetSelected()
- ViewTransitionTagSetSelected()
- ViewTransitionTagGetNumItems()
- ViewTransitionTagGetItem()
- ViewTransitionTagGetTag()
- ViewTransitionTagGetValue()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with appropriate methods that view transition logic expressions:

- ViewTransitionExpUpdate
- ViewTransitionTagUpdate
- ViewTransitionTagChange
- ViewTransitionTagSelect
- ViewTransitionTagBusy
- ViewTransitionTransUpdate
- ViewTransitionTransAdd
- ViewTransitionTransDelete
- ViewTransitionTransChange
- ViewTransitionTransSelect
- ViewTransitionTransBusy

Batch Information Messages

Use the following methods for batch information messages:

- MessageSetUnitFocus()
- MessageSetCLBFocus()
- MessageMsgGetSelected()
- MessageMsgSetSelected()
- MessageMsgGetNumItems()
- MessageMsgGetItem()
- MessageMsgGetMessage()

- MiscGetMessage()

Use the following events with appropriate methods for batch information messages:

- MessageMsgUpdate
- MessageMsgAdd
- MessageMsgDelete
- MessageMsgSelect
- MessageMsgBusy

Batch Error Messages

Use the following methods for batch error messages:

- ErrorClear()
- ErrorErrGetNumItems()
- ErrorErrGetItem()

Use the following events with the appropriate methods for batch error messages:

- ErrorErrUpdate
- ErrorErrAdd

Phase-Level Functions

The following lists show methods and events at the phase level.

Phase Control

Use the following methods for phase control:

- PhaseAbortPhase()
- PhaseAbortPhaseWC()
- PhaseHoldPhase()
- PhaseHoldPhaseWC()
- PhaseRestartPhase()
- PhaseRestartPhaseWC()
- PhaseStartPhase()
- PhaseStartPhaseWC()
- PhaseAckPhase()
- PhaseAckPhaseWC()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Phase Properties

Use the following methods for phase properties:

- PhaseInstrGetInstr()
- PhaseSetCtrlButton1()

- PhaseSetCtrlButton1WC()
- PhaseSetCtrlButton2()
- PhaseSetCtrlButton2WC()
- PhaseEnterComment()
- PhaseEnterCommentWC()
- PhaseEditParameter()
- PhaseEditParameterWC()
- PhasePhaseGetSelected()
- PhasePhaseSetSelected()
- PhasePhaseGetNumItems()
- PhasePhaseGetItem()
- PhasePhaseGetEquipment()
- PhasePhaseGetUnitProcedure()
- PhasePhaseGetOperation()
- PhasePhaseGetPhase()
- PhasePhaseGetStatus()
- PhasePhaseGetDescription()
- PhasePhaseGetEditMask()
- PhasePhaseGetCtrlButtonLabel1()
- PhasePhaseGetCtrlButtonLabel2()
- PhasePhaseGetOperMsg()
- PhasePhaseGetOperMsgSt()
- PhasePhaseGetLabel()
- PhasePhaseGetDocPath()
- PhaseViewDocument()
- PhaseViewDocumentWC()
- PhaseAckDocument()
- PhaseAckDocumentWC()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following event with appropriate methods for phase properties:

- PhaseInstrUpdate

Phase Formula Parameters

Use the following methods to define phase formula parameters:

- PhaseParamGetSelected()
- PhaseParamGetNumItems()

- PhaseParamGetItem()
- PhaseParamGetParam()
- PhaseParamGetType()
- PhaseParamGetExt ()
- PhaseParamGetEditType()
- PhaseParamGetUom()
- PhaseParamGetDataClass()
- PhaseParamGetSet()
- PhaseParamGetExtValue()
- PhaseParamGetDescription()
- MiscGetEnumName()

Use the following events with the appropriate methods that define phase formula parameters:

- PhaseParamUpdate
- PhaseParamChange
- PhaseParamSelect
- PhaseParamBusy

Phase Interlocks

Use the following methods to define phase interlocks:

- PhaseIlockGetSelected()
- PhaseIlockSetSelected()
- PhaseIlockGetNumItems()
- PhaseIlockGetItem()
- PhaseIlockGetIlock()
- PhaseIlockGetStatus()

Use the following events with the appropriate methods that define phase interlocks:

- PhaseIlockUpdate
- PhaseIlockChange
- PhaseIlockSelect
- PhaseIlockBusy

Equipment-Level Functions

The following list shows methods and events at the equipment level.

Equipment Selection

Use the following methods to select equipment:

- SelectSelectEquip()
- SelectSelectEquipWC()

- SelectInstGetSelected()
- SelectInstSetSelected()
- SelectInstGetNumItems()
- SelectInstGetItem()
- SelectInstGetInstance()
- SelectEquipGetSelected()
- SelectEquipSetSelected()
- SelectEquipGetNumItems()
- SelectEquipGetItem()
- SelectEquipGetEquipment()
- SelectEquipGetStatus()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with the appropriate methods to select equipment:

- SelectInstUpdate
- SelectInstAdd
- SelectInstDelete
- SelectInstSelect
- SelectInstBusy
- SelectEquipUpdate
- SelectEquipAdd
- SelectEquipDelete
- SelectEquipChange
- SelectEquipSelect
- SelectEquipBusy

Equipment Allocation

Use the following methods to allocate equipment:

- EquipmentSetUnitFocus()
- EquipmentSetCLBFocus()
- EquipmentAllocateEquipment()
- EquipmentAllocateEquipmentWC()
- EquipmentReleaseEquipment()
- EquipmentReleaseEquipmentWC()
- EquipmentAbortUnit()
- EquipmentAbortUnitWC()
- EquipmentHoldUnit()

- EquipmentHoldUnitWC()
- EquipmentRestartUnit ()
- EquipmentRestartUnitWC()
- EquipmentEquipGetSelected()
- EquipmentEquipSetSelected()
- EquipmentEquipGetNumItems()
- EquipmentEquipGetItem()
- EquipmentEquipGetEquipment()
- EquipmentEquipGetType()
- EquipmentEquipGetAllocation()
- EquipmentEquipGetStatus()
- EquipmentEquipGetUnitCtrl()
- EquipmentInstGetSelected()
- EquipmentInstSetSelected()
- EquipmentInstGetNumItem()
- EquipmentInstGetItem()
- EquipmentInstGetInstance()

NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with the appropriate methods to allocate equipment:

- EquipmentEquipUpdate
- EquipmentEquipChange
- EquipmentEquipSelect
- EquipmentEquipBusy
- EquipmentInstUpdate
- EquipmentInstSelect
- EquipmentInstBusy

Equipment Allocation Queue

Use the following methods to allocate equipment queues:

- AllocQueueSetAccessSecurity()
- AllocQueueClearAccessSecurity()
- AllocQueHasValidAccessSecurity()
- AllocQueueSetEquipFocus()
- AllocQueueGetSelected()
- AllocQueueSetSelected()
- AllocQueueGetNumItems()

- AllocQueueGetInstance()
- AllocQueueGetItem()
- AllocQueueGetCampaign()
- AllocQueueGetLot()
- AllocQueueGetBatch()
- AllocQueueGetRecipeId()
- AllocQueueGetSize()
- AllocQueueGetTrain()
- AllocQueueGetMode()
- AllocQueueGetSequence()
- AllocQueueMoveBatchUp()
- AllocQueueMoveBatchUpWC()
- AllocQueueMoveBatchDown()
- AllocQueueMoveBatchDownWC()
- AllocQueueGetFormula()

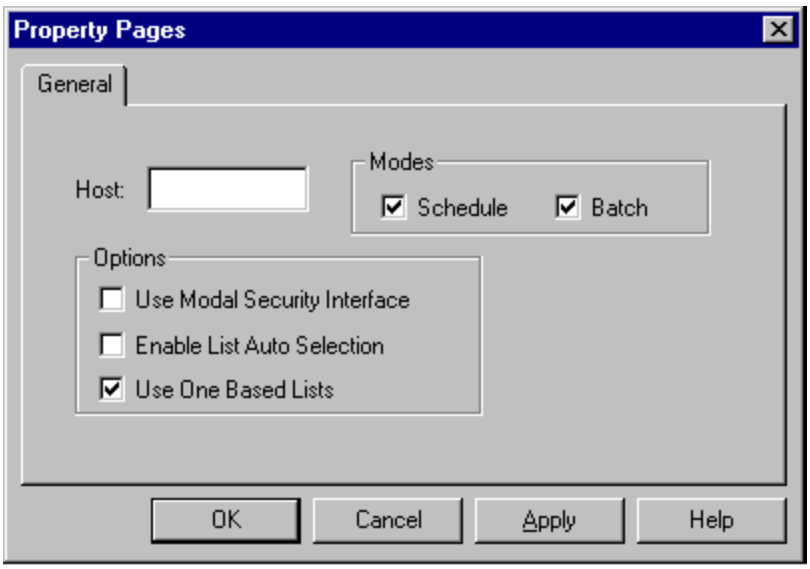
NOTE: Any method suffixed with "WC" is for providing the DoneBy/CheckBy comments as arguments.

Use the following events with the appropriate methods that allocate equipment queues:

- AllocQueueAdd
- AllocQueueChange
- AllocQueueDelete
- AllocQueueSelect
- AllocQueueUpdate

Properties

Each batch object has a Control Name and other properties available for configuration.



This section describes properties that are available for the Batch ActiveX object.

AutoSelectMode

This property specifies the list select policy that OcxBatch uses:

0 = Disable auto-selection
1 = Enable auto-selection

You must set this property before you call the **Init** method. You can set the property programmatically or through the property sheet.

Data Type: Short

Access: RW

BatchMode

This property specifies that the control is to be used with all the functional areas except the Batch Scheduling functional area.

Note: Batch scheduling can be used simultaneously by setting the ScheduleMode property.

You must set this property b**efore you** call the **Init** method. You can set the property programmatically or through the property sheet.

Data Type: Short

Access: RW

EnvInstance

This property specifies the Instance name that this control should use to connect to the batch Environment Manager application.

Typically, you do not set this property to anything. Use the property only if you are trying to create an application that exists in the batch environment database.

Data Type: String

Access: RW

Host

This property contains the same of the computer running Batch Manager.

You must set this property b**efore you** call the **Init** method. You can set the property programmatically or through the property sheet.

Data Type: String

Access: RW

OneBasedMode

This property specifies the list base policy OcxBatch uses:

0 = Zero-based lists
1 = One-based lists

The setting of this property is usually driven by the operation of the GUI list control that is used in the container to show the OcxBatch lists.

You must set this property b**efore you** call the **Init** method. You can set the property programmatically or through the property sheet.

When you use zero-based lists, –1 signifies that no items are selected. When you use one-based lists, 0 signifies that no items are selected.

Data Type: Short

Access: RW

ScheduleMode

This property specifies that the control is used with the Batch Scheduling and Display Errors functional areas.

You must set this property b**efore you** call the **Init** method. You can set the property programmatically or through the property sheet.

Data Type: Short

Access: RW

SecurityMode

This property specifies the type of security dialog that is displayed at run time:

0 = Enable modeless dialog
1 = Enable the modal dialog

You must set this property b**efore you** call the **Init** method. You can set the property programmatically or through the property sheet.

Data Type: Short

Access: RW

Methods

The following methods are available for the Batch Management ActiveX object.

AllocQueHasValidAccessSecurity()

This method determines if the operator passed the security requirements for manipulating the allocation queue. This method can be called from within the OnSecurityPending event if the security operation is complete (for example, State = 0). A Short return value provides the result. The value can be interpreted as follows:

0 = User failed security
1 = User passed security

Syntax

```
ReturnValue = OcxBatchVar.AllocQueHasValidAccessSecurity ()
```

AllocQueueClearAccessSecurity()

This method cancels all allocation queue security set by the AllocQueueSetAccessSecurity method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueClearAccessSecurity ()
```

AllocQueueGetBatch()

This method queries the batch identification of a batch based on its index in the AllocQueue list. A String return value (16 characters) provides the batch ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetBatch (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetCampaign()

This method queries the campaign identification of a batch based on its index in the AllocQueue list. A String return value (16 characters) provides the campaign ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetCampaign (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetInstance()

This method queries the equipment instance of a batch based on its index in the AllocQueue list. A String return value (16 characters) contains the equipment instance name used by the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetInstance (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetItem()

This method queries all information of a batch based on its index in the AllocQueue list. A String return value (155 characters) contains the campaign ID, lot ID, batch ID, equipment instance, recipe ID, batch size, train name, batch mode, batch status, and sequence number fields each separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetLot()

This method queries the lot identification of a batch based on its index in the AllocQueue list. A String return value (16 characters) provides the lot ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetLot (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetMode()

This method queries the mode of a batch based on its index in the AllocQueue list. A Short return value provides the mode of the batch. The value can be interpreted as follows:

0 = Automatic
1 = Semi-automatic
2 = Manual

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetMode (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetNumItems()

This method queries the number of batches in the AllocQueue list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetNumItems ()
```

AllocQueueGetRecipeId()

This method queries the recipe identification of a batch based on its index in the AllocQueue list. A String return value (16 characters) provides the recipe ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetRecipeId (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetSelected()

This method queries the index of the currently selected batch in the AllocQueue list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetSelected ()
```

AllocQueueGetSequence()

This method queries the sequence number (the position in the list) of a batch based on its index in the AllocQueue list. A Short return value provides the sequence number of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetSequence (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetSize()

This method queries the size of a batch based on its index in the AllocQueue list. A Long return value provides the size of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetSize (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetTrain()

This method queries the train name of a batch based on its index in the AllocQueue list. A String return value (16 characters) provides the train name of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetTrain (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueMoveBatchDown()

This method moves the currently selected batch in the AllocQueue list down by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueMoveBatchDown ( Campaign, Lot, Batch,
Instance, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
ID of the campaign

Lot
Data Type: String
ID of the lot

Batch
Data Type: String
ID of the batch

Instance
Data Type: String
Name of the unit instance

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

AllocQueueMoveBatchDownWC()

This method moves the currently selected batch in the AllocQueue list down by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the AllocQueueMoveBatchDownWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueMoveBatchDownWC ( Campaign, Lot, Batch, Instance, DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
ID of the campaign

Lot
Data Type: String
ID of the lot

Batch
Data Type: String
ID of the batch

Instance
Data Type: String
Name of the unit instance

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by passwordDoneByComment
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

AllocQueueMoveBatchUp()

This method moves the currently selected batch in the AllocQueue list up by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueMoveBatchUp ( Campaign, Lot, Batch,
Instance, DoneBy, CheckBy, DoneByPswd, CheckByPswd )
```

Parameters

Campaign
Data Type: String
ID of the campaign

Lot
Data Type: String
ID of the lot

Batch
Data Type: String
ID of the batch

Instance
Data Type: String
Name of the unit instance

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String

Done by password

CheckByPswd
Data Type: String
Check by password

AllocQueueMoveBatchUpWC()

This method moves the currently selected batch in the AllocQueue list up by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the AllocQueueMoveBatchUpWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueMoveBatchUpWC ( Campaign, Lot, Batch,
Instance, DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment,
CheckByComment)
```

Parameters

Campaign
Data Type: String
ID of the campaign

Lot
Data Type: String
ID of the lot

Batch
Data Type: String
ID of the batch

Instance
Data Type: String
Name of the unit instance

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String

Done by comment

CheckByComment
Data Type: String
Check by comment

AllocQueueSetAccessSecurity()

This method opens a modeless security dialog box if the Allocation Queue security function is defined in the Security Editor. After a user enters his security information, the OnSecurityPending event is called. In this event, an application can determine if the user is allowed to manipulate the allocation queue. Using this technique, the user does not need to be prompted for security every time he wants to view or manipulate the allocation queue. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueSetAccessSecurity ()
```

AllocQueueSetEquipFocus()

This is a required method for populating the Allocation Queue list. The name of the unit or connection is passed as a parameter and the allocation queue list updates with any batches waiting for the equipment. This method should be called when the selected equipment is Allocated or Busy. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.AllocQueueSetEquipFocus ( Equipment)
```

Parameters

Equipment
Data Type: String
Name of equipment to focus

AllocQueueSetSelected()

This method selects a specific batch based on its index in the AllocQueue list. This method is required in order to perform actions on a particular batch (move up or move down).

Syntax

```
OcxBatchVar.AllocQueueSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

AllocQueueGetFormula()

This method queries the formula name of a batch based on its index in the AllocQueue list. A String return value (128 characters) provides the formula name of the batch.

Syntax

```
ReturnValue = OcxBatchVar.AllocQueueGetFormula (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchAbortBatch()

This method aborts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.OcxBatchVar.BatchAbortBatch ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchAbortBatchWC()

This method aborts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: Use the BatchAbortBatchWC() syntax only when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.OcxBatchVar.BatchAbortBatchWC ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchHoldBatch()

This method holds the currently selected batch in the BatchSched list. This method should be called only if the batch status is Run. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchHoldBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchHoldBatchWC()

This method holds the currently selected batch in the BatchSched list. This method should be called only if the batch status is Run. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: Use the BatchHoldBatchWC () method syntax only when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchHoldBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String

Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchLockBatch()

This method locks the currently selected batch in the BatchSched list so that a run-time procedural jump can occur. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchLockBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchLockBatchWC()

This method locks the currently selected batch in the BatchSched list so that a run-time procedural jump can occur. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchLockBatchWC () when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchLockBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchRestartBatch()

This method starts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchRestartBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchRestartBatchWC()

This method starts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchRestartBatchWC () when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchRestartBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchSchedGetCLB()

This method queries the CLB of an active batch based on its index in the BatchSched list. A String return value (50 characters) provides the CLB with the campaign, lot, and batch fields combined and separated by forward slash (/) characters. This string can be used directly to set the CLB focus of other functional areas (RecipeSetCLBFocus).

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetCLB (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetEditMask()

This method queries the permissible actions of an active batch based on its index in the BatchSched list. The Long return value is a bit mask that can be used to determine what batch actions are permitted. If the bit is set, the action is permitted. If it is clear, the action is not permitted. The Operator Action Required bit is set if the batch is waiting for some operator input (such as Answer Question). The value can be interpreted as follows:

0 = Batch start
1 = Batch hold
2 = Batch restart
3 = Batch abort
4 = Set batch mode
5 = Operator action is pending
6 = Batch can be locked
7 = Batch can be unlocked

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetEditMask (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetItem()

This method queries all information of an active batch based on its index in the BatchSched list. A String return value (131 characters) contains the campaign, lot, batch, recipe, size, train, mode, status, and action required fields each separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetMode()

This method queries the mode (as an integer) of an active batch based on its index in the BatchSched list. A Long return value provides the mode of the batch. The value can be interpreted as follows:

0 = Automatic
1 = Semi-Automatic
2 = Manual

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetMode (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetModeStr()

This method queries the mode (as a string) of an active batch based on its index in the BatchSched list. A String return value (9 characters) provides the mode of the batch.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetModeStr (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetNumItems()

This method queries the number of active batches in the BatchSched list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetNumItems ()
```

BatchSchedGetRecipeId()

This method queries the recipe identification of an active batch based on its index in the BatchSched list. A String return value (16 characters) provides the recipe ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetRecipeId (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetSelected()

This method queries the index of the currently selected active batch in the BatchSched list. A Short return value provides the selected item's index.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetSelected ()
```

BatchSchedGetSize()

This method queries the size of an active batch based on its index in the BatchSched list. A Long return value provides the batch size.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetSize (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetStatus()

This method queries the status (as an integer) of an active batch based on its index in the BatchSched list. A Long return value provides the status of the batch. The value can be interpreted as follows:

0 = Open
1 = Ready
3 = Run
4 = Done
6 = Aborted
7 = Held
8 = Aborting
10 = Locking
11 = Locked

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetStatus (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetStatusStr()

This method queries the status (as a string) of an active batch based on its index in the BatchSched list. A String return value (8 characters) provides the status of the batch.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetStatusStr (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedGetTrain()

This method queries the assigned train of an active batch based on its index in the BatchSched list. A String return value (16 characters) provides the train name.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetTrain (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSchedSetSelected()

This method selects a specific batch in the active batch list based on its index in the BatchSched list. This method is required in order to perform actions on an active batch (such as start or hold) and also for populating dependent lists (such as phases and parameters).

Syntax

```
OcxBatchVar.BatchSchedSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

BatchSetAutomaticMode()

This method changes the mode of the currently selected batch in the BatchSched list to Automatic. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetAutomaticMode ( DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchSetAutomaticModeWC()

This method changes the mode of the currently selected batch in the BatchSched list to Automatic. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchSetAutomaticModeWC () when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetAutomaticModeWC (DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String

Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchSetFocus()

This method creates a unit-centric view of the Batch Execution schedule. Only batches that are related to units in the passed list (through their train) are displayed in the BatchSched list. Passing an empty string ("") into the UnitList argument reverts the view back to batch-centric. By default the Batch Execution function is batch-centric. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetFocus (UnitList)
```

Parameters

UnitList
Data Type: String
List of units, 16 characters each, separated by a forward slash (/) character

BatchSetJumpToken()

This method moves the position of the cursor in a batch to the designated jump point. The parameter represents the desired jump point and can be obtained from the GetJumpTokenId method available in the SFC ActiveX Control. The batch must be locked using the BatchLockBatch method prior to moving the cursor and must be unlocked using the BatchUnlockBatch method after the jump is complete.

Syntax

```
OcxBatchVar.BatchSetJumpToken (Jump_Token_Id)
```

Parameters

Jump_Token_Id
Data Type: Long
Position to which to jump in the batch

BatchSetManualMode()

This method changes the mode of the currently selected batch in the BatchSched list to Manual. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetManualMode (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String

Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchSetManualModeWC()

This method changes the mode of the currently selected batch in the BatchSched list to Manual. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchSetManualModeWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetManualModeWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchSetSemiAutoMode()

This method changes the mode of the currently selected batch in the BatchSched list to Semi-Automatic. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetSemiAutoMode (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchSetSemiAutoModeWC ()

This method changes the mode of the currently selected batch in the BatchSched list to Semi-Automatic. This method should only be called if the batch status is Ready, Run, or Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchSetSemiAutoModeWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchSetSemiAutoModeWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchStartBatch()

his method starts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchStartBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchStartBatchWC()

This method starts the currently selected batch in the BatchSched list. This method should only be called if the batch status is Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchStartBatchWC () method syntax when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchStartBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchStartManualOperation()

This method enables the creation and start of the manual operation batch. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.BatchStartManualOperation ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchStartManualOperationWC()

This method enables the creation and start of the manual operation batch. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the BatchStartManualOperationWC () when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchStartManualOperationWC (DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchUnlockBatch()

This method unlocks the currently selected batch in the BatchSched list following a run-time procedural jump. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

Syntax

```
ReturnCode = OcxBatchVar.BatchUnlockBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

BatchUnlockBatchWC()

This method unlocks the currently selected batch in the BatchSched list following a run-time procedural jump. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

NOTE: You must use the BatchUnlockBatchWC () when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.BatchUnlockBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

BatchSchedGetFormula()

This method queries the formula identification of an active batch based on its index in the BatchSched list. A String return value (128 characters) provides the Formula name.

Syntax

```
ReturnValue = OcxBatchVar.BatchSchedGetFormula (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

CommentEnterComment()

This method enters a batch comment for the CLB currently in focus. See *"RecipeSetCLBFocus()"* and *"RecipeSetUnitFocus()"*. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.CommentEnterComment (Comment, DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

Comment
Data Type: String
Text for specific comment

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String

Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

CommentEnterCommentWC()

This method enters a batch comment for the CLB currently in focus. See *"RecipeSetCLBFocus()"* and *"RecipeSetUnitFocus()"*. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the CommentEnterCommentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.CommentEnterCommentWC (Comment, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Comment
Data Type: String
Text for specific comment

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EditPhaseChangeParam()

This method changes the value of the parameter currently selected in the EditPhaseParam list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EditPhaseChangeParam (Value, DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

Value
Data Type: String
Value for parameter (as string)

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EditPhaseChangeParamWC()

This method changes the value of the parameter currently selected in the EditPhaseParam list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EditPhaseChangeParamWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EditPhaseChangeParamWC (Value, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Value
Data Type: String
Value for parameter (as string)

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EditPhaseChangePhase()

This method changes the settings of the phase currently selected in the EditPhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EditPhaseChangePhase ( OperMask, Instruct, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

OperMask
Data Type: Long
Mask of bits representing new phase settings:
0 = Ack Entry Required
1 = Ack Entry DoneBy Required
2 = Ack Entry CheckBy Required
3 = Ack Exit Required
4 = Ack Exit DoneBy Required
5 = Ack Exit CheckBy Required
6 = Comment Required

Intstruct
Data Type: String
Instruction text to append

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EditPhaseChangePhaseWC()

This method changes the settings of the phase currently selected in the EditPhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EditPhaseChangePhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EditPhaseChangePhaseWC ( OperMask, Instruct, DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

OperMask
Data Type: Long
Mask of bits representing new phase settings:

0 = Ack Entry Required
1 = Ack Entry DoneBy Required
2 = Ack Entry CheckBy Required
3 = Ack Exit Required
4 = Ack Exit DoneBy Required
5 = Ack Exit CheckBy Required
6 = Comment Required

Intstruct
Data Type: String
Instruction text to append

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EditPhaseInstrGetInstr()

This method queries the instructions of the currently selected phase in the EditPhasePhase list. A String return value (unlimited number of characters) provides the instruction text.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseInstrGetInstr ()
```

EditPhaseParamGetDataClass()

This method queries the data class (as an integer) of a parameter based on its index in the EditPhaseParam list. A Short return value provides the data class. The value can be interpreted as follows:

0 = Analog
1 = Discrete
2 = String
3 = Enumeration

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetDataClass ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetDescription()

This method queries the description of a parameter based on its index in the EditPhaseParam list. A String return value (120 characters) provides the description of the parameter.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetDescription (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetItem()

This method queries all information of a parameter based on its index in the EditPhaseParam list. A String return value (166 characters) contains the parameter name, parameter type, material ID, unit of measure, value, data class, and enumeration set name fields each separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetMaterialId()

This method queries the material identification of a parameter based on its index in the EditPhaseParam list. A String return value (16 characters) provides the material ID of the parameter.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetMaterialId ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetMaterialName()

This method queries the material name of a parameter based on its index in the EditPhaseParam list. A String return value (40 characters) provides the material name of the parameter.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetMaterialName (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetNumItem()

This method queries the number of parameters in the EditPhaseParam list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetNumItems ()
```

EditPhaseParamGetParameter()

This method queries the parameter name of a parameter based on its index in the EditPhaseParam list. A String return value (16 characters) provides the parameter name.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetParameter ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetSelected()

This method queries the index of the currently selected parameter in the EditPhaseParam list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetSelected ()
```

EditPhaseParamGetSet()

This method queries the set name of the enumeration of a parameter based on its index in the EditPhaseParam list. This method should be called if the EditPhaseParamGetDataClass method indicates that the parameter is an enumeration. A String return value (16 characters) contains the enumeration set name.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetSet (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetType()

This method queries the parameter type (as an integer) of a parameter based on its index in the EditPhaseParam list. A Short return value provides the type of parameter. The value can be interpreted as follows:

0 = Input
9 = Process variable
12 = Output

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetUom()

This method queries the engineering units of a parameter based on its index in the EditPhaseParam list. This method should be called if the EditPhaseParamGetType function indicates that the parameter is a process variable. A String return value (16 characters) contains the engineering units value.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetUom (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamGetValue()

This method queries the value of a parameter based on its index in the EditPhaseParam list. A String return value (80 characters) provides the parameter value.

Syntax

```
ReturnValue = OcxBatchVar.EditPhaseParamGetValue (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseParamSetSelected()

This method selects a specific parameter in the parameter list based on its index in the EditPhaseParam list. This method is required to change a parameter value.

Syntax

```
OcxBatchVar.EditPhaseParamSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetDescription()

This method queries the description of a phase based on its index in the EditPhasePhase list. A String return value (120 characters) provides the description of the phase.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetDescription (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetInstance()

This method queries the instance of a phase based on its index in the EditPhasePhase list. A String return value (16 characters) provides the instance name of the phase.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetInstance ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetItem()

This method queries all information of a phase based on its index in the EditPhasePhase list. A String return value (80 characters) contains the instance name, unit procedure name, operation name, phase label, and phase name fields. Each field is separated by two spaces. The operation name is included in the string value regardless of the number of recipe levels defined.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetLabel()

This method queries the label of a phase based on its index in the EditPhasePhase list. A String return value (8 characters) provides the label of the phase.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetLabel (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetNumItems()

This method queries the number of phases in the EditPhasePhase list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetNumItems ()
```

EditPhasePhaseGetOperation()

This method queries the operation name in which a phase is defined based on its index in the EditPhasePhase list. A String return value (16 characters) provides the operation name.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetOperation ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetOperMask()

This method queries the defined actions of a phase based on its index in the EditPhasePhase list. The Long return value is a bit mask that can be used to determine what actions are defined. If the bit is set, the action is defined. If it is clear, the action is not defined. The value can be interpreted as follows.

Bit (Value)	Description
0 (1)	Ack Entry required

Bit (Value)	Description
1 (2)	Ack Entry DoneBy required
2 (4)	Ack Entry DoneBy required
3 (8)	Ack Exit required
4 (16)	Ack Exit DoneBy required
5 (32)	Ack Exit CheckBy required
6 (64)	Comment required
15 (32768)	Continue Mode enabled
20 (1048576)	Continue Mode can be enabled
21 (2097152)	Manual Phase - Ack On Exit always enabled

(7)	General Ack entry mask
(56)	General Ack exit mask
(16447)	General Ack mask

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetOperMask ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetPhase()

This method queries the name of a phase based on its index in the EditPhasePhase list. A String return value (16 characters) provides the phase name.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetPhase (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseGetSelected()

This method queries the index of the currently selected phase in the EditPhasePhase list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetSelected ()
```

EditPhasePhaseGetUnitProcedure()

This method queries the unit procedure name in which a phase is defined based on its index in the EditPhasePhase list. A String return value (16 characters) provides the unit procedure name.

Syntax

```
ReturnValue = OcxBatchVar.EditPhasePhaseGetUnitProcedure (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhasePhaseSetSelected()

This method selects a specific phase in the phase list based on its index in the EditPhasePhase list. This method is required in order to change settings for a phase and also for populating the dependent parameter list.

Syntax

```
OcxBatchVar.EditPhasePhaseSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EditPhaseSetCLBFocus()

This is a required method for using the Edit Phase Parameter functional area and establishes a batch-centric view in the EditPhasePhase list. This method can be called any time as long as the CLB defined by the parameter exists in the BatchSched list. Only the phases that are not active (visible in the PhasePhase list) are available in this list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EditPhaseSetCLBFocus (CLB)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch ID data separated by forward slash (/) characters

EquipmentAbortUnit()

This method aborts the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentAbortUnit (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EquipmentAbortUnitWC()

This method aborts the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EquipmentAbortUnitWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentAbortUnitWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EquipmentAllocateEquipment()

This method allocates the currently selected equipment in the EquipmentEquip list to the currently selected instance in the EquipmentInst list. This method should only be called if the equipment is not allocated. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentAllocateEquipment ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EquipmentAllocateEquipmentWC()

This method allocates the currently selected equipment in the EquipmentEquip list to the currently selected instance in the EquipmentInst list. This method should only be called if the equipment is not allocated. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EquipmentAllocateEquipmentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentAllocateEquipmentWC ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EquipmentEquipGetAllocation()

This method queries the allocation status of a unit or connection based on its index in the EquipmentEquip list. A String return value (16 characters) provides the allocation status of the equipment.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetAllocation ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentEquipGetEquipment()

This method queries the name of a unit or connection based on its index in the EquipmentEquip list. A String return value (16 characters) provides the name of the equipment.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetEquipment ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentEquipGetItem()

This method queries all information of a unit or connection based on its index in the EquipmentEquip list. A String return value (82 characters) contains the equipment name, equipment type, allocation status, equipment status, and equipment state fields. Each field is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetItem (Row)
```

EquipmentEquipGetNumItems()

This method queries the number of units and connections in the EquipmentEquip list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetNumItems ()
```

EquipmentEquipGetSelected()

This method queries the index of the currently selected unit or connection in the EquipmentEquip list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetSelected ()
```

EquipmentEquipGetStatus()

This method queries the equipment status of a unit or connection based on its index in the EquipmentEquip list. A String return value (16 characters) provides the equipment status of the equipment.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetStatus (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentEquipGetType()

This method queries the type of equipment based on its index in the EquipmentEquip list. A Long return value provides the equipment type. The value can be interpreted as follows:

0 = Unit
1 = Connection

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentEquipGetUnitCtrl()

This method queries whether the unit control tags have been defined for a unit or connection based on its index in the EquipmentEquip list. A Long return value provides the existence of the unit control tags. The value can be interpreted as follows:

0 = Enabled
1 = Disabled

Syntax

```
ReturnValue = OcxBatchVar.EquipmentEquipGetUnitCtrl ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentEquipSetSelected()

This method selects a specific unit or connection based on its index in the EquipmentEquip list. This method is required to control the equipment (such as Allocate, Release, or Hold) and also for populating the dependent equipment instance list.

Syntax

```
OcxBatchVar.EquipmentEquipSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentHoldUnit()

This method holds the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentHoldUnit (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EquipmentHoldUnitWC()

This method holds the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EquipmentHoldUnitWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentHoldUnitWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EquipmentInstGetInstance()

This method queries the instance name of a unit or connection based on its index in the EquipmentInst list. A String return value (16 characters) provides the name of the instance.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentInstGetInstance ( Row )
```

```
OcxBatchVar.EquipmentEquipSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentInstGetItem()

This method queries all information of an equipment instance based on its index in the EquipmentInst list. A String return value (16 characters) contains the instance name. This is a one field list. Therefore, the value returned using this method is the same as the value returned by the EquipmentInstGetInstance method.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentInstGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentInstGetNumItems()

This method queries the number of instances in the EquipmentInst list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentInstGetNumItems ()
```

EquipmentInstGetSelected

This method queries the index of the currently selected instance in the EquipmentInst list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.EquipmentInstGetSelected ()
```

EquipmentInstSetSelected()

This method selects a specific instance based on its index in the EquipmentEquip list. This method is required to control the equipment (Allocate, Release, or Hold).

Syntax

```
OcxBatchVar.EquipmentInstSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

EquipmentReleaseEquipment()

This method releases the currently selected equipment in the EquipmentEquip list from the currently selected instance in the EquipmentInst list. This method should be called only if the equipment is allocated. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentReleaseEquipment ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EquipmentReleaseEquipmentWC()

This method releases the currently selected equipment in the EquipmentEquip list from the currently selected instance in the EquipmentInst list. This method should be called only if the equipment is allocated. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EquipmentReleaseEquipmentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentReleaseEquipmentWC ( DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EquipmentRestartUnit()

This method restarts the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentRestartUnit (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

EquipmentRestartUnitWC()

This method restarts the unit currently selected in the EquipmentEquip list. This method should be called only if the EquipmentEquipGetUnitCtrl method is enabled (returns zero). The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the EquipmentRestartUnitWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentRestartUnitWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

EquipmentSetCLBFocus()

This method is required for using the equipment allocation functional area. It establishes a batch-centric view in the EquipmentEquip list. This method can be called only when the CLB defined by the parameter exists in the BatchSched List and has a status of Run, Held, or Aborting. If the batch status is other than this, the parameter to this method should be empty (//). You can use the event BatchSchedFocusState for easy batch-centric focus setting. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentSetCLBFocus (CLB)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch ID data separated by forward slash (/) characters

EquipmentSetUnitFocus()

This method creates a unit-centric view of the Equipment Allocation functional area. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.EquipmentSetUnitFocus (Unit)
```

Parameters

Unit
Data Type: String
Unit name

ErrorClear()

This method clears the ErrorErr list. There is no return value.

Syntax

```
OcxBatchVar.ErrorClear
```

ErrorErrGetItem()

This method queries all information of an error based on its index in the ErrorErr list. A String return value (60 characters) contains the error description.

Syntax

```
ReturnValue = OcxBatchVar.ErrorErrGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ErrorErrGetNumItems()

This method queries the number of errors in the ErrorErr list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ErrorErrGetNumItems ()
```

Init()

This method initializes communication with Batch Manager. The Term method should also be called when application is complete. The SystemShuttingDown event should also be used to clean up properly upon Batch Manager shutdown. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.Init ()
```

MessageMsgGetItem()

This method queries all information of a batch message based on its index in the MessageMsg list. A String return value (80 characters) contains the message. This is a one field list. Therefore, the value returned using this method is the same as the value returned by the MessageMsgGetMessage method.

Syntax

```
ReturnValue = OcxBatchVar.MessageMsgGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

MessageMsgGetMessage()

This method queries the batch message based on its index in the MessageMsg list. A String return value (80 characters) provides the message.

Syntax

```
ReturnValue = OcxBatchVar.MessageMsgGetMessage (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

MessageMsgGetNumItems()

This method queries the number of messages in the MessageMsg list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.MessageMsgGetNumItems ()
```

MessageMsgGetSelected()

This method queries the index of the currently selected message in the MessageMsg list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.MessageMsgGetSelected ()
```

MessageMsgSetSelected()

This method selects a specific batch message based on its index in the MessageMsg list. There are no actions permitted on individual batch messages. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.MessageMsgSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

MessageSetCLBFocus()

This is a required method for using the Batch Messages functional area and establishes a batch-centric view in the MessageMsg list. This method can be called only when the CLB defined by the parameter exists in the BatchSched List and has a status of Run, Held, or Aborting. If the batch status is other than this, the parameter to this method should be empty (//). You can use the event BatchSchedFocusState for easy batch-centric focus setting. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.MessageSetCLBFocus (CLB)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch ID data separated by forward slash (/) characters

MessageSetUnitFocus()

This method creates a unit-centric view of the Batch Messages functional area. The return code is a Short integer value that can be used for error handling.*"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.MessageSetUnitFocus (Unit)
```

Parameters

Unit
Data Type: String
Unit name

MiscGet2Levels()

This method queries whether two or three recipe levels have been configured in the system. This method is used to configure the SFC ActiveX control as well as to dictate whether the unit procedure methods should be used in this control. A Short return value indicates if two recipe levels have been defined. The value can be interpreted as follows:

0 = Three recipe levels defined
1 = Two recipe levels defined

Syntax

```
ReturnValue = BatchSFCVar.MiscGet2Levels ()
```

MiscGetEnumName()

This method queries the name of the enumeration based on the enumeration set name and the enumeration value. For example, if the enumeration set name is Mode, the value 1 is defined as Manual, 4 as Semi-Auto, and 20 as Automatic in the Process Model Editor for Enumerations, then MiscGetEnumName("Mode", 4) would return Semi-Auto. This method is used to determine enumeration names for both the EditPhaseParam list and the PhaseParam list. A String return value (16 characters) contains the enumeration name.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetEnumName (SetName, EnumVal)
```

Parameters

SetName
Data Type: String
Enumeration set name

EnumVal
Data Type: Long
Enumeration value

MiscGetEnumNameByRow()

This method queries the name of the enumeration based on the enumeration set name and the enumeration value. For example, if the enumeration set name is Mode, the value 1 is defined as Manual, 4 as Semi-Auto, and 20 as Automatic in the Process Model Editor for Enumerations, then MiscGetEnumNameByRow("Mode", 2) would return Semi-Auto since that is the second enumeration value in the set. This method is used to determine enumeration names for both the EditPhaseParam list and the PhaseParam list. A String return value (16 characters) contains the enumeration name.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetEnumNameByRow( SetName, Row)
```

Parameters

SetName
Data Type: String
Enumeration set name

Row
Data Type: Short
Row index of enumeration set

MiscGetEnumValue()

This method queries the value of the enumeration based on the enumeration set name and the enumeration name. For example, if the enumeration set name is Mode, the value 1 is defined as Manual, 4 as Semi-Auto, and 20 as Automatic in the Process Model Editor for Enumerations, then MiscGetEnumValue("Mode", "Semi-Auto") would return 4. This method is used to determine enumeration values for both the EditPhaseParam list and the PhaseParam list. A long integer return value contains the enumeration value.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetEnumValue(SetName, EnumName)
```

Parameters

SetName
Data Type: String
Enumeration set name

EnumName
Data Type: String
Enumeration name string

MiscGetMessage()

This method is used internally by Batch Manager to retrieve batch message catalog information.

Important: Do not use this method within applications.

MiscGetNumEnums()

This method queries the number of enumeration values in the specified enumeration set based on the enumeration set name. For example, if the enumeration set name is Mode, the value 1 is defined as Manual, 4 as Semi-Auto, and 20 as Automatic in the Process Model Editor for Enumerations, then MiscGetNumEnums("Mode") would return 3. This method is used to determine number of members in an enumeration set. A long integer return value contains the number of members in the enumeration set.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetNumEnums (SetName)
```

Parameters

SetName
Data Type: String
Enumeration set name

MiscGetRecipeDefBatchSize()

This method queries the default batch size defined for a specific recipe. A Long return value provides the size. A value of –1 indicates an error.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetRecipeDefBatchSize
(RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe ID

MiscGetRecipeMaxBatchSize()

This method queries the maximum batch size defined for a specific recipe. A Long return value provides the size. A value of –1 indicates an error.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetRecipeMaxBatchSize ( RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe ID

MiscGetRecipeMinBatchSize()

This method queries the minimum batch size defined for a specific recipe. A Long return value provides the size. A value of –1 indicates an error.

Syntax

```
ReturnValue = OcxBatchVar.MiscGetRecipeMinBatchSize ( RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe ID

PhaseAbortPhase()

This method aborts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAbortPhase (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseAbortPhaseWC()

This method aborts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseAbortPhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAbortPhaseWC ( DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

PhaseAckDocument ()

This method acknowledges the document associated with the currently selected phase in the PhasePhase list. This method should be called only if an acknowledge is required. A required acknowledgement can be determined by evaluating the value returned from the PhasePhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAckDocument (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseAckDocumentWC()

This method acknowledges the document associated with the currently selected phase in the PhasePhase list. This method should be called only if an acknowledge is required. A required acknowledgement can be determined by evaluating the value returned from the PhasePhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseAckDocumentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAckDocumentWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

PhaseAckPhase()

This method acknowledges the currently selected phase in the PhasePhase list. This method should only be called if an acknowledge is required. A required acknowledgement can be determined by evaluating the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAckPhase (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseAckPhaseWC()

This method acknowledges the currently selected phase in the PhasePhase list. This method should only be called if an acknowledge is required. A required acknowledgement can be determined by evaluating the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseAckPhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAckPhaseWC ( DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

PhaseEditParameter()

This method changes a parameter value for the currently selected parameter in the PhaseParam list. This method should only be called if the PhaseParamGetEditType method returns a non-zero value. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

Syntax

```
ReturnCode = OcxBatchVar.PhaseEditParameter (Value, DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

Value
Data Type: String
New parameter value

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseEditParameterWC

This method changes a parameter value for the currently selected parameter in the PhaseParam list. This method should only be called if the PhaseParamGetEditType method returns a non-zero value. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseEditParameterWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseEditParameterWC (Value, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Value
Data Type: String
New parameter value

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

PhaseEnterComment()

This method enters a comment for the currently selected phase in the PhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseEnterComment (Comment, DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

Comment
Data Type: String
Comment text

DoneBy
Data Type: String

Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseEnterCommentWC

This method enters a comment for the currently selected phase in the PhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseEnterCommentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseEnterCommentWC (Comment, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Comment
Data Type: String
Comment text

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

PhaseHoldPhase()

This method holds the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Run. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

Syntax

```
ReturnCode = OcxBatchVar.PhaseHoldPhase (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseHoldPhaseWC()

This method holds the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Run. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

NOTE: You must use the PhaseHoldPhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseHoldPhaseWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String

Check by Comment

PhaseIlockGetIlock()

This method queries the interlock based on its index in the Phaselock list. A String return value (84 characters) provides the interlock tag.

Syntax

```
ReturnValue = OcxBatchVar.PhaseIlockGetIlock (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseIlockGetItem()

This method queries all information of an interlock based on its index in the Phaselock list. A String return value (166 characters) contains the interlock tag and the value fields separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.PhaseIlockGetItem (Row)
```

PhaseIlockGetNumItems()

This method queries the number of interlocks in the Phaselock list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.PhaseIlockGetNumItems ()
```

PhaseIlockGetSelected()

This method queries the index of the currently selected interlock in the Phaselock list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.PhaseIlockGetSelected ()
```

PhaseIlockGetStatus()

This method queries the interlock value on its index in the Phaselock list. A String return value (80 characters) provides the interlock value.

Syntax

```
ReturnValue = OcxBatchVar.PhaseIlockGetStatus (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseIlockSetSelected()

This method selects a specific interlock based on its index in the Phaselock list. There are no actions permitted on individual batch messages. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.PhaseIlockSetSelected (Row)
```

Parameters

Row
Data Type: Short

Index of item to select

PhaseInstrGetInstr()

This method queries the instructions associated with the currently selected phase in the PhasePhase list. A String return value (unlimited number of characters) provides a multi-line instruction.

Syntax

```
ReturnValue = OcxBatchVar.PhaseInstrGetInstr ()
```

PhaseParamGetDataClass()

This method queries the data class (as an integer) of a parameter based on its index in the PhaseParam list. A Short return value provides the data class. The value can be interpreted as follows:

0 = Analog
1 = Discrete
2 = String
3 = Enumeration

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetDataClass (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetDescription()

This method queries the description of a parameter based on its index in the PhaseParam list. A String return value (120 characters) provides the description of the parameter.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetDescription ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetEditType()

This method queries the edit type of a parameter based on its index in the PhaseParam list. A Short return value provides the edit type. The value can be interpreted as follows:

1 = Numeric edit
2 = Alphanumeric edit
3 = Numeric edit required
4 = Alphanumeric edit required

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetEditType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetExt()

This method queries the extension of a parameter based on its index in the PhaseParam list. A Short return value provides the extension. The value can be interpreted as follows:

1 = Material ID
2 = Material Name
3 = Target
4 = Actual
5 = High deviation
6 = Low deviation
7 = Lot code
8 = Preact
9 = High limit
10 = Low limit

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetExt (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetExtValue()

This method queries the value of a parameter extension based on its index in the PhaseParam list. A String return value (80 characters) provides the parameter extension value.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetExtValue (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetItem()

This method queries all information of a parameter based on its index in the PhaseParam list. A String return value (180 characters) contains the parameter name, parameter type, extension, value, unit or measure, edit status, data class, and enumeration set name fields. Eachfield is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetNumItems()

This method queries the number of parameters in the PhaseParam list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetNumItems ()
```

PhaseParamGetParam()

This method queries the name of the parameter based on its index in the PhaseParam list. A String return value (16 characters) provides the name of the parameter.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetParam (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetSelected()

This method queries the index of the currently selected parameter in the PhaseParam list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetSelected ()
```

PhaseParamGetSet()

This method queries the set name of the enumeration of a parameter based on its index in the PhaseParam list. This method should be called if the PhaseParamGetDataClass function indicates that the parameter is an enumeration. A String return value (16 characters) contains the enumeration set name.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetSet (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetType()

This method queries the type of the parameter based on its index in the PhaseParam list. A Short return value provides the type of the parameter. The value can be interpreted as follows:

0 = Input
9 = Process variables
12 = Output

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamGetUom()

This method queries the engineering units of a parameter based on its index in the PhaseParam list. This method should be called if the PhaseParamGetType function indicates that the parameter is a process variable. A String return value (16 characters) contains the engineering units value.

Syntax

```
ReturnValue = OcxBatchVar.PhaseParamGetUom (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseParamSetSelected()

This method selects a specific parameter based on its index in the PhaseParam list. This method is required to change a parameter value.

Syntax

```
OcxBatchVar.PhaseParamSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetCtrlButtonLabel1()

This method queries the label of the first control button of the currently selected phase in the PhasePhase list. A String return value (8 characters) provides the control button label.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetCtrlButtonLabel1 (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetCtrlButtonLabel2()

This method queries the label of the second control button of the currently selected phase in the PhasePhase list. A String return value (8 characters) provides the control button label.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetCtrlButtonLabel2 (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetDescription()

This method queries the description of a phase based on its index in the PhasePhase list. A String return value (120 characters) provides the description of the phase.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetDescription ( Row )
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetDocPath()

This method queries the recipe defined document path of the currently selected phase in the PhasePhase list. A String return value (128 characters) provides the document path.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetDocPath (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetEditMask()

This method queries the required actions of a phase based on its index in the PhasePhase list. The Long return value is a bit mask used to determine what actions are required. If the bit is set, the action is required. If it is clear, the action is not required. The value can be interpreted as follows.

Bit (Value)	Description
0 (1)	Ack Entry required
1 (2)	Ack Entry DoneBy required
2 (4)	Ack Entry DoneBy required
3 (8)	Ack Exit required
4 (16)	Ack Exit DoneBy required
5 (32)	Ack Exit CheckBy required
6 (64)	Comment required
7 (128)	Enable Hold
8 (256)	Enable Restart
9 (512)	Enable Abort
10 (1024)	Enable Start
11 (2048)	Operator Action required
12 (4096)	Enable Control Button 1
13 (8192)	Enable Control Button 2
14 (16384)	Ack Partial Add required
15 (32768)	Continue Mode enabled
16 (65536)	View Document On Entry required
17 (131072)	View Document On Exit required
18 (262144)	Ack Document On Entry required
19 (524288)	Ack Document On Exit required
20 (1048576)	Continue Mode can be enabled
21 (2097152)	Manual Phase – Ack On Exit always enabled
(7)	General Ack entry mask

Bit (Value)	Description
(56)	General Ack exit mask
(16447)	General Ack mask

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetEditMask (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetEquipment()

This method queries the name of the unit or connection of a phase based on its index in the PhasePhase list. A String return value (16 characters) provides the name of the unit or connection.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetEquipment (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetItem()

This method queries all information of a phase based on its index in the PhasePhase list. A String return value (88 characters) contains the equipment name, unit procedure name, operation name, phase name, status and action required fields each separated by two spaces. The operation name is included in the string value regardless of the number of recipe levels defined.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetLabel()

This method queries the label of a phase based on its index in the PhasePhase list. A String return value (8 characters) provides the label.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetLabel (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetNumItems()

This method queries the number of phases in the PhasePhase list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetNumItems ()
```

PhasePhaseGetOperation()

This method queries the operation name of a phase based on its index in the PhasePhase list. A String return value (16 characters) provides the name of the operation.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetOperation (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetOperMsg()

This method queries the operator message (as an integer) of a phase based on its index in the PhasePhase list. A Long return value provides the operator message. The value can be interpreted as follows:

0 = None
1 = Waiting for manual start
2 = Waiting for entry acknowledge
3 = Waiting for exit acknowledge
4 = Waiting for required edits
5 = Waiting for comment
6 = Waiting for phase status
7 = Waiting for phase to start
8 = Partial add—waiting for acknowledge
9 = Waiting for document viewing
10 = Waiting for document acknowledge
11 = Writing phase parameters
12 = Reading phase parameters

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetOperMsg (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetOperMsgStr()

This method queries the operator message (as a string) of a phase based on its index in the PhasePhase list. A String return value (110 characters) provides the operator message.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetOperMsgStr (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetPhase()

This method queries the name of a phase based on its index in the PhasePhase list. A String return value (16 characters) provides the name of the phase.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetPhase (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetSelected()

This method queries the index of the currently selected phase in the PhasePhase list. A Short return value provides the selected item's index.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetSelected ()
```

PhasePhaseGetStatus()

This method queries the status of a phase based on its index in the PhasePhase list. A Long return value provides the status. The value can be interpreted as follows:

1 = Ready
2 = Wait
3 = Run
4 = Done
5 = Interlocked
6 = Aborted
7 = Held
9 = Continue

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetStatus (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseGetUnitProcedure()

This method queries the unit procedure name in which a phase is defined based on its index in the PhasePhase list. A String return value (16 characters) provides the unit procedure name.

Syntax

```
ReturnValue = OcxBatchVar.PhasePhaseGetUnitProcedure ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhasePhaseSetSelected()

This method selects a specific phase based on its index in the PhasePhase list. This method is required to perform actions on an active phase (such as Start and Hold) and also to populate the dependent phase parameter list.

Syntax

```
OcxBatchVar.PhasePhaseSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

PhaseRestartPhase()

This method restarts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseRestartPhase (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseRestartPhaseWC()

This method restarts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Held. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseRestartPhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseRestartPhaseWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

PhaseSetCtrlButton1()

This method performs the first control button action for the currently selected phase in the PhasePhase list. This method should be called only if the Enable Control Button 1 bit is set in the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseSetCtrlButton1 (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseSetCtrlButton1WC()

This method performs the first control button action for the currently selected phase in the PhasePhase list. This method should be called only if the Enable Control Button 1 bit is set in the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseSetCtrlButton1WC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd,DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

NOTE: You must use the PhaseSetCtrlButton1WC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

PhaseSetCtrlButton2()

This method performs the second control button action for the currently selected phase in the PhasePhase list. This method should only be called if the Enable Control Button 2 bit is set in the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseSetCtrlButton2 (DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseSetCtrlButton2WC()

This method performs the second control button action for the currently selected phase in the PhasePhase list. This method should only be called if the Enable Control Button 2 bit is set in the value returned from the PhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseSetCtrlButton2WC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseSetCtrlButton2WC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

PhaseStartPhase()

This method starts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseStartPhase (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

PhaseStartPhaseWC()

This method starts the currently selected phase in the PhasePhase list. This method should be called only if the phase status is Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the PhaseStartPhaseWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseStartPhaseWC ( DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

PhaseViewDocument()

This method enables the listing of the document assigned to the currently selected phase in the PhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseViewDocument (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String

Done by password

CheckByPswd
Data Type: String
Check by password

PhaseViewDocumentWC()

This method enables the listing of the document assigned to the currently selected phase in the PhasePhase list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*".

NOTE: You must use the PhaseViewDocumentWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.PhaseViewDocumentWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

QuestAnswerQuest()

This method answers the currently selected question in the QuestionQuest list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.QuestAnswerQuest (Answer, DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

Answer
Data Type: Short
Answer to question:
0 = No
1 = Yes

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

QuestAnswerQuestWC()

This method answers the currently selected question in the QuestionQuest list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the QuestAnswerQuestWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.QuestAnswerQuestWC (Answer, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Answer
Data Type: Short
Answer to question:
0 = No
1 = Yes

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

QuestionQuestGetItem()

This method queries all information of a question based on its index in the QuestionQuest list. A String return value (40 characters) contains the question text. This is a single-field list. Therefore, the value returned using this method is the same as the value returned by the QuestionQuestGetQuestion method.

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

QuestionQuestGetNumItems()

This method queries the number of questions in the QuestionQuest list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetNumItems ()
```

QuestionQuestGetQuestion()

This method queries the text of a question based on its index in the QuestionQuest list. A String return value (40 characters) provides the question text.

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetQuestion ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

QuestionQuestGetSecurity()

This method queries the security setting of a question based on its index in the QuestionQuest list. A Long return value provides the security required for the question. The value can be interpreted as follows:

0 = None
1 = DoneBy
2 = CheckBy

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetSecurity ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

QuestionQuestGetSelected()

This method queries the index of the currently selected question in the QuestionQuest list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetSelected ()
```

QuestionQuestGetType()

This method queries the type of a question based on its index in the QuestionQuest list. A Long return value provides the question type. The value can be interpreted as follows:

0 = Yes or No (loop)
1 = Yes (transition)

Syntax

```
ReturnValue = OcxBatchVar.QuestionQuestGetType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

QuestionQuestSetSelected()

This method selects a specific question based on its index in the QuestionQuest list. This method is required to answer a question.

Syntax

```
OcxBatchVar.QuestionQuestSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to select

RecipeSetCLBFocus()

This is a required method for using the phase processing, answer questions, and equipment selection functional areas and establishing a batch-centric view in the respective lists. These functional areas list information pertaining to the batch that is defined by the CLB parameter. This method can be called only when the CLB defined by the parameter exists in the BatchSched List and has a status of Run, Held, or Aborting. If the batch status is other than Run, Held, or Aborting, the parameter to this method should be empty (//). You can use the event BatchSchedFocusState for easy batch-centric focus setting. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.RecipeSetCLBFocus (CLB)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch ID data separated by forward slash (/) characters

RecipeSetUnitFocus()

This method creates a unit-centric view of the phase processing, answer questions, and equipmentselection functional areas and establishes a batch-centric view in the respective lists. These functional areas list information pertaining to the batch that is running in the unit parameter. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.RecipeSetUnitFocus (Unit)
```

Unit
Data Type: String

Unit name

SaveRecipeRecipeExists()

This method determines if the specified recipe ID used to save a control recipe is valid. A Short return value of 1 indicates success. A Short return value of 0 indicates failure.

Syntax

```
ReturnValue = OcxBatchVar.SaveRecipeExists (RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe ID

SaveRecipeSave()

This method saves the control recipe information for a batch as a recipe in the recipe database. This method should be called only if the batch has a status of Done. If a different user is provided for the DoneBy and Author parameters, the DoneBy user is used for the Author. If this happens, the SaveRecipeAuthorChanged event is called. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.SaveRecipeSave (CLB, RecipeId, Author, OptMask,
Comment, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch data separated by forward slash (/) characters

RecipeId
Data Type: String
Recipe ID

Author
Data Type: String
Recipe author

OptMask
Data Type: Long
Flag to set whether approvals and/or equipment is retained:
0 = Retain approvals
1 = Retain equipment

Comment
Data Type: String
Recipe version comments

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

SaveRecipeSaveWC()

This method saves the control recipe information for a batch as a recipe in the recipe database. This method should be called only if the batch has a status of Done. If a different user is provided for the DoneBy and Author parameters, the DoneBy user is used for the Author. If this happens, the SaveRecipeAuthorChanged event is called. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the SaveRecipeSaveWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.SaveRecipeSaveWC(CLB, RecipeId, Author, OptMask,
Comment, DoneBy, CheckBy, DoneByPswd, CheckByPswd,
DoneByComment, CheckByComment)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch data separated by forward slash (/) characters

RecipeId
Data Type: String
Recipe ID

Author
Data Type: String
Recipe author

OptMask
Data Type: Long
Flag to set whether approvals and/or equipment is retained:
0 = Retain approvals
1 = Retain equipment

Comment
Data Type: String
Recipe version comments

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

ScheduleAddBatch()

This method adds a batch to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleAddBatchWC (Campaign, Lot, Batch, Recipe,
Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleAddBatchWC()

This method adds a batch to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleAddBatchWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleAddBatchWC (Campaign, Lot, Batch, Recipe,
Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment,
CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String

Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

ScheduleChangeBatch()

This method changes the information for the currently selected batch in the ScheduleSched list. The batch status changes to Open. This method can be called only if the batch status is Open or Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*

Syntax

```
ReturnCode = OcxBatchVar.ScheduleChangeBatch ( Campaign, Lot, Batch, Recipe,
Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleChangeBatchWC()

This method changes the information for the currently selected batch in the ScheduleSched list. The batch status changes to Open. This method can be called only if the batch status is Open or Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleChangeBatchWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleChangeBatchWC ( Campaign, Lot, Batch, Recipe,
Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment,
CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleCleanup()

This method removes all batches with a status of Aborted or Done from the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleCleanup (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleCleanupWC()

This method removes all batches with a status of Aborted or Done from the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleCleanupWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleCleanupWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleClearAccessSecurity()

This method cancels all batch scheduling security set by the ScheduleSetAccessSecurity method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleClearAccessSecurity ()
```

ScheduleDeleteBatch()

This method deletes the currently selected batch from the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleDeleteBatch (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleDeleteBatchWC()

This method deletes the currently selected batch from the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleDeleteBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

NOTE: You must use the ScheduleDeleteBatchWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

ScheduleGetExecInOrder()

This method queries the status of the Batch Execute in Order mode. A Short return value provides the status. The value can be interpreted as follows:

0 = Run in order disabled
1 = Run in order enabled

Syntax

```
ReturnValue = OcxBatchVar.ScheduleGetExecInOrder ()
```

ScheduleHasValidAccessSecurity()

This method determines if the operator passed the security requirements for performing batch scheduling functions. This method can be called from within the OnSecurityPending event if the security operation is complete (State = 0). A Short return value provides the result. The value can be interpreted as follows:

0 = User failed security
1 = User passed security

Syntax

```
ReturnValue = OcxBatchVar.ScheduleHasValidAccessSecurity ()
```

ScheduleInitAll()

This method initializes all batches with a status of Open in the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Value*".

Syntax

```
ReturnCode = OcxBatchVar.ScheduleInitAll (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleInitAllWC()

This method initializes all batches with a status of Open in the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleInitAllWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleInitAllWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleInitBatch()

This method initializes the currently selected batch in the ScheduleSched list. This method should be called only if the batch status is Open. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleInitBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleInitBatchWC()

This method initializes the currently selected batch in the ScheduleSched list. This method should be called only if the batch status is Open. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleInitBatchWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

NOTE: You must use the ScheduleInitBatchWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

ScheduleMoveBatchAfter()

This method moves the currently selected batch in the ScheduleSched list to the position after the batch identified in the Campaign, Lot, and Batch parameters. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchAfter ( Campaign, Lot, Batch, DoneBy,
CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMoveBatchAfterWC()

This method moves the currently selected batch in the ScheduleSched list to the position after the batch identified in the Campaign, Lot, and Batch parameters. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleMoveBatchAfterWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchAfterWC ( Campaign, Lot, Batch,
DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleMoveBatchBefore()

This method moves the currently selected batch in the ScheduleSched list to the position before the batch identified in the Campaign, Lot, and Batch parameters. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchBefore ( Campaign, Lot, Batch,
DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMoveBatchBeforeWC()

This method moves the currently selected batch in the ScheduleSched list to the position before the batch identified in the Campaign, Lot, and Batch parameters. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleMoveBatchBeforeWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchBeforeWC ( Campaign, Lot, Batch,
DoneBy, CheckBy, DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleMoveBatchDown()

This method moves the currently selected batch in the ScheduleSched list down by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchDown ( DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMoveBatchDownWC()

This method moves the currently selected batch in the ScheduleSched list down by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values*"..

NOTE: You must use the ScheduleMoveBatchDownWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchDownWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleMoveBatchUp()

This method moves the currently selected batch in the ScheduleSched list up by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchUp (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMoveBatchUpWC()

This method moves the currently selected batch in the ScheduleSched list up by one position. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleMoveBatchUpWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMoveBatchUpWC (DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleMultiAddBatch()

This method adds multiple batches to the ScheduleSched list by dividing the Batch Size argument by the default batch size defined in the recipe. To determine whether to call this method or to call the ScheduleAddBatch method, use the MiscGetRecipeMaxBatchSize method and compare the value to the user entered batch size. If thebatch size entered by the operator is greater than the maximum batch size, this method can be used to add multiple batches. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMultiAddBatch ( Campaign, Lot, Batch, Recipe,
Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMultiAddBatchWC()

This method adds multiple batches to the ScheduleSched list by dividing the Batch Size argument by the default batch size defined in the recipe. To determine whether to call this method or to call the ScheduleAddBatch method, use the MiscGetRecipeMaxBatchSize method and compare the value to the user entered batch size. If thebatch size entered by the operator is greater than the maximum batch size, this method can be used to add multiple batches. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleMultiAddBatchWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMultiAddBatchWC ( Campaign, Lot, Batch,
Recipe, Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd,
DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

RecipeId
Data Type: String
Recipe ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

ScheduleAddFormulaBatch()

This method adds a formula specific batch to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleAddFormulaBatch (Campaign, Lot, Batch, Recipe,
Formula, Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleAddFormulaBatchWC()

This method adds a formula specific batch to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleAddFormulaBatchWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleAddFormulaBatchWC ( Campaign, Lot, Batch,
Recipe, Formula,Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd,
DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleMultiAddFormulaBatch()

This method adds multiple formula specific batches to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMultiAddFormulaBatchWC ( Campaign, Lot, Batch,
Recipe, Formula,Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleMultiAddFormulaBatchWC()

This method adds multiple formula specific batches to the ScheduleSched list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleMultiAddFormulaBatchWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleMultiAddFormulaBatchWC ( Campaign, Lot, Batch,
Recipe, Formula,Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd,
DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String

Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleChangeFormulaBatch()

This method changes the information for the currently selected formula specific batch in the ScheduleSched list. The formula batch status changes to Open. This method can be called only if the formula batch status is Open or Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see "Error Return Values".

Syntax

```
ReturnCode = OcxBatchVar.ScheduleChangeFormulaBatch( Campaign, Lot, Batch,
Recipe, Formula,Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String
Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleChangeFormulaBatchWC()

This method changes the information for the currently selected formula specific batch in the ScheduleSched list. The formula batch status changes to Open. This method can be called only if the formula batch status is Open or Ready. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleChangeFormulaBatchWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleChangeFormulaBatchWC ( Campaign, Lot, Batch,
Recipe, Formula,Size, Train, Mode, DoneBy, CheckBy, DoneByPswd, CheckByPswd,
DoneByComment, CheckByComment)
```

Parameters

Campaign
Data Type: String
Campaign ID

Lot
Data Type: String
Lot ID

Batch
Data Type: String
Batch ID

Recipe
Data Type: String
Recipe ID

Formula
Data Type: String
Formula ID

Size
Data Type: Long
Batch Quantity

Train
Data Type: String

Name of train

Mode
Data Type: Short
Batch Mode:
0 = Automatic
1 = Semi-Automatic
2 = Manual

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleUpdateFormulaList()

This method forces the ScheduleFormula list to be updated from the Recipe system with all available formulas. The return code is a Short integer value that can be used for error handling. For more information on the error codes returned, see *"Error Return Values" on page 232*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleUpdateFormulaList (Recipe)
```

Parameters

Recipe
Data Type: Short
Recipe ID

ScheduleFormulaGetItem()

This method queries all information of a formula based on its index in the ScheduleFormula list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleFormulaGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleFormulaGetFormula()

This method queries the name of a formula based on its index in the ScheduleFormula list. A String return value (128 characters) provides the formula name.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleFormulaGetFormula( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleFormulaGetSelected()

This method queries the index of the currently selected formula in the ScheduleFormula list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleFormulaGetSelected ()
```

ScheduleFormulaSetSelected()

This method selects a specific formula based on its index in the ScheduleFormula list. There are no actions permitted on individual formula. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ScheduleFormulaSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleFormulaGetNumItems()

This method queries the number of recipes in the ScheduleFormula list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleFormulaGetNumItems ()
```

ScheduleSchedGetFormula()

This method queries the formula identification of a batch based on its index in the ScheduleSched list. A String return value (128 characters) provides the Formula name of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetFormula(Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleRecipeGetItem()

This method queries all information of a recipe based on its index in the ScheduleRecipe list. A String return value (182 characters) contains the recipe ID, recipe name, recipe state, and recipe type fields. Each field is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetItem (Row)
```

Parameters

Row
Data Type: Short

Index of item to retrieve

ScheduleRecipeGetNumItems()

This method queries the number of recipes in the ScheduleRecipe list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetNumItems ()
```

ScheduleRecipeGetRecipeId()

This method queries the identification of a recipe based on its index in the ScheduleRecipe list. A String return value (16 characters) provides the recipe ID.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetRecipeId ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleRecipeGetRecipeName()

This method queries the name of a recipe based on its index in the ScheduleRecipe list. A String return value (16 characters) provides the recipe name.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetRecipeName ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleRecipeGetRecipeState()

This method queries the state of a recipe based on its index in the ScheduleRecipe list. A String return value (16 characters) provides the recipe state.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetRecipeState (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleRecipeGetRecipeType()

This method queries the type of a recipe based on its index in the ScheduleRecipe list. A String return value (16 characters) provides the recipe type.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetRecipeType ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleRecipeGetSelected()

This method queries the index of the currently selected recipe in the ScheduleRecipe list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleRecipeGetSelected ()
```

ScheduleRecipeSetSelected()

This method selects a specific recipe based on its index in the ScheduleRecipe list. There are no actions permitted on individual recipes. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ScheduleRecipeSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetBatch()

This method queries the batch identification of a batch based on its index in the ScheduleSched list. A String return value (16 characters) provides the batch ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetBatch (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetCampaign()

This method queries the campaign identification of a batch based on its index in the ScheduleSched list. A String return value (16 characters) provides the campaign ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetCampaign ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetItem()

This method queries all information of a batch based on its index in the ScheduleSched list. A String return value (127 characters) contains the campaign ID, lot ID, batch ID, recipe ID, batch size, train name, batch mode, and batch status fields. Each field is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetLot()

This method queries the lot identification of a batch based on its index in the ScheduleSched list. A String return value (16 characters) provides the lot ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetLot (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetMode()

This method queries the mode of a batch based on its index in the ScheduleSched list. A Short return value provides the mode of the batch. The value can be interpreted as follows:

0 = Automatic
1 = Semi-Automatic
2 = Manual

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetMode (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetNumItems()

This method queries the number of batches in the ScheduleSched list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetNumItems ()
```

ScheduleSchedGetRecipeId()

This method queries the recipe identification of a batch based on its index in the ScheduleSched list. A String return value (16 characters) provides the recipe ID of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetRecipeId ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetSelected()

This method queries the index of the currently selected batch in the ScheduleSched list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetSelected ()
```

ScheduleSchedGetSize()

This method queries the size of a batch based on its index in the ScheduleSched list. A Long return value provides the size of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetSize (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedGetTrain()

This method queries the train name of a batch based on its index in the ScheduleSched list. A String return value (16 characters) provides the train name of the batch.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleSchedGetTrain (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSchedSetSelected()

This method selects a specific batch based on its index in the ScheduleSched list. This method is required to perform actions on a particular batch (such as initialize or delete).

Syntax

```
OcxBatchVar.ScheduleSchedSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleSetAccessSecurity()

This method lists a modeless security dialog box if the Access Schedule security function is defined in the Security Editor. After a user enters his security information, the OnSecurityPending event is called. In this event, a batch scheduling application can determine if the user is allowed to perform the batch scheduling functions. Using this technique, the user does not need to be prompted for security every time he wants to perform a scheduling function. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 232*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleSetAccessSecurity ()
```

ScheduleSetExecInOrder()

This method enables or disables the Batch Execute in Order mode. This affects all batch schedule based applications. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 232*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleSetExecInOrder ( State, DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

State
Data Type: Short
Flag to enable and disable execute in order mode:

0 = Enable execute in order
1 = Disable execute in order

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ScheduleSetExecInOrderWC()

This method enables or disables the Batch Execute in Order mode. This affects all batch schedule based applications. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ScheduleSetExecInOrderWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleSetExecInOrderWC (State, DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

State
Data Type: Short
Flag to enable and disable execute in order mode:

0 = Enable execute in order
1 = Disable execute in order

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String

Check by password

DoneByComment
Data Type: String
Check by Comment

CheckByComment
Data Type: String
Check by Comment

ScheduleStateGetItem()

This method queries all information of a state based on its index in the ScheduleState list. A String return value (16 characters) contains the state name. This is a single-field list. Therefore, the value returned using this method is the same as the value returned by the ScheduleStateGetState method.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleStateGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleStateGetNumItems()

This method queries the number of states in the ScheduleState list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleStateGetNumItems ()
```

ScheduleStateGetSelected()

This method queries the index of the currently selected state in the ScheduleState list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleStateGetSelected ()
```

ScheduleStateGetState()

This method queries the name of a state based on its index in the ScheduleState list. A String return value (16 characters) provides the state name.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleStateGetState (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleStateSetSelected()

This method selects a specific state based on its index in the ScheduleState list. There are no actions permitted on individual states. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ScheduleStateSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleTrainGetItem()

This method queries all information of a train based on its index in the ScheduleTrain list. A String return value (16 characters) contains the train name. This is a single-field list. Therefore, the value returned using this method is the same as the value returned by the ScheduleTrainGetTrain method.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTrainGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleTrainGetNumItems()

This method queries the number of trains in the ScheduleTrain list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTrainGetNumItems ()
```

ScheduleTrainGetSelected()

This method queries the index of the currently selected train in the ScheduleTrain list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTrainGetSelected ()
```

ScheduleTrainGetTrain()

This method queries the name of a train based on its index in the ScheduleTrain list. A String return value (16 characters) provides the train name.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTrainGetTrain (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleTrainSetSelected()

This method selects a specific train based on its index in the ScheduleTrain list. There are no actions permitted on individual trains. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ScheduleTrainSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to select

ScheduleTypeGetItem()

This method queries all information of a type based on its index in the ScheduleType list. A String return value (16 characters) contains the type name. This is a single- field list. Therefore, the value returned using this method is the same as the value returned by the ScheduleTypeGetType method.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTypeGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleTypeGetNumItems()

This method queries the number of types in the ScheduleType list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTypeGetNumItems ()
```

ScheduleTypeGetSelected()

This method queries the index of the currently selected type in the ScheduleType list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTypeGetSelected ()
```

ScheduleTypeGetType()

This method queries the name of a type based on its index in the ScheduleType list. A String return value (16 characters) provides the type name.

Syntax

```
ReturnValue = OcxBatchVar.ScheduleTypeGetType (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleTypeSetSelected()

This method selects a specific type based on its index in the ScheduleType list. There are no actions permitted on individual types. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ScheduleTypeSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ScheduleUpdateRecipeList()

This method forces the ScheduleRecipe list to be updated from the Recipe Database with all approved recipes. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleUpdateRecipeList ()
```

ScheduleUpdateStateList()

This method forces the ScheduleState list to be updated from the Recipe Database with all defined states. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleUpdateStateList ()
```

ScheduleUpdateTrainList()

This method forces the ScheduleTrain list to be updated from the Process Model database (ModelDB) with all defined trains. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleUpdateTrainList ()
```

ScheduleUpdateTypeList()

This method forces the ScheduleType list to be updated from the Recipe Database with all defined types. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ScheduleUpdateTypeList ()
```

SelectEquipGetEquipment()

This method queries the name of a unit based on its index in the SelectEquip list. A String return value (16 characters) provides the unit name.

Syntax

```
ReturnValue = OcxBatchVar.SelectEquipGetEquipment (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectEquipGetItem()

This method queries all information of a unit based on its index in the SelectEquip list. A String return value (34 characters) contains the unit name and equipment status fields. Each field is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.SelectEquipGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectEquipGetNumItems()

This method queries the number of units in the SelectEquip list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.SelectEquipGetNumItems ()
```

SelectEquipGetSelected()

This method queries the index of the currently selected unit in the SelectEquip list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.SelectEquipGetSelected()
```

SelectEquipGetStatus()

This method queries the equipment status of a unit based on its index in the SelectEquip list. A String return value (16 characters) provides the equipment status of the unit.

Syntax

```
ReturnValue = OcxBatchVar.SelectEquipGetStatus (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectEquipSetSelected()

This method selects a specific unit based on its index in the SelectEquip list. This method is required along with the SelectInstSetSelected method to allocate a specific unit.

Syntax

```
OcxBatchVar.SelectEquipSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectInstGetInstance()

This method queries the name of a process instance based on its index in the SelectInst list. A String return value (16 characters) provides the instance name.

Syntax

```
ReturnValue = OcxBatchVar.SelectInstGetInstance (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectInstGetItem()

This method queries all information of a process instance based on its index in the SelectInst list. A String return value (16 characters) contains the instance name. This is a single-field list. Therefore, the value returned using this method is the same as the value returned by the SelectInstGetInstance method.

Syntax

```
ReturnValue = OcxBatchVar.SelectInstGetItem (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectInstGetNumItems()

This method queries the number of process instances in the SelectInst list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.SelectInstGetNumItems ()
```

SelectInstGetSelected()

This method queries the index of the currently selected process instance in the SelectInst list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.SelectInstGetSelected ()
```

SelectInstSetSelected()

This method selects a specific process instance based on its index in the SelectInst list. This method is required to populate the dependent equipment selection list.

Syntax

```
OcxBatchVar.SelectInstSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

SelectSelectEquip()

This method allocates the currently selected unit in the SelectEquip list to the currently selected process instance in the SelectInst list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.SelectSelectEquip (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

SelectSelectEquipWC()

This method allocates the currently selected unit in the SelectEquip list to the currently selected process instance in the SelectInst list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the SelectSelectEquipWC() method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.SelectSelectEquipWC(DoneBy, CheckBy, DoneByPswd,
CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

PhaseAckDocument ()

This method acknowledges the document associated with the currently selected phase in the PhasePhase list. This method should be called only if an acknowledge is required. A required acknowledgement can be determined by evaluating the value returned from the PhasePhaseGetEditMask method. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.PhaseAckDocument (DoneBy, CheckBy, DoneByPswd,
CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

Term()

This method terminates communication with the Batch Manager. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.Term ()
```

ViewTransitionExpGetExp()

This method queries the transition logic expression associated with the currently selected transition in the ViewTransitionTrans list. A String return value (unlimited number of characters) provides the multi-line transition expression.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionExpGetExp ()
```

ViewTransitionForceTransition()

This method forces the state to True of the currently selected transition in the ViewTransitionTrans list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ViewTransitionForceTransition (DoneBy, CheckBy,
DoneByPswd, CheckByPswd)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

ViewTransitionForceTransitionWC()

This method forces the state to True of the currently selected transition in the ViewTransitionTrans list. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

NOTE: You must use the ViewTransitionForceTransitionWC () method when the DoneByComment and CheckByComment is made mandatory in the System Parameter.

Syntax

```
ReturnCode = OcxBatchVar.ViewTransitionForceTransitionWC (DoneBy, CheckBy,
DoneByPswd, CheckByPswd, DoneByComment, CheckByComment)
```

Parameters

DoneBy
Data Type: String
Done by user ID

CheckBy
Data Type: String
Check by user ID

DoneByPswd
Data Type: String
Done by password

CheckByPswd
Data Type: String
Check by password

DoneByComment
Data Type: String
Done by comment

CheckByComment
Data Type: String
Check by comment

ViewTransitionSetCLBFocus()

This is a required method for using the view active transitions functional area and establishes a batch-centric view in the ViewTransitionTrans list. This method can be called anytime as long as the CLB defined by the parameter exists in the BatchSched list. Only the active transitions for a batch are available. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = OcxBatchVar.ViewTransitionSetCLBFocus ( CLB)
```

Parameters

CLB
Data Type: String
Campaign, Lot, and Batch ID data separated by forward slash (/) characters

ViewTransitionTagGetItem()

This method queries all information of a tag used in a transition logic expression based on its index in the ViewTransitionTag list. A String return value (166 characters) contains the tag name and tag value fields separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTagGetItem ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTagGetNumItems()

This method queries the number of tags in the ViewTransitionTag list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTagGetNumItems ()
```

ViewTransitionTagGetSelected()

This method queries the index of the currently selected tag in the ViewTransitionTag list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTagGetSelected ()
```

ViewTransitionTagGetTag()

This method queries the name of a tag based on its index in the ViewTransitionTag list. A String return value (84 characters) provides the tag name.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTagGetTag (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTagGetValue()

This method queries the value of a tag based on its index in the ViewTransitionTag list. A String return value (80 characters) provides the value of the tag.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTagGetValue ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTagSetSelected()

This method selects a specific tag based on its index in the ViewTransitionTag list. There are no actions permitted on individual tags. Therefore, this method is not typically used.

Syntax

```
OcxBatchVar.ViewTransitionTagSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTransGetDesc()

This method queries the description of a transition logic object based on its index in the ViewTransitionTrans list. A String return value (120 characters) provides the description text.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetDesc ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTransGetItem()

This method queries all information of a transition logic object based on its index in the ViewTransitionTrans list. A String return value (37 characters) contains the transition name, transition label, and time remaining value fields. Each field is separated by two spaces.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetItem ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTransGetLabel()

This method queries the label of a transition logic object based on its index in the ViewTransitionTrans list. A String return value (8 characters) provides the label of the transition.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetLabel ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTransGetNumItems()

This method queries the number of transition logic objects in the ViewTransitionTrans list. A Short return value contains the number of items in the list.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetNumItems ()
```

ViewTransitionTransGetSelected()

This method queries the index of the currently selected transition logic object in the ViewTransitionTrans list. A Short return value provides the index of the selected item.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetSelected ()
```

ViewTransitionTransGetTrans()

This method queries the name of a transition logic object based on its index in the ViewTransitionTrans list. A String return value (16 characters) provides the name of the transition.

Syntax

```
ReturnValue = OcxBatchVar.ViewTransitionTransGetTrans ( Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

ViewTransitionTransSetSelected()

This method selects a specific transition logic object based on its index in the ViewTransitionTrans list. This method is required to Force an active transition and also to populate the dependent transition expression and tag lists.

Syntax

```
OcxBatchVar.ViewTransitionTransSetSelected (Row)
```

Parameters

Row
Data Type: Short
Index of item to retrieve

Events

The following events are available for the Batch ActiveX object.

AllocQueueAdd

This event is called to request an application to add a batch to the container AllocQueue list.

Syntax

```
OcxBatchVar_AllocQueueAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: Short
Preformatted itemfield is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Instance	16 characters
Recipe ID	16 characters
Batch Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters

AllocQueueChange

This event is called to request an application that row data should be changed in the container AllocQueue list. Row indicates the row number that is changed. Item contains the updated text data.

Syntax

```
OcxBatchVar_AllocQueueChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: Short
Preformatted item to add to list. Each Item string consists of nine fields. Each field is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Instance	16 characters
Recipe ID	16 characters
Batch Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters

AllocQueueDelete

This event is called to request an application to delete a batch from the container AllocQueue list.
Syntax

```
OcxBatchVar_AllocQueueDelete (Row)
```

Parameters

Row
Data Type: Short
Delete the selected item

AllocQueueSelect

This event is called to request an application to select a batch in the container AllocQueue list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.
Syntax

```
OcxBatchVar_AllocQueueSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

AllocQueueUpdate

This event is called to request an application to completely refresh the AllocQueue list. From within this event handler, the container list should be cleared, the number of items in the AllocQueue list should be obtained, and the container list should be populated by iterating through the AllocQueue list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the SelectInstBusy event should be canceled.
Syntax

```
OcxBatchVar_AllocQueueUpdate ()
```

BatchMsgBoxMessage

This event is called to inform an application that a Batch Manager message is pending that should be shown to the user. From within this event handler, the string should be shown in a message box.

Syntax

```
OcxBatchVar_BatchMsgBoxMessage (szMsg)
```

Parameters

szMsg
Data Type: String
Batch Manager message to appear to the user

BatchSchedAdd

This event is called to request an application to add an item to the BatchSched list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_BatchSchedAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String [109]
Preformatted item to add to list. Each Item string consists of nine fields. Each field is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Recipe ID	16 characters
Batch Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters
Batch Action Required	2 characters

BatchSchedBusy

This event is called to inform an application that the BatchSched list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a BatchSchedUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_BatchSchedBusy ()
```

BatchSchedChange

This event is called to request an application to change an item in the BatchSched list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a BatchSchedSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_BatchSchedChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of nine fields. Each field is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Recipe ID	16 characters
Batch Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters
Batch Action Required	2 characters

BatchSchedDelete

This event is called to request an application to delete an item from the BatchSched list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_BatchSchedDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

BatchSchedFocusState

This event is called to request an application to change the focus of the functions being used. This event is only valid if using a batch-centric view (setting CLB focuses). The focus methods that can be called off this event are RecipeSetCLBFocus, MessageSetCLBFocus, and EquipmentSetCLBFocus. If the state is 1, the currently selected batch CLB should be passed as the focus. If the state is 0, an empty CLB (//) should be passed as the focus. The state is derived from the batch status of the currently selected batch.

Syntax

```
OcxBatchVar_BatchSchedFocusState (State)
```

Parameters

State
Data Type: Short
Focus state:
0 = Empty CLB
1 = CLB

BatchSchedSelect

This event is called to request an application to select an active batch in the container BatchSched list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_BatchSchedSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

BatchSchedUpdate

This event is called to request an application to completely refresh the BatchSched list. From within this event handler, the container list should be cleared, the number of items in the BatchSched list should be obtained, and the container list should be populated by iterating through the BatchSched list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the BatchSchedBusy event should be canceled.

Syntax

```
OcxBatchVar_BatchSchedUpdate ()
```

EditPhaseInstrUpdate

This event is called to request an application to retrieve the modified phase instructions. From within this event handler, the EditPhaseInstrGetInstr method should be called to get the latest instructions.

Syntax

```
OcxBatchVar_EditPhaseInstrUpdate ()
```

EditPhaseParamBusy

This event is called to inform an application that the EditPhaseParam list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until an EditPhaseParamUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_EditPhaseParamBusy ()
```

EditPhaseParamChange

This event is called to request an application to change an item in the EditPhaseParam list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by an EditPhaseParamSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EditPhaseParamChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

EditPhaseParamSelect

This event is called to request an application to select a phase parameter in the container EditPhaseParam list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EditPhaseParamSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

EditPhaseParamUpdate

This event is called to request an application to completely refresh the EditPhaseParam list. From within this event handler, the container list should be cleared, the number of items in the EditPhaseParam list should be obtained, and the container list should be populated by iterating through the EditPhaseParam list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the EditPhaseParamBusy event should be canceled.

Syntax

```
OcxBatchVar_EditPhaseParamUpdate ()
```

EditPhasePhaseAdd

This event is called to request an application to add an item to the EditPhasePhase list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_EditPhasePhaseAdd (Row, Item)
```

Parameters

Row
Data Type: Short

Position to add item
Item
Data Type: String
Preformatted item to add to list. Each Item string consists of five fields. Each field is separated by two spaces:

Equipment Name	16 characters
Unit Procedure Name	16 characters
Operation Name	16 characters
Phase Label	8 characters
Phase Name	16 characters

Note: The operation name is included in the string even if only two recipes levels are defined.

EditPhasePhaseBusy

This event is called to inform an application that the EditPhasePhase list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until an EditPhasePhaseUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_EditPhasePhaseBusy ()
```

EditPhasePhaseDelete

This event is called to request an application to delete an item from the EditPhasePhase list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EditPhasePhaseDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

EditPhasePhaseSelect

This event is called to request an application to select a phase in the container EditPhasePhase list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EditPhasePhaseSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

EditPhasePhaseUpdate

This event is called to request an application to completely refresh the EditPhasePhase list. From within this event handler, the container list should be cleared, the number of items in the EditPhasePhase list should be obtained, and the container list should be populated by iterating through the EditPhasePhase list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the EditPhasePhaseBusy event should be canceled.

Syntax

```
OcxBatchVar_EditPhasePhaseUpdate ()
```

EquipmentEquipBusy

This event is called to inform an application that the EquipmentEquip list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until an EquipmentEquipUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_EquipmentEquipBusy ()
```

EquipmentEquipChange

This event is called to request an application to change an item in the EquipmentEquip list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by an EquipmentEquipSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EquipmentEquipChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of five fields. Each field is separated by two spaces:

Equipment Name	16 characters
Equipment Type	10 characters
Allocation Status	16 characters
Equipment Status	16 Chars
Equipment State	16 characters

EquipmentEquipSelect

This event is called to request an application to select a unit or connection in the container EquipmentEquip list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EquipmentEquipSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

EquipmentEquipUpdate

This event is called to request an application to completely refresh the EquipmentEquip list. From within this event handler, the container list should be cleared, the number of items in the EquipmentEquip list should be obtained, and the container list should be populated by iterating through the EquipmentEquip list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the EquipmentEquipBusy event should be canceled.

Syntax

```
OcxBatchVar_EquipmentEquipUpdate ()
```

EquipmentInstBusy

This event is called to inform an application that the EquipmentInst list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until an EquipmentInstUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_EquipmentInstBusy ()
```

EquipmentInstSelect

This event is called to request an application to select an equipment instance in the container EquipmentInst list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_EquipmentInstSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

EquipmentInstUpdate

This event is called to request an application to completely refresh the EquipmentInst list. From within this event handler, the container list should be cleared, the number of items in the EquipmentInst list should be obtained, and the container list should be populated by iterating through the EquipmentInst list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the EquipmentInstBusy event should be canceled.

Syntax

```
OcxBatchVar_EquipmentInstUpdate ()
```

ErrorErrAdd

This event is called to request an application to add an item to the ErrorErr list. From within this event handler, the item should be added to the container list at the specified position.

Syntax

```
OcxBatchVar_ErrorErrAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String
Preformatted item to add to list

ErrorErrUpdate

This event is called to request an application to completely refresh the ErrorErr list. From within this event handler, the container list should be cleared, the number of items in the ErrorErr list should be obtained, and the container list should be populated by iterating through the ErrorErr list getting and adding each retrieved item.

Syntax

```
OcxBatchVar_ErrorErrUpdate ()
```

MessageMsgAdd

This event is called to request an application to add an item to the MessageMsg list. From within this event handler, the item should be added to the container list at the specified position. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_MessageMsgAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String
Preformatted item to add to list

MessageMsgBusy

This event is called to inform an application that the MessageMsg list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a MessageMsgUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_MessageMsgBusy ()
```

MessageMsgDelete

This event is called to request an application to delete an item from the MessageMsg list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_MessageMsgDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

MessageMsgSelect

This event is called to request an application to select a batch message in the container MessageMsg list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_MessageMsgSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

MessageMsgUpdate

This event is called to request an application to completely refresh the MessageMsg list. From within this event handler, the container list should be cleared, the number of items in the MessageMsg list should be obtained, and the container list should be populated by iterating through the MessageMsg list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the MessageMsgBusy event should be canceled.

Syntax

```
OcxBatchVar_MessageMsgUpdate ()
```

PhaseIlockBusy

This event is called to inform an application that the PhaseIlock list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a PhaseIlockUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_PhaseIlockBusy ()
```

PhaseIlockChange

This event is called to request an application to change an item in the Phaseilock list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a PhaseIlockSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhaseIlockChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of two fields. Each field is separated by two spaces:

Tag Name	84 characters
Tag Value	80 characters

PhaseIlockSelect

This event is called to request an application to select an interlock in the container Phaseilock list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhaseIlockSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

PhaseIlockUpdate

This event is called to request an application to completely refresh the Phaseilock list. From within this event handler, the container list should be cleared, the number of items in the Phaseilock list should be obtained, and the container list should be populated by iterating through the Phaseilock list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the PhaseIlockBusy event should be canceled.

Syntax

```
OcxBatchVar_PhaseIlockUpdate ()
```

PhaseInstrUpdate

This event is called to request an application to retrieve the modified phase instructions. From within this event handler, the PhaseInstrGetInstr method should be called to get the latest instructions.

Syntax

```
OcxBatchVar_PhaseInstrUpdate ()
```

PhaseParamBusy

This event is called to inform an application that the PhaseParam list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a PhaseParamUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_PhaseParamBusy ()
```

PhaseParamChange

This event is called to request an application to change an item in the PhaseParam list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a PhaseParamSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhaseParamChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to change in list. Each item string consists of eight fields. Each field is separated by two spaces:

Parameter Name	16 characters
Parameter Typez	12 characters
Parameter Extension	12 characters
Value	80 characters
Unit of Measure	16 characters
Editing Capability	8 characters
Data Class	2 characters
Enumeration Set Name	16 characters

PhaseParamSelect

This event is called to request an application to select a phase parameter in the container PhaseParam list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhaseParamSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

PhaseParamUpdate

This event is called to request an application to completely refresh the PhaseParam list. From within this event handler, the container list should be cleared, the number of items in the PhaseParam list should be obtained, and the container list should be populated by iterating through the PhaseParam list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the PhaseParamBusy event should be canceled.

Syntax

```
OcxBatchVar_PhaseParamUpdate ()
```

PhasePhaseAdd

This event is called to request an application to add an item to the PhasePhase list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_PhasePhaseAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of six fields. Each field is separated by two spaces:

Equipment Name	16 characters
Unit Procedure Name	16 characters
Operation Name	16 characters
Phase Name	16 characters
Phase Status	8 characters
Action Required	2 characters

Note: The operation name is included in the string even if only two recipes levels are defined

PhasePhaseBusy

This event is called to inform an application that the PhasePhase list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a PhasePhaseUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_PhasePhaseBusy ()
```

PhasePhaseChange

This event is called to request an application to change an item in the PhasePhase list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a PhasePhaseSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhasePhaseChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of six fields. Each field is separated by two spaces:

Equipment Name	16 characters
Unit Procedure Name	16 characters
Operation Name	16 characters
Phase Name	16 characters
Phase Status	8 characters
Action Required	2 characters

PhasePhaseDelete

This event is called to request an application to delete an item from the PhasePhase list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhasePhaseDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

PhasePhaseSelect

This event is called to request an application to select a phase in the container PhasePhase list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_PhasePhaseSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

PhasePhaseUpdate

This event is called to request an application to completely refresh the PhasePhase list. From within this event handler, the container list should be cleared, the number of items in the PhasePhase list should be obtained, and the container list should be populated by iterating through the PhasePhase list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the PhasePhaseBusy event should be canceled.

Syntax

```
OcxBatchVar_PhasePhaseUpdate ()
```

QuestionQuestAdd

This event is called to request an application to add an item to the QuestionQuest list. From within this event handler, the item should be added to the container list at the specified position. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_QuestionQuestAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to select item

Item
Data Type: String
Preformatted item to add to list

QuestionQuestBusy

This event is called to inform an application that the QuestionQuest list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a QuestionQuestUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_QuestionQuestBusy ()
```

QuestionQuestDelete

This event is called to request an application to delete an item from the QuestionQuest list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_QuestionQuestDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

QuestionQuestSelect

This event is called to request an application to select a question in the container QuestionQuest list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_QuestionQuestSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

QuestionQuestUpdate

This event is called to request an application to completely refresh the QuestionQuest list. From within this event handler, the container list should be cleared, the number of items in the QuestionQuest list should be obtained, and the container list should be populated by iterating through the QuestionQuest list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the QuestionQuestBusy event should be canceled.

Syntax

```
OcxBatchVar_QuestionQuestUpdate ()
```

SaveRecipeAuthorChanged

This event is called to inform an application that the author passed to the SaveRecipeSave method has been replaced by the DoneBy field also passed to that method.

Syntax

```
OcxBatchVar_SaveRecipeAuthorChanged (Author)
```

Parameters

Author
Data Type: String
Author of recipe

ScheduleExecInOrder

This event is called to inform an application of a change to the Batch Manager Execute in Order state.

Syntax

```
OcxBatchVar_ScheduleExecInOrder (State)
```

Parameters

State
Data Type: Boolean
Flag to signal run in order state:
True = Run in order enabled
False = Run in order disabled

ScheduleFormulaBusy

This event is called to inform an application that the ScheduleFormula list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleFormulaUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_ScheduleFormulaBusy ()
```

ScheduleFormulaSelect

This event is called to request an application to select a formula in the container ScheduleFormula list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleFormulaSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleFormulaUpdate

This event is called to request an application to completely refresh the ScheduleFormula list. From within this event handler, the container list should be cleared, the number of items in the ScheduleFormula list should be obtained, and the container list should be populated by iterating through the ScheduleFormula list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleFormulaBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleFormulaUpdate ()
```

ScheduleRecipeBusy

This event is called to inform an application that the ScheduleRecipe list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleRecipeUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_ScheduleRecipeBusy ()
```

ScheduleRecipeSelect

This event is called to request an application to select a recipe in the container ScheduleRecipe list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleRecipeSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleRecipeUpdate

This event is called to request an application to completely refresh the ScheduleRecipe list. From within this event handler, the container list should be cleared, the number of items in the ScheduleRecipe list should be obtained, and the container list should be populated by iterating through the ScheduleRecipe list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleRecipeBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleRecipeUpdate ()
```

ScheduleSchedAdd

This event is called to request an application to add an item to the ScheduleSched list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_ScheduleSchedAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String [119]
Preformatted item to add to list. Each Item string consists of nine fields. Each field is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Recipe ID	16 characters
Recipe Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters

ScheduleSchedBusy

This event is called to inform an application that the ScheduleSched list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleSchedUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_ScheduleSchedBusy ()
```

ScheduleSchedChange

This event is called to request an application to change an item in the ScheduleSched list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a ScheduleSchedSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleSchedChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String [119]
Preformatted item to add to list. Each Item string consists of nine fields. Each field is separated by two spaces:

Campaign ID	16 characters
Lot ID	16 characters
Batch ID	16 characters
Recipe ID	16 characters
Recipe Quantity	8 characters
Train Name	16 characters
Batch Mode	9 characters
Batch Status	8 characters

ScheduleSchedDelete

This event is called to request an application to delete an item from the ScheduleSched list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleSchedDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

ScheduleSchedSelect

This event is called to request an application to select a scheduled batch in the container ScheduleSched list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleSchedSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleSchedUpdate

This event is called to request an application to completely refresh the ScheduleSched list. From within this event handler, the container list should be cleared, the number of items in the ScheduleSched list should be obtained, and the container list should be populated by iterating through the ScheduleSched list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleSchedBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleSchedUpdate ()
```

ScheduleStateBusy

This event is called to inform an application that the ScheduleState list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleStateUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_ScheduleStateBusy ()
```

ScheduleStateSelect

This event is called to request an application to select a state in the container ScheduleState list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleStateSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleStateUpdate

This event is called to request an application to completely refresh the ScheduleState list. From within this event handler, the container list should be cleared, the number of items in the ScheduleState list should be obtained, and the container list should be populated by iterating through the ScheduleState list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleStateBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleStateUpdate ()
```

ScheduleTrainBusy

This event is called to inform an application that the ScheduleTrain list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleTrainUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_ScheduleTrainBusy ()
```

ScheduleTrainSelect

This event is called to request an application to select a train in the container ScheduleTrain list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleTrainSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleTrainUpdate

This event is called to request an application to completely refresh the ScheduleTrain list. From within this event handler, the container list should be cleared, the number of items in the ScheduleTrain list should be obtained, and the container list should be populated by iterating through the ScheduleTrain list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleTrainBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleTrainUpdate ()
```

ScheduleTypeBusy

This event is called to inform an application that the ScheduleType list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ScheduleTypeUpdate event is received. Many busy events may be received before getting the update event.

Syntax

```
OcxBatchVar_ScheduleTypeBusy ()
```

ScheduleTypeSelect

This event is called to request an application to select a type in the container ScheduleType list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ScheduleTypeSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ScheduleTypeUpdate

This event is called to request an application to completely refresh the ScheduleType list. From within this event handler, the container list should be cleared, the number of items in the ScheduleType list should be obtained, and the container list should be populated by iterating through the ScheduleType list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ScheduleTypeBusy event should be canceled.

Syntax

```
OcxBatchVar_ScheduleTypeUpdate ()
```

SecurityPending

This event is called to inform an application that a security operation is pending. This is only useful if the modeless security operation is used (see *"SecurityMode"*) or if the ScheduleSetAccessSecurity method is being used. This event can be used to disable the application while security is pending.

Syntax

```
OcxBatchVar_SecurityPending (State)
```

Parameters

State
Data Type: Short
Flag to signal pending security:
0 = Complete
1 = Pending

SelectEquipAdd

This event is called to request an application to add an item to the SelectEquip list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_SelectEquipAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of two fields. Each field is separated by two spaces:

Unit Name	16 characters
Unit Status	16 characters

SelectEquipBusy

This event is called to inform an application that the SelectEquip list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a SelectEquipUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_SelectEquipBusy ()
```

SelectEquipChange

This event is called to request an application to change an item in the SelectEquip list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a SelectEquipSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_SelectEquipChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of two field. Each field is separated by two spaces:

Unit Name	16 characters
Unit Status	16 characters

SelectEquipDelete

This event is called to request an application to delete an item from the SelectEquip list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_SelectEquipDelete (Row)
```

Parameters

Row
Data Type: Short

Position to delete item

SelectEquipSelect

This event is called to request an application to select a unit in the container SelectEquip list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_SelectEquipSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

SelectEquipUpdate

This event is called to request an application to completely refresh the SelectEquip list. From within this event handler, the container list should be cleared, the number of items in the SelectEquip list should be obtained, and the container list should be populated by iterating through the SelectEquip list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the SelectEquipBusy event should be canceled.

Syntax

```
OcxBatchVar_SelectEquipUpdate ()
```

SelectInstAdd

This event is called to request an application to add an item to the SelectInst list. From within this event handler, the item should be added to the container list at the specified position. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_SelectInstAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String [16]
Preformatted item to add to list

SelectInstBusy

This event is called to inform an application that the SelectInst list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a SelectInstUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_SelectInstBusy ()
```

SelectInstDelete

This event is called to request an application to delete an item from the SelectInst list. From within this event handler, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_SelectInstDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

SelectInstSelect

This event is called to request an application to select a process instance in the container SelectInst list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_SelectInstSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

SelectInstUpdate

This event is called to request an application to completely refresh the SelectInst list. From within this event handler, the container list should be cleared, the number of items in the SelectInst list should be obtained, and the container list should be populated by iterating through the SelectInst list getting and adding each retrieved item. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the SelectInstBusy event should be canceled.

Syntax

```
OcxBatchVar_SelectInstUpdate ()
```

SystemShuttingDown

This event is called to inform an application that the Batch Server is shutting down or stopping all run-time applications. This event must be handled by any application that calls the Init method. Upon receiving this event an application should call the Term method. Failure to do this can cause the Batch Server shutdown sequence to wait indefinitely until the control is destroyed. The control should not be destroyed while in this event.

Syntax

```
OcxBatchVar_SystemShuttingDown ()
```

ViewTransitionExpUpdate

This event is called to request an application to retrieve the modified expression. From within this event handler, the ViewTransitionGetExp method should be called to get the latest expression.

Syntax

```
OcxBatchVar_ViewTransitionExpUpdate ()
```

ViewTransitionTagBusy

This event is called to inform an application that the ViewTransitionTag list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ViewTransitionTagUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_ViewTransitionTagBusy ()
```

ViewTransitionTagChange

This event is called to request an application to change an item in the ViewTransitionTag list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a ViewTransitionTagSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ViewTransitionTagChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of two fields. Each field is separated by two spaces:

Tag Name	84 characters
Tag value	80 characters

ViewTransitionTagSelect

This event is called to request an application to select a tag in the container ViewTransitionTag list. From within this event handler, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ViewTransitionTagSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ViewTransitionTagUpdate

This event is called to request an application to completely refresh the ViewTransitionTag list. From within this event handler, the container list should be cleared, the number of items in the ViewTransitionTag list should be obtained, and the container list should be populated by iterating through the ViewTransitionTag list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ViewTransitionTagBusy event should be canceled.

Syntax

```
OcxBatchVar_ViewTransitionTagUpdate ()
```

ViewTransitionTransAdd

This event is called to request an application to add an item to the ViewTransitionTrans list. From within this event handler, the item should be added to the container list at the specified position. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set.

Syntax

```
OcxBatchVar_ViewTransitionTransAdd (Row, Item)
```

Parameters

Row
Data Type: Short
Position to add item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of three fields. Each field is separated by two spaces:

Transition Name	16 characters
Transition Label	8 characters
Time Remaining	9 characters

ViewTransitionTransBusy

This event is called to inform an application that the ViewTransitionTrans list is busy getting extensive updates. From within this event handler, an application busy state (a working cursor) can be set. The busy state is not cleared until a ViewTransitionTransUpdate event is received. Many busy events may be received before getting the update event

Syntax

```
OcxBatchVar_ViewTransitionTransBusy ()
```

ViewTransitionTransChange

This event is called to request an application to change an item in the ViewTransitionTrans list. From within this event handler, the item should be removed from the container list at the specified position and the new item must be added. Some container list controls support a simple change mechanism. The complete item string can be used or individual item fields can be obtained and formatted as needed. Because many container list controls force an item to be deleted and added again, this event is always followed by a ViewTransitionTransSelect event. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ViewTransitionTransChange (Row, Item)
```

Parameters

Row
Data Type: Short
Position to change item

Item
Data Type: String
Preformatted item to add to list. Each Item string consists of three fields. Each field is separated by two spaces:

Transition Name	16 characters
Transition Label	8 characters
Time Remaining	9 characters

ViewTransitionTransDelete

This event is called to request an application to delete an item from the ViewTransitionTrans list. From within this event handler,, the item should be removed from the container list at the specified position. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ViewTransitionTransDelete (Row)
```

Parameters

Row
Data Type: Short
Position to delete item

ViewTransitionTransSelect

This event is called to request an application to select a transition logic object in the container ViewTransitionTrans list. From within this event handler,, the item should be selected in the container list at the specified position. This event can also be used to extract information about the selected item and propagate the information to the rest of the application as necessary. For example, a button can be enabled or disabled based on a state retrieved from the selected item. Do not try to change the list selection from within this event.

Syntax

```
OcxBatchVar_ViewTransitionTransSelect (Row)
```

Parameters

Row
Data Type: Short
Position to select item

ViewTransitionTransUpdate

This event is called to request an application to completely refresh the ViewTransitionTrans list. From within this event handler,, the container list should be cleared, the number of items in the ViewTransitionTrans list should be obtained, and the container list should be populated by iterating through the ViewTransitionTrans list getting and adding each retrieved item. The complete item string can be used or individual item fields can be obtained and formatted as needed. An item can be selected in the list from within this event. Typically, this is not necessary, especially when the AutoSelectMode property is set. Also, when this event gets called, the busy state initiated by the ViewTransitionTransBusy event should be canceled.

Syntax

```
OcxBatchVar_ViewTransitionTransUpdate ()
```

Error Return Values

There are two kinds of errors generated from the OcxBatch control. First are the errors returned directly by the control from its methods. Second are the errors caught by Batch Manager and returned asynchronously through the controls Display Error function. The following list of errors simply defines the errors returned by the control. An application can try to catch these by checking return values. An application can catch Batch Manager errors by using the Display Errors function as a global error handler. However, Batch Manager errors are not available immediately following the method call that caused the error. The application must set up an asynchronous error catching mechanism or simply list Batch Manager errors.

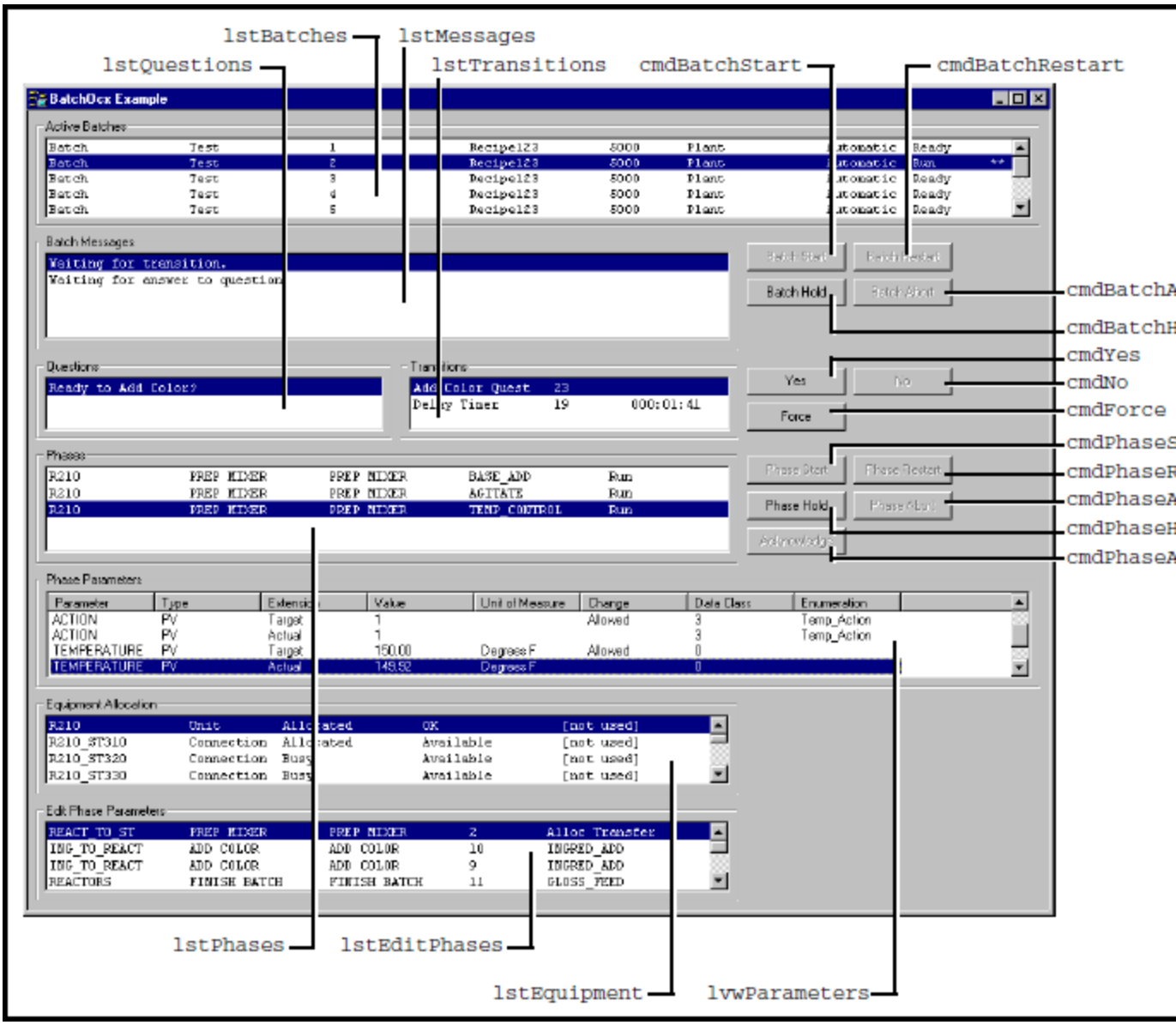
Below is a description of the possible errors returned from OcxBatch method calls.

Value	Description
0	No error
100	Invalid CLB
101	Batch status is invalid for operation
102	Invalid security information
104	Error talking to Batch Manager
105	Invalid equipment
106	Invalid equipment instance
107	Invalid equipment type
108	Invalid phase
109	Invalid phase status
110	No ack required
111	Invalid parameter
112	Invalid question

Value	Description
113	Invalid question answer
114	Invalid schedule
115	Invalid handshake
116	Invalid unit control available
117	Recipe exists
118	Batch Manager is busy
119	Invalid recipe quantity
120	Failed to connect to InfoMngr
121	No document assigned to phase
122	Invalid document assigned to phase
123	Invalid acknowledgement of phase document
170	Init method has not yet been called
171	OcxBatch general system error
172	OcxBatch security pending (not usually an error)
173	OcxBatch cannot connect to Batch Server.
174	OcxBatch has already had the Init method called
175	OcxBatch license failure
200	OcxBatch system error
1000-1999	DoneBy Security error
2000-2999	CheckBy Security error

Example

The following example illustrates how to use the Batch Management ActiveX control. In this example, a single form includes nine lists that represent the basic information available from the control. Basic list boxes are used for each list except for the phase parameters list. Because of the number and sizes of the fields in this list, a list view control was used. The application code shows how the control methods and events are handled. The active batch list is used to focus on a specific batch selected from the list and all other lists update accordingly. There are also several buttons that enable or disable automatically based on the selected items in the lists. The primary goal of this example is to help demonstrate how focusing and event handling should be used. The same concepts shown here can be applied elsewhere in the control. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure. This example assumes that the Batch ActiveX control has been properly added to the application.




```
// Beginning of example code with general startup and shutdown logic
private void BatchActiveXControl_Load(object sender, EventArgs e)
{
```

```
// Define required local variable
int ReturnCode;
// Set host property and initialize control
OCXBatchCtrl.Host = " BATCHSERVER ";
ReturnCode = OCXBatchCtrl.Init();
}
private void OCXBatchCtrl_SystemShuttingDown(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Terminate control and end program when system is shutting down
ReturnCode = OCXBatchCtrl.Term();
}
private void PopulateBatch()
{
lstBatches.Items.Clear();
int iNumSchedItems = OCXBatchCtrl.ScheduleSchedGetNumItems();
if (iNumSchedItems > 0)
{
//Populate List Box
lstBatches.Items.Clear();
for (int iIndex = 0; iIndex < iNumSchedItems;iIndex++)
{
OCXBatchCtrl.ScheduleSchedSetSelected((short)iIndex);
string sItem = OCXBatchCtrl.ScheduleSchedGetItem((short)iIndex);
lstBatches.Items.Insert(iIndex, sItem);
}
}
}
private void cmdBatchStart_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Start the batch when the Batch Start button is pressed
ReturnCode = OCXBatchCtrl.BatchStartBatch("", "", "", "");
}
private void cmdBatchHold_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Hold the batch when the Batch Hold button is pressed
ReturnCode = OCXBatchCtrl.BatchHoldBatch("", "", "", "");
}
private void cmdBatchRestart_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// ReStart the batch when the Batch ReStart button is pressed
ReturnCode = OCXBatchCtrl.BatchRestartBatch("", "", "", "");
}
private void cmdBatchAbort_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Abort the batch when the Batch Abort button is pressed
ReturnCode = OCXBatchCtrl.BatchAbortBatch("", "", "", "");
}
private void cmdYes_Click(object sender, EventArgs e)
```

```
{
// Define required local variable
int ReturnCode;
//Answer Yes to a question when the Yes button is pressed
ReturnCode = OCXBatchCtrl.QuestAnswerQuest(1, "", "", "", "");
}
private void cmdNo_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
//Answer No to a question when the Yes button is pressed
ReturnCode = OCXBatchCtrl.QuestAnswerQuest(0, "", "", "", "");
}
private void cmdForce_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
//Answer Yes to a question when the Yes button is pressed
ReturnCode = OCXBatchCtrl.ViewTransitionForceTransition("", "", "", "");
}
private void cmdPhaseStart_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Start the phase when the Phase Start button is pressed
ReturnCode = OCXBatchCtrl.PhaseStartPhase("", "", "", "");
}
private void cmdPhaseHold_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
// Hold the phase when the Phase Hold button is pressed
ReturnCode = OCXBatchCtrl.PhaseHoldPhase("", "", "", "");
}
private void cmdPhaseRestart_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
//Restart the phase when the Phase Restart button is pressed
ReturnCode = OCXBatchCtrl.PhaseRestartPhase("", "", "", "");
}
private void cmdPhaseAbort_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
//Abort the phase when the Phase Abort button is pressed
ReturnCode = OCXBatchCtrl.PhaseAbortPhase("", "", "", "");
}
private void cmdAcknowledge_Click(object sender, EventArgs e)
{
// Define required local variable
int ReturnCode;
//Acknowledge the phase when the Phase Acknowledge button is pressed
ReturnCode = OCXBatchCtrl.PhaseAckPhase("", "", "", "");
}
private void lstBatches_SelectedIndexChanged(object sender, EventArgs e)
{
// Define required local variables
```

```
string CLB;
int ReturnCode;
/*Using batch focus, select the batch that was selected in
the Active Batches list, get the CLB of the selected batch,
and change the focus of the other lists to reflect the
selected batch*/
OCXBatchCtrl.BatchSchedSetSelected((short)lstBatches.SelectedIndex);
CLB = OCXBatchCtrl.BatchSchedGetCLB((short)lstBatches.SelectedIndex);
ReturnCode = OCXBatchCtrl.RecipeSetCLBFocus(CLB);
ReturnCode = OCXBatchCtrl.MessageSetCLBFocus(CLB);
ReturnCode = OCXBatchCtrl.EquipmentSetCLBFocus(CLB);
ReturnCode = OCXBatchCtrl.EditPhaseSetCLBFocus(CLB);
ReturnCode = OCXBatchCtrl.ViewTransitionSetCLBFocus(CLB);
}
private void lstMessages_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the message that was selected in the Batch Messages list
OCXBatchCtrl.MessageMsgSetSelected((short)lstMessages.SelectedIndex);
}
private void lstQuestions_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the message that was selected in the Batch Questions list
OCXBatchCtrl.QuestionQuestSetSelected((short)lstQuestions.SelectedIndex);
}
private void lstTransitions_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the message that was selected in the Batch Transitions list
OCXBatchCtrl.ViewTransitionTagSetSelected((short)lstTransitions.SelectedInde
x);
}
private void lstPhases_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the message that was selected in the Batch Transitions list
OCXBatchCtrl.PhasePhaseSetSelected((short)lstPhases.SelectedIndex);
}
private void lvwParameters_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the parameter that was selected in the Phase Parameters list
OCXBatchCtrl.PhaseParamSetSelected((short)lvwParameters.SelectedIndices[0];
}
private void lstEquipment_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the equipment that was selected in the Equipment Allocation list
OCXBatchCtrl.EquipmentEquipSetSelected((short)lstEquipment.SelectedIndex);
}
private void lstEditPhases_SelectedIndexChanged(object sender, EventArgs e)
{
// Select the phase that was selected in the Edit Phase Parameters list
OCXBatchCtrl.EditPhasePhaseSetSelected((short)lstEditPhases.SelectedIndex);
}
// The next group of functions responds to OCXBatch Batch Schedule events
private void OCXBatchCtrl_

(object sender, AxOCXBATCHLib._DOcxBatchEvents_BatchSchedAddEvent e)
{
// Define required local variable
int position;
int Row = e.row;
```

```
string Item = e.item;
// Add the new batch item to the Active Batches list at the
//row specified, and if possible, make sure the currently
//selected batch remains selected
position = lstBatches.SelectedIndex;
lstBatches.Items.Insert(Row, Item);
if (position == -1)
lstBatches.SelectedIndex = 0;
else
if (position < Row)
lstBatches.SelectedIndex = position;
else
lstBatches.SelectedIndex = position + 1;
}
private void OCXBatchCtrl_ScheduleSchedAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_ScheduleSchedAddEvent e)
{
// Define required local variable
int position;
int Row = e.row;
string Item = e.item;
// Add the new batch item to the Active Batches list at the
//e.Row specified, and if possible, make sure the currently
//selected batch remains selected
position = lstBatches.SelectedIndex;
lstBatches.Items.Insert(Row, Item);
if (position == -1)
lstBatches.SelectedIndex = 0;
else
if (position < Row)
lstBatches.SelectedIndex = position;
else
lstBatches.SelectedIndex = position + 1;
}
private void OCXBatchCtrl_BatchSchedBusy(object sender, EventArgs e)
{
// Indicate to the user that the Active Batches list is being updated
lstBatches.Items.Clear();
//lstBatches.Items.Add("Updating...");
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_BatchSchedChange(object sender,
AxOCXBATCHLib._DOcxBatchEvents_BatchSchedChangeEvent e)
{
// Define required local variable
int position;
int Row = e.row;
string Item = e.item;
// Change the batch item in the Active Batches list at the
//Row specified, and make sure the currently selected batch remains selected
position = lstBatches.SelectedIndex;
lstBatches.Items.RemoveAt(Row);
lstBatches.Items.Insert(Row, Item);
lstBatches.SelectedIndex = position;
}
private void OCXBatchCtrl_BatchSchedDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_BatchSchedDeleteEvent e)
{
```

```
// Define required local variable
int position;
int Row = e.row;
// Delete the batch from the Active Batches list at the Row
// specified, and if possible, make sure the currently
// selected batch remains selected
position = lstBatches.SelectedIndex;
lstBatches.SelectedIndex = Row;
lstBatches.Items.RemoveAt(Row);
if (position < Row)
lstBatches.SelectedIndex = position;
else
lstBatches.SelectedIndex = position - 1;
}
private void OCXBatchCtrl_BatchSchedSelect(object sender,
AxOCXBATCHLib._DOcxBatchEvents_BatchSchedSelectEvent e)
{
// Define required local variable;
long BatchEditMask = 0;
int Row = e.row;

// Set the batch control buttons accordingly whenever a batch
// is selected in the Active Batches list
if (Row == -1)
{
cmdBatchStart.Enabled = false;
cmdBatchHold.Enabled = false;
cmdBatchRestart.Enabled = false;
cmdBatchAbort.Enabled = false;
}
else
{
BatchEditMask = OCXBatchCtrl.BatchSchedGetEditMask((short)Row);
if((BatchEditMask & 1) == 0)
cmdBatchStart.Enabled = false;
else
cmdBatchStart.Enabled = true;
if((BatchEditMask & 2) == 0)
cmdBatchHold.Enabled = false;
else
cmdBatchHold.Enabled = true;
if ((BatchEditMask & 4) == 0)
cmdBatchRestart.Enabled = false;
else
cmdBatchRestart.Enabled = true;
if ((BatchEditMask & 8) == 0)
cmdBatchAbort.Enabled = false;
else
cmdBatchAbort.Enabled = true;
      }
}
private void OCXBatchCtrl_BatchSchedUpdate(object sender, EventArgs e)
{
// Define required local variable;
int i = 0;
// Update the entire Active Batches list lstBatches.Clear();
for (i = 1; i <= OCXBatchCtrl.BatchSchedGetNumItems(); i++)
{
```

```
lstBatches.Items.Add(OCXBatchCtrl.BatchSchedGetItem((short)(i - 1)));
}
if (lstBatches.Items.Count > 0)
{
lstBatches.SelectedIndex = 0;
}
else
{
lstBatches.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
// The next group of functions responds to OCXBatch Edit Phase Parameter events
private void OCXBatchCtrl_EditPhasePhaseAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_EditPhasePhaseAddEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Add the new phase item to the Edit Phase Parameters list
//at the row specified, and if possible, make sure the
//currently selected phase remains selected;
position = lstEditPhases.SelectedIndex;
lstEditPhases.Items.Insert(Row, Item);
if (position == -1)
{
lstEditPhases.SelectedIndex = 0;
}
else
{
if (position < Row)
{
lstEditPhases.SelectedIndex = position;
}
else
{
lstEditPhases.SelectedIndex = position + 1;
}
}
}
private void OCXBatchCtrl_EditPhasePhaseDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_EditPhasePhaseDeleteEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
// Delete the phase from the Edit Phase Parameters list at
//the row specified, and if possible, make sure the currently
//selected phase remains selected
position = lstEditPhases.SelectedIndex;
lstEditPhases.Items.RemoveAt(Row);
if (position < Row)
{
lstEditPhases.SelectedIndex = position;
}
else
{
```

```
lstEditPhases.SelectedIndex = position - 1;
}
}
private void OCXBatchCtrl_EditPhasePhaseBusy(object sender, EventArgs e)
{
// Indicate to the user that the Edit Phase Parameters listis being updated
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_EditPhasePhaseUpdate(object sender, EventArgs e)
{
// Define required local variable
int i = 0;
// Update the entire Edit Phase Parameters list
lstEditPhases.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.EditPhasePhaseGetNumItems(); i++)
{
lstEditPhases.Items.Add(OCXBatchCtrl.EditPhasePhaseGetItem((short)
(i - 1)));
}
if (lstEditPhases.SelectedIndex > 0)
{
lstEditPhases.SelectedIndex = 0;
}
else
{
lstEditPhases.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
//The next group of functions responds to OCXBatch Equipment Allocation events
private void OCXBatchCtrl_EquipmentEquipBusy(object sender, EventArgs e)
{
// Indicate to the user that the Equipment Allocation list is being updated
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_EquipmentEquipChange(object sender,
AxOCXBATCHLib._DOcxBatchEvents_EquipmentEquipChangeEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Change the equipment item in the Equipment Allocation
//list at the row specified, and make sure the currently
//selected unit or connection remains selected
position = lstEquipment.SelectedIndex;
lstEquipment.Items.RemoveAt(Row);
lstEquipment.Items.Insert(Row, Item);
lstEquipment.SelectedIndex = position;
}
private void OCXBatchCtrl_EquipmentEquipUpdate(object sender, EventArgs e)
{
// Define required local variable
int i = 0;
// Update the entire Equipment Allocation list
lstEquipment.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.EquipmentEquipGetNumItems(); i++)
{
```

```
lstEquipment.Items.Add(OCXBatchCtrl.EquipmentEquipGetItem((short)
(i - 1)));
if (lstEquipment.Items.Count > 0)
{
lstEquipment.SelectedIndex = 0;
}
else
{
lstEquipment.SelectedIndex = -1;
}
}
Cursor.Current = Cursors.Default;
}
//The next group of functions responds to OCXBatch Batch Message events
private void OCXBatchCtrl_MessageMsgAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_MessageMsgAddEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Add the new message item to the Batch Messages list at
//the row specified, and if possible, make sure the currently
//selected message remains selected
position = lstMessages.SelectedIndex;
lstMessages.Items.Insert(Row, Item);
if (position == -1)
{
lstMessages.SelectedIndex = 0;
}
else
{
if (position < Row)
{
lstMessages.SelectedIndex = position;
}
else
{
lstMessages.SelectedIndex = position + 1;
}
}
}
private void OCXBatchCtrl_MessageMsgBusy(object sender, EventArgs e)
{
// Indicate to the user that the Batch Messages list is being updated;
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_MessageMsgDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_MessageMsgDeleteEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
// Delete the message from the Batch Messages list at the
//row specified, and if possible, make sure the currently
//selected message remains selected
position = lstMessages.SelectedIndex;
lstMessages.Items.RemoveAt(Row);
```

```
if (position < Row)
{
lstMessages.SelectedIndex= position;
}
else
{
lstMessages.SelectedIndex = position - 1;
}
}
private void OCXBatchCtrl_MessageMsgUpdate(object sender, EventArgs e)
{
// Define required local variable
int i = 0;
// Update the entire Batch Messages list
lstMessages.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.MessageMsgGetNumItems(); i++)
{
}
lstMessages.Items.Add(OCXBatchCtrl.MessageMsgGetItem((short)(i - 1)));
if (lstMessages.Items.Count > 0)
{
lstMessages.SelectedIndex = 0;
}
else
{
lstMessages.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
// The next group of functions responds to OCXBatch Phase Parameter events
private void OCXBatchCtrl_PhaseParamBusy(object sender, EventArgs e)
{
// Indicate to the user that the Batch Messages list is being updated;
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_PhaseParamChange(object sender,
AxOCXBATCHLib._DOcxBatchEvents_PhaseParamChangeEvent e)
{
// Define required local variable
ListViewItem l = null;
int Row = e.row;
string Item = e.item;
// Change the parameter item in the Phase Parameters list at the row specified
l = lvwParameters.SelectedItems[Row];
l.SubItems.Add(Item.Substring(18, 12).Trim());
l.SubItems.Add(Item.Substring(32, 12).Trim());
l.SubItems.Add(Item.Substring(46, 80).Trim());
l.SubItems.Add(Item.Substring(128, 16).Trim());
l.SubItems.Add(Item.Substring(146, 8).Trim());
l.SubItems.Add(Item.Substring(156, 2).Trim());
l.SubItems.Add(Item.Substring(160, 16).Trim());
}
private void OCXBatchCtrl_PhaseParamUpdate(object sender, EventArgs e)
{
// Define required local variables
int i = 0;
ListViewItem l = null;
// Update the entire Phase Parameters list
```

```
string Item;
lvwParameters.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.PhaseParamGetNumItems(); i++)
{
Item = (string)OCXBatchCtrl.PhaseParamGetItem((short)i);
l = lvwParameters.Items.Add(Item.Substring(0, 16).Trim());
l.SubItems.Add(Item.Substring(18, 12).Trim());
l.SubItems.Add(Item.Substring(32, 12).Trim());
l.SubItems.Add(Item.Substring(46, 80).Trim());
l.SubItems.Add(Item.Substring(128, 16).Trim());
l.SubItems.Add(Item.Substring(146, 8).Trim());
l.SubItems.Add(Item.Substring(156, 2).Trim());
l.SubItems.Add(Item.Substring(160, 16).Trim());
}
Cursor.Current = Cursors.Default;
}
// The next group of functions responds to OCXBatch Phase events
private void OCXBatchCtrl_PhasePhaseAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_PhasePhaseAddEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Add the new phase item to the Phases list at the row
//specified, & if possible, make sure the currently selected phase remains
selected;
position = lstPhases.SelectedIndex;
lstPhases.Items.Insert(Row, Item);
if (position == -1)
{
lstPhases.SelectedIndex = 0;
}
else
{
if (position < Row)
{
lstPhases.SelectedIndex = position;
}
else
{
lstPhases.SelectedIndex = position + 1;
}
}
}
private void OCXBatchCtrl_PhasePhaseBusy(object sender, EventArgs e)
{
// Indicate to the user that the Phases list is being updated
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_PhasePhaseChange(object sender,
AxOCXBATCHLib._DOcxBatchEvents_PhasePhaseChangeEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Change the phase item in the Phases list at the row
```

```
//specified, and make sure the currently selected phase remains selected
position = lstPhases.SelectedIndex;
lstPhases.Items.RemoveAt(Row);
lstPhases.Items.Insert(Row, Item);
lstPhases.SelectedIndex = position;
}
private void OCXBatchCtrl_PhasePhaseDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_PhasePhaseDeleteEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
// Delete the phase from the Phases list at the row
//specified, and if possible, make sure the currently
//selected phase remains selected
position = lstPhases.SelectedIndex;
lstPhases.Items.RemoveAt(Row);
if (position < Row)
{
lstPhases.SelectedIndex = position;
}
else
{
lstPhases.SelectedIndex = position - 1;
}
}
private void OCXBatchCtrl_PhasePhaseSelect(object sender,
AxOCXBATCHLib._DOcxBatchEvents_PhasePhaseSelectEvent e)
{
// Define required local variable
long PhaseEditMask = 0;
int Row = e.row;
// Set the phase control buttons accordingly
// whenever a phase is selected in the Phases list
PhaseEditMask = OCXBatchCtrl.PhasePhaseGetEditMask((short)Row);
if((PhaseEditMask & 1024) == 0)
cmdPhaseStart.Enabled = false;
else
cmdPhaseStart.Enabled = true;
if ((PhaseEditMask & 128) == 0)
cmdPhaseHold.Enabled = false;
else
cmdPhaseHold.Enabled = true;
if ((PhaseEditMask & 256) == 0)
cmdPhaseRestart.Enabled = false;
else
cmdPhaseRestart.Enabled = true;
if ((PhaseEditMask & 512) == 0)
cmdPhaseAbort.Enabled = false;
else
cmdPhaseAbort.Enabled = true;
if ((PhaseEditMask & 16447) == 0)
cmdPhaseAck.Enabled = false;
else
cmdPhaseAck.Enabled = true;
}
private void OCXBatchCtrl_PhasePhaseUpdate(object sender, EventArgs e)
{
```

```
// Define required local variable
int i = 0;
// Update the entire Phases list
lstPhases.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.PhasePhaseGetNumItems(); i++)
{
lstPhases.Items.Add(OCXBatchCtrl.PhasePhaseGetItem((short)(i - 1)));
}
if (lstPhases.Items.Count > 0)
{
lstPhases.SelectedIndex = 0;
}
else
{
lstPhases.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
// The next group of functions responds to OCXBatch Question events
private void OCXBatchCtrl_QuestionQuestAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_QuestionQuestAddEvent e)
{
// Define required local variable;
int position = 0;
int Row = e.row;
string Item = e.item;
// Add the new question() item to the Questions list at the
// row specified, & if possible, make sure the currently
// selected question remains selected;
position = lstQuestions.SelectedIndex;
lstQuestions.Items.Insert(Row, Item);
if (position == -1)
{
lstQuestions.SelectedIndex = 0;
}
else
{
if (position < Row)
{
lstQuestions.SelectedIndex = position;
}
else
{
lstQuestions.SelectedIndex = position + 1;
}
}
}
private void OCXBatchCtrl_QuestionQuestBusy(object sender, EventArgs e)
{
// Indicate to the user that the Questions list is being updated
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_QuestionQuestDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_QuestionQuestDeleteEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
```

```
// Delete the question from the Questions list at the row
//specified, and if possible, make sure the currently
//selected question remains selected
position = lstQuestions.SelectedIndex;
lstQuestions.Items.RemoveAt(Row);
if (position < Row)
{
lstQuestions.SelectedIndex = position;
}
else
{
lstQuestions.SelectedIndex = position - 1;
}
}
private void OCXBatchCtrl_QuestionQuestSelect(object sender,
AxOCXBATCHLib._DOcxBatchEvents_QuestionQuestSelectEvent e)
{
// Define required local variable
long QuestionType = 0;
int Row = e.row;
// Set the question yes and no buttons accordingly whenever
//a question is selected in the Questions list
if (Row == -1)
{
cmdYes.Enabled = false;
cmdNo.Enabled = false;
}
else
{
QuestionType = OCXBatchCtrl.QuestionQuestGetType((short)Row);
if (QuestionType == 0)
{
cmdYes.Enabled = true;
cmdNo.Enabled = true;
}
else
{
cmdYes.Enabled = true;
cmdNo.Enabled = false;
}
}
}
private void OCXBatchCtrl_QuestionQuestUpdate(object sender, EventArgs e)
{
// Define required local variable
int i = 0;
// Update the entire Questions list
lstQuestions.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.QuestionQuestGetNumItems(); i++)
{
lstQuestions.Items.Add(OCXBatchCtrl.QuestionQuestGetItem((short)
(i - 1)));
}
if (lstQuestions.Items.Count > 0)
{
lstQuestions.SelectedIndex = 0;
}
else
```

```
{
lstQuestions.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
// The next group of functions responds to OCXBatch Transition events
private void OCXBatchCtrl_ViewTransitionTransAdd(object sender,
AxOCXBATCHLib._DOcxBatchEvents_ViewTransitionTransAddEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Add the new transition item to the Transitions list at
//the row specified, and if possible, make sure the currently
//selected transition remains selected
position = lstTransitions.SelectedIndex;
lstTransitions.Items.Insert(Row, Item);
if (position == -1)
{
lstTransitions.SelectedIndex = 0;
}
else
{
if (position < Row)
{
lstTransitions.SelectedIndex = position;
}
else
{
lstTransitions.SelectedIndex = position + 1;
}
}
}
private void OCXBatchCtrl_ViewTransitionTransBusy(object sender, EventArgs e)
{
// Indicate to the user that the Transitions list is being updated
Cursor.Current = Cursors.WaitCursor;
}
private void OCXBatchCtrl_ViewTransitionTransChange(object sender,
AxOCXBATCHLib._DOcxBatchEvents_ViewTransitionTransChangeEvent e)
{
// Define required local variable
int position = 0;
int Row = e.row;
string Item = e.item;
// Change the transition item in the Transitions list at the
//row specified, and make sure the currently selected
//transition remains selected
position = lstTransitions.SelectedIndex;
lstTransitions.Items.RemoveAt(Row);
lstTransitions.Items.Insert(Row, Item);
lstTransitions.SelectedIndex = position;
}
private void OCXBatchCtrl_ViewTransitionTransDelete(object sender,
AxOCXBATCHLib._DOcxBatchEvents_ViewTransitionTransDeleteEvent e)
{
// Define required local variable
```

```
int position = 0;
int Row = e.row;
// Delete the transition from the Transitions list at the
//row specified, and if possible, make sure the currently
//selected transition remains selected
position = lstTransitions.SelectedIndex;
lstTransitions.Items.RemoveAt(Row);
if (position < Row)
{
lstTransitions.SelectedIndex = position;
}
else
{
lstTransitions.SelectedIndex = position - 1;
}
}
private void OCXBatchCtrl_ViewTransitionTransSelect(object sender,
AxOCXBATCHLib._DOcxBatchEvents_ViewTransitionTransSelectEvent e)
{
// Define required local variable
int Row = e.row;
// Set the transition force button accordingly whenever a
//transition is selected in the Transitions list
if (Row == -1)
{
cmdForce.Enabled = false;
}
else
{
cmdForce.Enabled = true;
}
}
private void OCXBatchCtrl_ViewTransitionTransUpdate(object sender, EventArgs
e)
{
// Define required local variable
int i = 0;
// Update the entire Transitions list
lstTransitions.Items.Clear();
for (i = 1; i <= OCXBatchCtrl.ViewTransitionTransGetNumItems(); i++)
{
lstTransitions.Items.Add(OCXBatchCtrl.ViewTransitionTransGetItem
((short(i - 1)));
}
if (lstTransitions.Items.Count > 0)
{
lstTransitions.SelectedIndex = 0;
}
else
{
lstTransitions.SelectedIndex = -1;
}
Cursor.Current = Cursors.Default;
}
```

CHAPTER 5

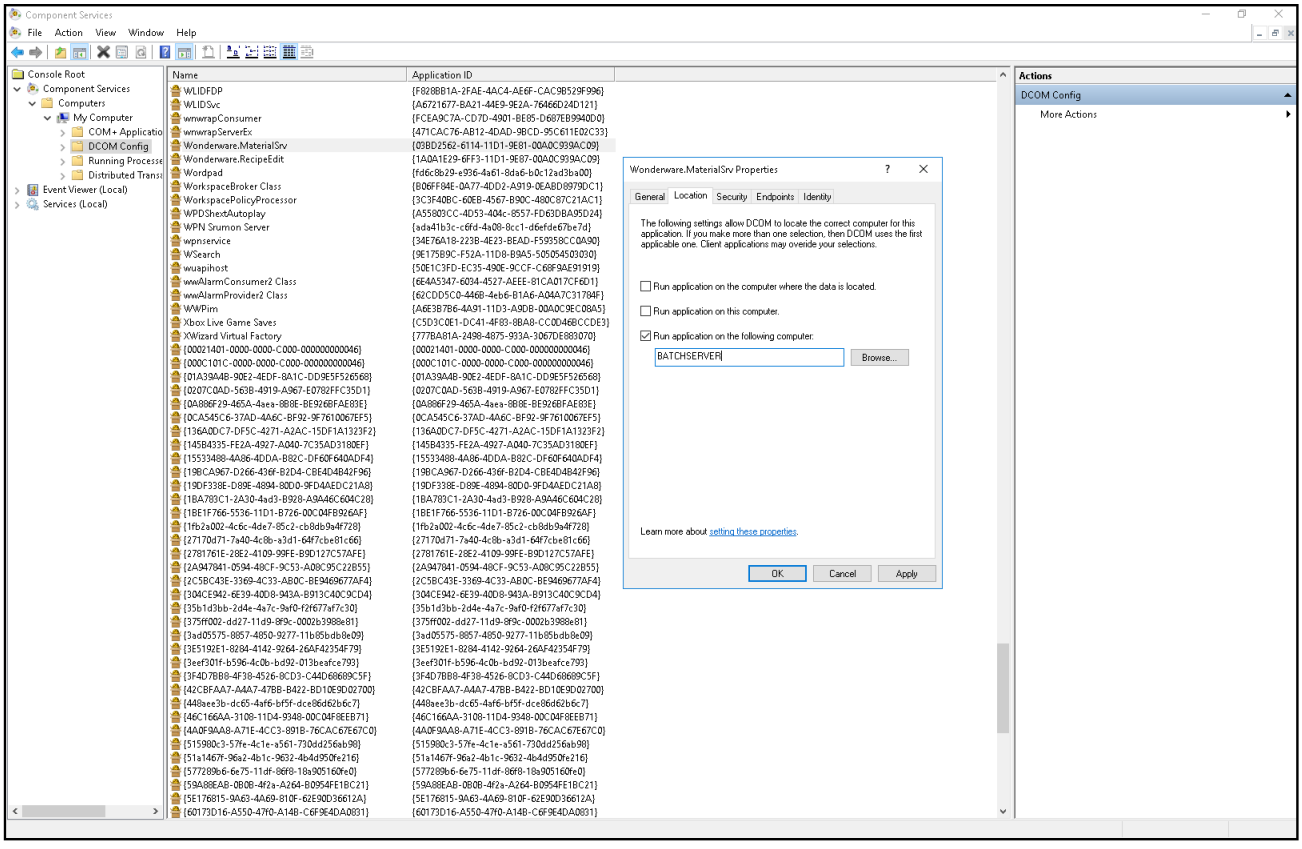
Material Database Automation Server

The Material database automation server (MaterialSrv.exe) is a set of object classes that you can use to create custom applications that read and modify the batch control system material database.

Refer to the appropriate COM-based environment documentation for specific details on installing automation servers within that environment.

To run a custom application that incorporates functions from the materials server on an Batch Management client, you must configure the DCOMCNFG utility as shown in the following figures to direct the application to the materials server on the Batch Management Server.

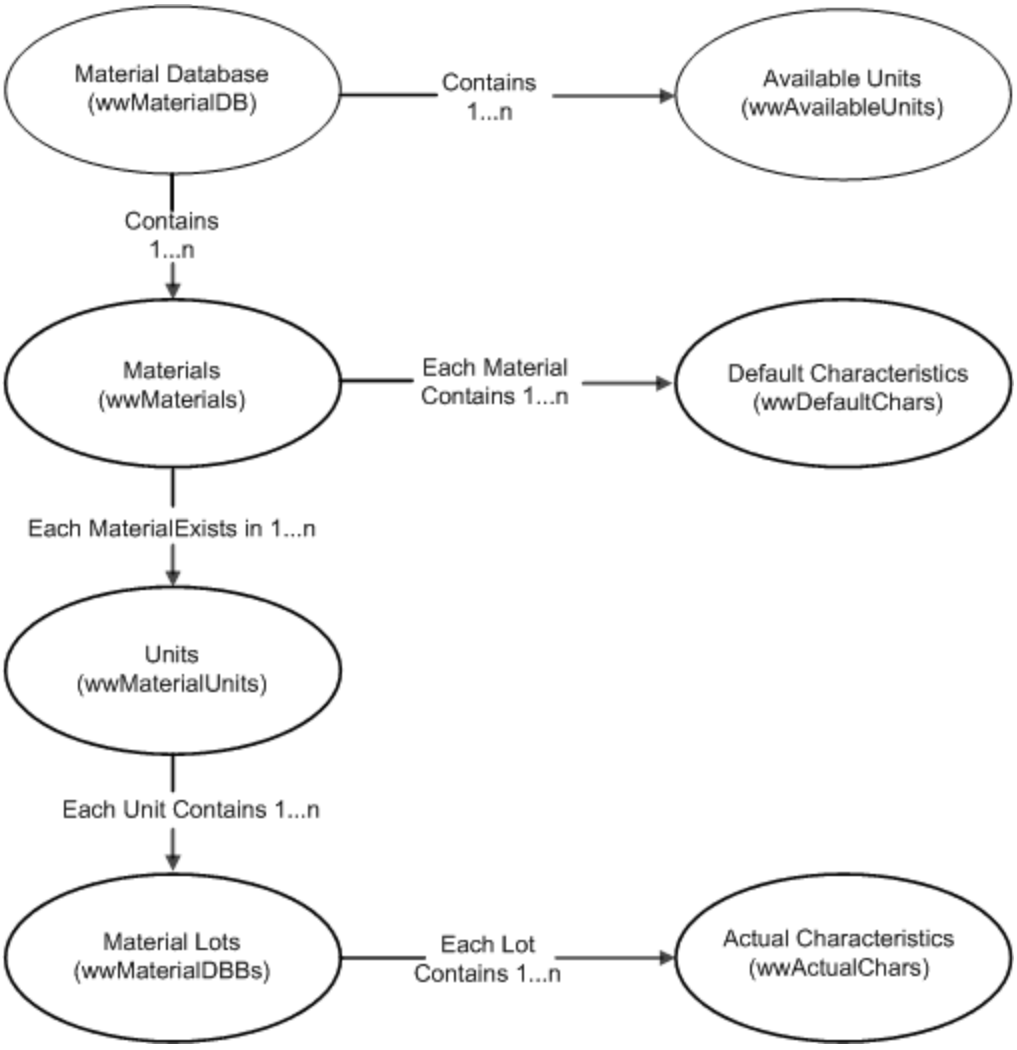
Note: To run the Materials Database Automation Severs on a 64-bit machine, you must configure the DCOMCNFG utility in 32-bit version. To do this, use the command **mmc comexp.msc /32** in the system Run prompt.



In This Chapter

Object Classes

The material automation server is comprised of several object classes that are related to one another. Understanding this relationship is critical to use the server effectively. The following diagram illustrates the collection of objects and shows the relationship between the object classes.



Material Database Object Class

The highest object class is the material database (wwMaterialDB). You must define an object of this type to access the Batch Management Material database. From this object, you can add and query materials, and you can query all the units in the process model that are available for assignment.

A property of the material database object class is the material defined in the database (wwMaterials). This is a collection object from which you can determine the total number of materials. The materials collection contains one-to-many individual material objects (wwMaterial).

Each material object (wwMaterial) contains properties that enable you to query material details, such as the material ID, name, description, unit of measure, deviations, and total quantity. In addition to the properties, each material object provides methods for defining one-to-many default characteristics and assigning the material to one-to-many units.

Default Characteristics Object Class

The default characteristics object class (wwDefaultChars) is a collection object that contains the total number of default characteristics defined for the material. The default characteristics collection contains one-to-many individual default characteristic objects (wwDefaultChar) that permit deleting and changing characteristic values.

Material Units Object Class

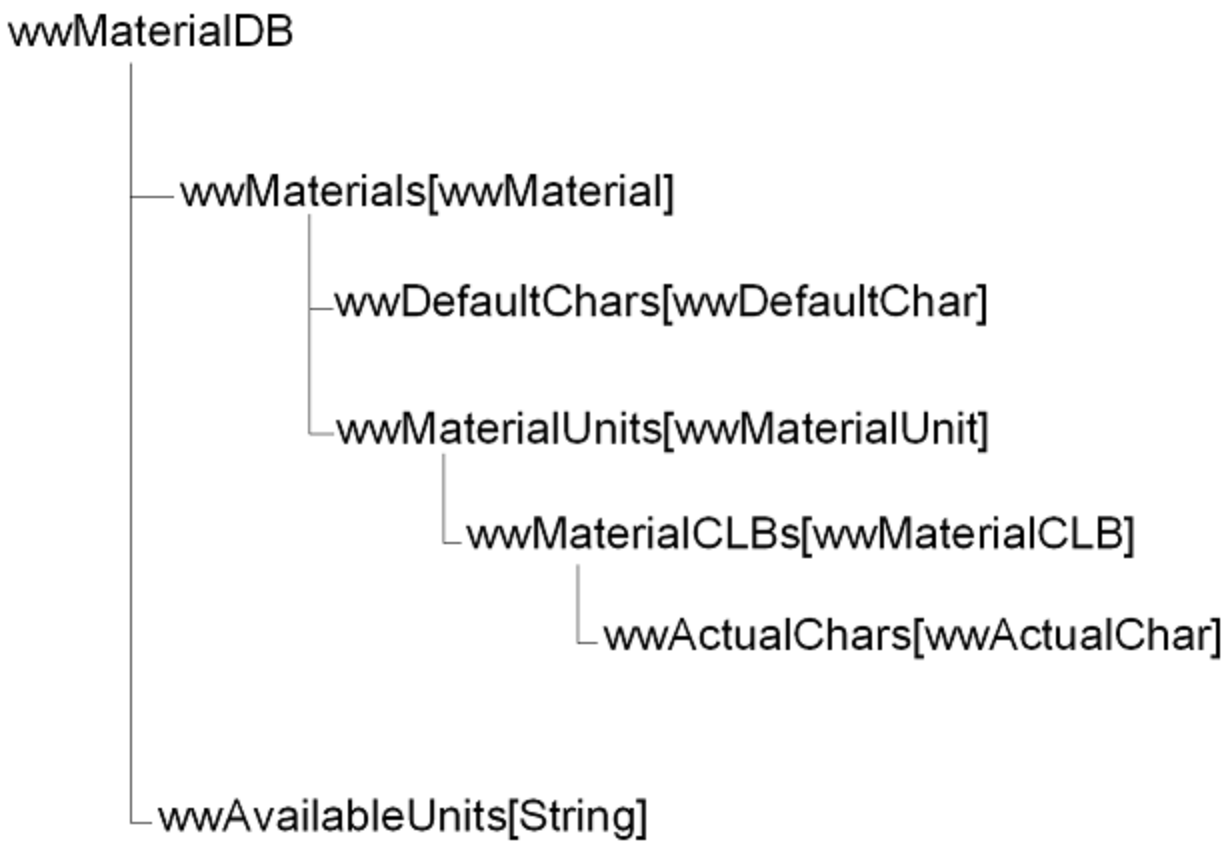
The material units object class (wwMaterialUnits) is a collection object that contains the total number of units to which the material has been assigned. The material units collection contains one-to-many individual material unit objects (wwMaterialUnit) that permit adding and querying lot tracking information.

For each material unit object (wwMaterialUnit), there is a collection of lot tracking information (wwMaterialCLBs). This collection contains one-to-many individual lot tracking objects (wwMaterialCLB) that permit querying lot tracking information and modifying actual characteristics.

Actual Characteristics Object Class

The actual characteristics object class (wwActualChars) is a collection object that contains the total number of actual characteristics defined for a specific lot (wwMaterialCLB) of the material. The actual characteristics collection contains one-to-many individual actual characteristic objects (wwActualChar) that permit changing characteristic values.

The following chart is another representation of the material server class hierarchy.



Material Server Class Library

The following section is an alphabetical listing of all of the object classes available with the material server. Included with each class is a description of the object, details about the properties and methods available with each, and examples of code written using C# that show how the objects can be used.

wwActualChar Object Class

You can use the wwActualChar object to access or modify the value of a characteristic for a specific material lot. These objects are populated when you call the QueryActualChars method on a wwMaterialCLB object. You can retrieve a reference to one of these objects by using the Item property of the wwActualChars collection object.

Properties

The following properties are available for the wwActualChar class.

Name

This property is the name of the characteristic.

Data Type: String

Access: RW

Value

This property is the value of the lot specific characteristic.

Data Type: String

Access: RW

Methods

This section describes the methods available for the wwActualChar class.

ChangeValue()

This method changes the value of the characteristic. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = ChangeValue (NewValue)
```

Parameters

New Value
Data Type: String [16]
New value for the actual characteristic

Example

The following sample C# code illustrates how to define, query, and use the wwActualChar object.

The example searches for Lot_123 of the Vinegar material in the BulkTk1 unit and changes the value of the actual pH characteristic to 4.3.

```
private void ChangeButton_Click(object sender, EventArgs e)
        {
            wwMaterialUnits LocMatUnits;
            wwActualChar MatActualChar= null;
            wwActualChars MatActualChars;
LocMatUnits = GetMatUnits();
```

```
            if (LocMatUnits == null)
                return;
             for (int ut = 1; ut <= LocMatUnits.Count; ut++)
            {
                MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
                if (UnitsLsb.Items.Count > 0)
                if (MatUnit.UnitName == UnitsLsb.SelectedItem.ToString())
                    break;
            }
            MatUnit.QueryCLBs();
            MatClbs = (wwMaterialCLBs)MatUnit.MaterialCLBs;
            for (int i = 1; i <= MatClbs.Count; i++)
            {
                MatClb = (wwMaterialCLB)MatClbs.Item(i);
                if (i == (System.Convert.ToInt16(
lotTrackingLsv.SelectedItems[0].SubItems[0].Text)))
                    break;
            }
              DateTime dttm = new DateTime(MatClb.Year, MatClb.Month,
MatClb.Day);
              dateTimePicker1.Value = dttm;
              MatClb.QueryActualChars();
              MatActualChars =(wwActualChars) MatClb.ActualChars;
                for(short k=1; k<= MatActualChars.Count; k++)
                {
                    short temp =
System.Convert.ToInt16(ActualcharsLsv.SelectedItems[0].SubItems[0].Text);
                    MatActualChar = (wwActualChar)MatActualChars.Item(k);
                    if(k == temp)
                    break;
                }
                if (MatActualChar != null)
                 MatActualChar.ChangeValue(ActualValTxt.Text);
                if (ActualcharsLsv.Items.Count > 0)
                    ActualcharsLsv.TopItem.Selected = true;
                LoadLotTrackingValues();
}
Private void SetLotTrackingInfo
{
wwMaterialUnits LocMatUnits;
wwActualChar MatActualChar;
wwActualChars MatActualChars;
wwMaterials mats;
wwMaterial Material;
wwMaterialUnit MatUnit = null;
wwMaterialCLBs MatClbs;
wwMaterialCLB MatClb;
// LotTrackingLsv is the listview
 string strCampaign = lotTrackingLsv.SelectedItems[0].SubItems[1].Text
 string strLot = lotTrackingLsv.SelectedItems[0].SubItems[2].Text;
 string strBatch = lotTrackingLsv.SelectedItems[0].SubItems[3].Text;
                if (strLot.Length > 0)
                {
                    CampaignTxt.Text = strCampaign;
                    LotText.Text = strLot;
                    BatchTxt.Text = strBatch;
                }
                LocMatUnits = GetMatUnits(); // given below function
```

```
                if (LocMatUnits == null)
                    return;
                for (int ut = 1; ut <= LocMatUnits.Count; ut++)
                {
                    MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
// UnitsLsb is  listbox
                    if (UnitsLsb.Items.Count > 0)
  if (MatUnit.UnitName == UnitsLsb.SelectedItem.ToString())
                            break;
                }
                MatUnit.QueryCLBs();
                MatClbs = (wwMaterialCLBs)MatUnit.MaterialCLBs;
                for (int i = 1; i <= MatClbs.Count; i++)
                {
                    MatClb = (wwMaterialCLB)MatClbs.I
    if (i ==
System.Convert.ToInt16(lotTrackingLsv.SelectedItems[0].SubItems[0].Text))
                        break;
                }
                QualityText.Text = MatClb.Quantity.ToString();
DateTime dttm = new DateTime(MatClb.Year, MatClb.Month, MatClb.Day);
                dateTimePicker1.Value = dttm;
                MatClb.QueryActualChars();
                MatActualChars = (wwActualChars)MatClb.ActualChars;
                for (int k = 1; k <= MatActualChars.Count; k++)
                {
                    MatActualChar = (wwActualChar)MatActualChars.Item(k);
                    ListViewItem listItem = new ListViewItem(k.ToString());
                    listItem.SubItems.Add(MatActualChar.Name);
                    listItem.SubItems.Add(MatActualChar.Value.ToString());
                    ActualcharsLsv.Items.Add(listItem);             }}
private wwMaterialUnits GetMatUnits()
{
  wwMaterialUnits MatUnits=null;
if (MatStatus.GetMaterialType() == "Ingradiants")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
if (MatStatus.GetMaterialType() == "Finished Goods")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeFinishedGood); if
(MatStatus.GetMaterialType() == "Intermediates")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIntermediate);
if (MatStatus.GetMaterialType() == "Other products")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeOther);
if (MatStatus.GetMaterialType() == "All")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeAll);
     mdbc.QueryMaterialsById("r00-454-4545")
     mats = (wwMaterials)mdbc.Materials;
      if (mats.Count != 0)
          {
             Material = (wwMaterial)mats.Item(1);
             Material.QueryMaterialUnits();
             MatUnits = (wwMaterialUnits)Material.MaterialUnits;
          }
        return MatUnits;
}
```

wwActualChars Object Class

This object is a collection of wwActualChar objects. This collection is populated when you call the QueryActualChars method on a wwMaterialCLB object. You can retrieve a reference to this object from the ActualChars property on the wwMaterialCLB object.

Properties

The following property is available for the wwActualChars class.

Count

This property represents the number of characteristics defined for material.

Data Type: Long

Access: R

Methods

This section describes the methods available for the wwActualChars class.

Item()

This method returns a specific wwActualChar object based on the passed index.

Syntax

```
Object = ActCharVar.Item (Index)
```

Parameters

Index
Data Type: Long
The specific entry in the list of actual characteristics. Lists begin with 1.

Example

The following sample C# code illustrates how to define, query, and use the wwActualChars object.

The example populates a list of ingredients, populates a list of units to which the material has been assigned whenever a material is selected, populates a list of lot tracking information defined for the selected unit, and finally populates a list of actual characteristics for the selected lot tracking instance. The global variables are defined so that they are available throughout the entire program.

```
private void SetLotTrackingInfo()
{
lotTrackingLsv.Items.Clear();
wwMaterialUnits LocMatUnits;
LocMatUnits = GetMatUnits();// you get this function in the above page
if(LocMatUnits== null)
     return;
 // UnitsLsb is the list box containing units
 for (int ut = 1; ut <= LocMatUnits.Count; ut++)
   {
      MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
         if (UnitsLsb.Items.Count > 0)
if (MatUnit.UnitName == UnitsLsb.SelectedItem.ToString())
                    break;
   }
  if (LocMatUnits.Count == 0)
          return;
  MatUnit.QueryCLBs();
  MatClbs = (wwMaterialCLBs)MatUnit.MaterialCLBs;
  for (int i = 1; i <= MatClbs.Count; i++)
```

```
     {
       MatClb = (wwMaterialCLB)MatClbs.Item(i);
       ListViewItem listItem = new ListViewItem(i.ToString());
       listItem.SubItems.Add(MatClb.CampaignID);
       listItem.SubItems.Add(MatClb.LotID);
       listItem.SubItems.Add(MatClb.BatchID);
       lotTrackingLsv.Items.Add(listItem);
      }
}
private void SetAvailableUnits()
{
  AvaliableUnitsLsb.Items.Clear();
AvaliableUnitsLsb.Show();
if (MatStatus.GetMaterialType() == "Ingradiants")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
if (MatStatus.GetMaterialType() == "Finished Goods")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeFinishedGood);
if (MatStatus.GetMaterialType() == "Intermediates")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIntermediate);
if (MatStatus.GetMaterialType() == "Other products")
      mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeOther);
if (MatStatus.GetMaterialType() == "All")
    mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeAll);
mdbc.QueryMaterialsById(MatStatus.GetMaterialId());
mats = (wwMaterials)mdbc.Materials;
if (mats.Count == 0)
    return;
Material = (wwMaterial)mats.Item(1);
MatDb.QueryAvailableUnits();
MatDBunits = (wwAvailableUnits)MatDb.AvailableUnits;
for(int i=1; i <= MatDBunits.Count; i++)
   AvaliableUnitsLsb.Items.Add( MatDBunits.Item(i));
}
Private void SetLotTrackingInfo
{
wwMaterialUnits LocMatUnits;
wwActualChar MatActualChar;
wwActualChars MatActualChars;
wwMaterials mats;
wwMaterial Material;
wwMaterialUnit MatUnit = null;
wwMaterialCLBs MatClbs;
wwMaterialCLB MatClb;
// LotTrackingLsv is the listview
 string strCampaign = lotTrackingLsv.SelectedItems[0].SubItems[1].Text
 string strLot = lotTrackingLsv.SelectedItems[0].SubItems[2].Text;
 string strBatch = lotTrackingLsv.SelectedItems[0].SubItems[3].Text;
                if (strLot.Length > 0)
                {
                    CampaignTxt.Text = strCampaign;
                    LotText.Text = strLot;
                    BatchTxt.Text = strBatch;
                }
                LocMatUnits = GetMatUnits(); // given below function
                if (LocMatUnits == null)
                    return;
                for (int ut = 1; ut <= LocMatUnits.Count; ut++)
                {
```

```
                    MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
// UnitsLsb is  listbox
                    if (UnitsLsb.Items.Count > 0)
  if (MatUnit.UnitName == UnitsLsb.SelectedItem.ToString())
                            break;
                }
                MatUnit.QueryCLBs();
                MatClbs = (wwMaterialCLBs)MatUnit.MaterialCLBs;
                for (int i = 1; i <= MatClbs.Count; i++)
                {
                    MatClb = (wwMaterialCLB)MatClbs.Item(i);
                    if (i ==
System.Convert.ToInt16(lotTrackingLsv.SelectedItems[0].SubItems[0].Text))
                        break;
                }
                QualityText.Text = MatClb.Quantity.ToString();
          DateTime dttm = new DateTime(MatClb.Year, MatClb.Month, MatClb.Day);
                dateTimePicker1.Value = dttm;
                MatClb.QueryActualChars();
                MatActualChars = (wwActualChars)MatClb.ActualChars;
                for (int k = 1; k <= MatActualChars.Count; k++)
                {
                    MatActualChar = (wwActualChar)MatActualChars.Item(k);
                    ListViewItem listItem = new ListViewItem(k.ToString());
                    listItem.SubItems.Add(MatActualChar.Name);
listItem.SubItems.Add(MatActualChar.Value.ToString());
                    ActualcharsLsv.Items.Add(listItem);
}
}
```

wwAvailableUnits Object Class

This object is a collection of available unit strings. You can assign a material to any unit that is available. This collection is populated when you call the QueryAvailableUnits method on a wwMaterialDB object. You can retrieve a reference to this object from the AvailableUnits property on the wwMaterialDB object.

Properties

The following property is available for the wwAvailableUnits class.

Count

This property represents the number of units available for assignment.

Data Type: Long

Access: R

Methods

This section describes the methods available for the wwAvailableUnits class.

Item()

This method returns a specific unit name as a string based on the passed index.

Syntax

```
Object = AvailUnitsVar.Item (Index)
```

Parameters

Index
Data Type: Long

The specific entry in the list of actual characteristics. Lists begin with 1.

Example

The following sample C# code illustrates how to define, query, and use the wwAvailableUnits object.

This example populates a list of available units for a material database.

```
private void SetAvailableUnits()
{
      wwMaterialUnits LocMatUnits;
      wwMaterials mats;
      wwMaterial Material;
      wwMaterialUnit MatUnit = null;
      wwMaterialCLBs MatClbs;
      wwMaterialCLB MatClb;
      AvaliableUnitsLsb.Items.Clear();
      AvaliableUnitsLsb.Show();
      if (MatStatus.GetMaterialType() == "Ingradiants")
      mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
      mdbc.QueryMaterialsById(MatStatus.GetMaterialId());
      mats = (wwMaterials)mdbc.Materials;
      if (mats.Count == 0)
      return;
      Material = (wwMaterial)mats.Item(1);
      MatDb.QueryAvailableUnits();
      MatDBunits = (wwAvailableUnits)MatDb.AvailableUnits;
      for(int i=1; i <= MatDBunits.Count; i++)
      AvaliableUnitsLsb.Items.Add( MatDBunits.Item(i));
}
```

wwDefaultChar Object Class

You can use the wwDefaultChar object to access or modify a default characteristic for a particular material. These objects are populated when you call the Query DefaultChars method on a wwMaterial object. You can retrieve a reference to one of these objects by using the Item property of the wwDefaultChars collection object.

Properties

The following properties are available for the wwDefaultChar class.

Name

This property contains the name of the characteristic.

Data Type: String

Access: R

Value

This property contains the default value of material characteristic.

Data Type: String

Access: R

ValueType

This property contains the default characteristic type enumeration. For ValueType enumeration constants, see *"Enumerated Constants"*.

Data Type: Integer

Access: R

Methods

This section describes the methods available for the wwDefaultChar class.

Change()

This method changes the value of the default characteristic. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = DefaultCharVar.Change (Val, ValType)
```

Parameters

Val
Data Type: String
New value for default characteristics

ValType
Data Type: Integer
Default characteristic type enumeration. For ValType enumeration constants, see *"Enumerated Constants"*.

Delete()

This method deletes the default characteristic. This action also deletes the corresponding actual characteristic. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = DefaultCharVar.Delete ()
```

Example

The following sample C# code illustrates how to define, query, and use the wwDefaultChar object.

The example deletes all the default characteristics defined for the Water material. The global variables are defined so that they are available throughout the entire program.

```
private void deleteDefaultChar()
{
      short sretval = 0;
      wwDefaultChar defaultchar;
      wwDefaultChars defaultchars;
      wwMaterialUnits LocMatUnits;
      LocMatUnits = GetMatUnits(); // find the this function in above  page
      Material.QueryDefaultChars();
      defaultchars = (wwDefaultChars)Material.DefaultChars;
      int temp =
System.Convert.ToInt16(CharectaristicsLsv.SelectedItems[0].SubItems[0].Text)
;
      defaultchar = (wwDefaultChar)defaultchars.Item(temp);
      sretval = defaultchar.Delete();
}
```

wwDefaultChars Object Class

This object is a collection of wwDefaultChar objects. This collection is populated when you call the QueryDefaultChars method on a wwMaterial object. You can retrieve a reference to this object from the DefaultChars property on the wwMaterial object.

Properties

The following property is available for the wwDefaultChars class.

Count

This property represents the number of characteristics defined for a material.

Data Type: Long

Access: R

Methods

This section describes the methods available for the wwDefaultChars class.

Item()

This method returns a specific wwDefaultChar object based on the passed index.

Syntax

```
Object = DefaultCharsVar.Item (Index)
```

Parameters

Index
Data Type: Long
The specific entry in the list of actual characteristics. Lists begin with 1.

Example

The following sample C# code illustrates how to define, query, and use the wwDefaultChars object.

```
private wwMaterialUnits GetMatUnits()
{
      wwMaterialUnits MatUnits=null;
      if (MatStatus.GetMaterialType() == "Ingradiants")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
      if (MatStatus.GetMaterialType() == "Finished Goods")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeFinishedGood); if
      (MatStatus.GetMaterialType() == "Intermediates")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIntermediate);
      if (MatStatus.GetMaterialType() == "Other products")
      mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeOther);
      if (MatStatus.GetMaterialType() == "All")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeAll);
       mdbc.QueryMaterialsById("r00-454-4545")
      mats = (wwMaterials)mdbc.Materials;
      if (mats.Count != 0)
          {
             Material = (wwMaterial)mats.Item(1);
             Material.QueryMaterialUnits();
             MatUnits = (wwMaterialUnits)Material.MaterialUnits;
          }
        return MatUnits;
}

private void ChangeButton_Click(object sender, EventArgs e)
{
  short sretval = 0;
  wwDefaultChar defaultchar;
  wwDefaultChars defaultchars;
  wwMaterialUnits LocMatUnits;
```

```
  LocMatUnits = GetMatUnits();// u can find this function in the above
  Material.QueryDefaultChars();
  defaultchars = (wwDefaultChars)Material.DefaultChars;
      int temp =
System.Convert.ToInt16(CharectaristicsLsv.SelectedItems[0].SubItems[0].Text)
      defaultchar = (wwDefaultChar)defaultchars.Item(temp);
      if(IntegerRdb.Checked)
     sretval = defaultchar.Change(DefaultValueTxt.Text, 1);
      if (StringRdb.Checked)
     sretval = defaultchar.Change(DefaultValueTxt.Text, 2);
      if (RealRdb.Checked)
      sretval = defaultchar.Change(DefaultValueTxt.Text, 0);
```

wwMaterial Object Class

You can use the wwMaterial object to access or modify a material in the material database. These objects are populated when you call the QueryMaterials method on a wwMaterialDB object. You can retrieve a reference to one of these objects can be retrieved by using the Item property of the wwMaterials collection object.

Properties

The following properties are available for the wwMaterial class.

Id

This property uniquely identifies the material.

Data Type: String

Access: R

Name

This property contains the name of the material.

Data Type: String

Access: R

Description

This property contains a description of the material.

Data Type: String

Access: R

Type

This property contains the material type enumeration. For Type enumeration constants, see *"Enumerated Constants"*.

Data Type: Integer

Access: R

UnitOfMeasure

This property contains the material unit of measure.

Data Type: String

Access: R

HiDev

This property contains the high deviation value for material.

Data Type: Double

Access: R

LoDev

This property contains the low deviation value for material.

Data Type: Double

Access: R

DefaultChars

This property contains the wwDefaultChars reference.

Data Type: Object

Access: R

MaterialUnits

This property contains the wwMaterialUnits reference.

Data Type: Object

Access: R

TotalMaterialQty

This property contains the total quality of material in all units and lots.

Data Type: Double

Access: R

Methods

This section describes the methods available for the wwMaterial class.

AddDefChar()

This method adds a new default characteristic to the material. A corresponding actual characteristic is also created. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialVar.AddDefChar (CharName, Val, ValType)
```

Parameters

CharName
Data Type: String
Name of characteristic

Val
Data Type: String
Default value of characteristic

ValType
Data Type: Integer
Default characteristic type enumeration. For ValType enumeration constants, see *"Enumerated Constants"*.

AddUnit()

This method assigns the material to a new unit. A unit can contain only a single material. You can use the QueryAvailableUnits method on the wwMaterialDB object to obtain a collection of available units. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialVar.AddUnit (Unit, ConflictType)
```

Parameters

Unit
Data Type: String
Name of unit

ConflictType
Data Type: Integer
This field is required but is not used.
0 = Resolve by date
1 = Resolve by operator

Change()

This method changes the simple properties associated with the material. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialVar.Change (Name, Desc, Type, UofM, HiDev, LoDev)
```

Parameters

Name
Data Type: String
Name of new material

Desc
Data Type: String
Description of new material

Type
Data Type: Integer
Enumeration of new material. For Type enumeration constants, see *"Enumerated Constants"*.

UofM
Data Type: String
Unit of measure of new material

HiDev
Data Type: Double
High deviation of new material

LoDev
Data Type: Double
Low deviation of new material

Delete()

This method deletes the material. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialVar.Delete ()
```

QueryDefaultChars()

This method updates the DefaultChars collection with the latest set of default characteristics for the material.

Syntax

```
MaterialVar.QueryDefaultChars ()
```

QueryMaterialUnits()

This method updates the MaterialUnits collection with the latest set of assigned units for the material.

Syntax

```
MaterialVar.QueryMaterialUnits ()
```

Example

The following example C# code shows hot to add a new default characteristic to the material.

```
private void Add_Click(object sender, EventArgs e)
{
 wwMaterial Material;
 wwDefaultChar defaultchar;
 wwDefaultChars defaultchars;
 short sretval = 0;
 wwMaterialUnits LocMatUnits;
 LocMatUnits = GetMatUnits(); // u can find this function in the above
 if(RealRdb.Checked ==true)
 sretval = Material.AddDefChar(NameText.Text, DefaultValueTxt.Text, 0);
 if(IntegerRdb.Checked)
 sretval = Material.AddDefChar(NameText.Text, DefaultValueTxt.Text, 1);
 if(StringRdb.Checked)
 sretval = Material.AddDefChar(NameText.Text, DefaultValueTxt.Text, 2);
 CharectaristicsLsv.Items.Clear();
 LoadCharectaristicValues();
 }
```

wwMaterialCLB Object Class

You can use the wwMaterialCLB object to access or modify a specific lot of material. These objects are populated when you call the QueryCLBs method on a wwMaterialUnit object. You can retrieve a reference to one of these objects by using the Item property of the wwMaterialCLBs collection object.

Properties

The following properties are available for the wwMaterialCLB class.

CampaignID

This property contains the Campaign ID of the lot.

Data Type: String

Access: R

LotID

This property contains the Lot ID of the lot.

Data Type: String

Access: R

BatchID

This property contains the Batch ID of the lot.

Data Type: String

Access: R

Quantity

This property contains the Quantity in the lot.

Data Type: Double

Access: R

Month

This property contains the month that the lot was entered.

Data Type: Integer

Access: R

Day

This property contains the day that the lot was entered.

Data Type: Integer

Access: R

Year

This property contains the year that the lot was entered.

Data Type: Integer

Access: R

ActualChar

This property contains the wwActualChars reference.

Data Type: Object

Access: R

Methods

This section describes the methods available for the wwMaterialCLB class.

ChangeMaterialQuantity()

This method changes the quantity of the material lot entry. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialCLBVar.ChangeMaterialQuantity ( NewQuantity)
```

Parameters

NewQuantity
Data Type: Double
Value of new quantity

Delete()

This method deletes the material assigned to the selected unit. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialCLBVar.Delete ()
```

QueryActualChars()

This method updates the ActualChars collection with the latest set of actual characteristics for the material lot.

Syntax

```
QueryActualChars ()
```

Example

The following sample C# code illustrates how to define, query, and use the wwMaterialCLB object. The example search for the Lot_123 of the Vinegar material in the BulkTk1 unit and changes the quantity.

```
private void GetMaterials(object sender, EventArgs e)
{
wwMaterial Materials;
wwMaterialDBClass mdbc = new wwMaterialDBClass();
wwRecipeClass rc = new wwRecipeClass();
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
mdbc.QueryMaterialsById(MaterialLsv.SelectedItem.ToString());
                //mdbc.QueryMaterialsById(MaterialIdTxt.Text);
mats = (wwMaterials)mdbc.Materials;
Materials = (wwMaterial)mats.Item(1);
MaterialIdTxt.Text = Materials.Id;
MaterialNameTxt.Text = Materials.Name;
short typeval = Materials.Type;
UnitOfMeasureTxt.Text = rc.GetMaterialDBUofM(Materials.Id);
HighDevTxt.Text = Materials.HiDev.ToString();
LowDevTxt.Text = Materials.LoDev.ToString();
MaterialDecriptionTxt.Text = Materials.Description;
}
```

wwMaterialCLBs Object Class

This object is a collection of wwMaterialCLB objects. This collection is populated when you call the QueryCLBs method on a wwMaterialUnit object. You can retrieve a reference to this object from the MaterialCLBs property on the wwMaterialUnit object.

Properties

The following property is available for the wwMaterialCLBs class.

Count

This property represents the number of lots of material in a unit.

Data Type: Long

Access: R

Methods

This section describes the method available for the wwMaterialCLBs class.

Item()

This method returns a specific wwMaterialCLB object based on the passed index.

Syntax

```
Object = MaterialCLBsVar.Item (Index)
```

Parameters

Index
Data Type: Long
The specific entry in the list of lot tracking data. Lists begin with 1.

Example

The following sample C# code illustrates how to define, query, and use the wwMaterialCLBs object.

The example populates a list of ingredients, populates a list of units to which the material has been assigned whenever a material is selected, and finally populates a list of lot tracking information defined for the selected unit. The global variables are defined so that they are available throughout the entire program.

```
private void LoadLotTrackingValues()
{
 wwMaterialUnits LocMatUnits;
 wwActualChar MatActualChar;
 wwActualChars MatActualChars;
short retval = 0;
string strCampaign=lotTrackingLsv.SelectedItems[0].SubItems[1].Text;
string strLot = lotTrackingLsv.SelectedItems[0].SubItems[2].Text;
string strBatch = lotTrackingLsv.SelectedItems[0].SubItems[3].Text;
LocMatUnits = GetMatUnits();
  if (LocMatUnits == null)
       return;
for (int ut = 1; ut <= LocMatUnits.Count; ut++)
     {
        MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
        if (UnitsLsb.Items.Count > 0)
           if (MatUnit.UnitName == "testUnitName1"
                        break;
      }
      MatUnit.QueryCLBs();
      MatClbs = (wwMaterialCLBs)MatUnit.MaterialCLBs;
  for (int i = 1; i <= MatClbs.Count; i++)
      {
       MatClb = (wwMaterialCLB)MatClbs.Item(i);
if(MatClb.LotID= "TestLotid")
                       break;
      }
  retval =MatClb.ChangeMaterialQuantity(99.9); }
```

wwMaterialDB Object Class

You can use the wwMaterialDB object to access or modify information in the material database. This is the only object that a client actually creates. All the other objects defined by the material automation server are created by the server and used as references by the client.

Properties

The following properties are available for the wwMaterialDB class.

Materials

This property contains the wwMaterials reference that is available after one of the **QueryMaterials** methods have been run.

Data Type: Object

Access: R

AvailableUnits

This property contains the wwAvailableUnits reference that is available after the **QueryAvailableUnits** method has been run.

Data Type: Object

Access: R

Methods

This section describes the methods available for the wwMaterialDB class.

AddMaterial()

This method adds a material to the database. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialDBVar.AddMaterial (Id, Name, Desc, Type, UofM, HiDev, LoDev)
```

Parameters

Id
Data Type: String
New material ID

Name
Data Type: String
New material name

Desc
Data Type: String
New material description

Type
Data Type: Integer
New enumeration type. For Type enumeration constants, see *"Enumerated Constants"*.

UofM
Data Type: String
New unit of measure

HiDev
Data Type: Double
New high deviation

LoDev
Data Type: Double
New low deviation

GetMtrlAssignedToUnit()

This method queries the material currently assigned to the specified unit. The Material ID is returned. If no material was assigned to the unit, a blank string is returned.

Syntax

```
String = MaterialDBVar.GetMtrlAssignedToUnit (Unit)
```

Parameters

Unit
Data Type: String
Name of unit for which material is desired

QueryAvailableUnits()

This method updates the AvailableUnits collection with the latest set of unassigned units.

Syntax

```
MaterialDBVar.QueryAvailableUnits ()
```

QueryMaterials()

This method updates the Materials collection with the latest set of materials.

Syntax

```
MaterialDBVar.QueryMaterials ()
```

QueryMaterialsById()

This method updates the Materials collection with the single material that has the specified material identification.

Syntax

```
MaterialDBVar.QueryMaterialsById (MaterialId)
```

Parameters

MaterialId
Data Type: String
Material ID to find

QueryMaterialsByType()

This method updates the Materials collection with the latest set of materials that have the specified type.

Syntax

```
MaterialDBVar.QueryMaterialsByType (MaterialType)
String = MaterialDBVar.GetMtrlAssignedToUnit (Unit)
```

Parameters

MaterialType
Data Type: wwMtrlTypeEnum
Material type enumeration. For Type enumeration constants, see *"Enumerated Constants"*.

TestConnection()

This method tests to see if the fundamental batch services (Environment Manager and the database lock manager) are running. A Long return value provides the status of the services. The value can be interpreted as follows:

0 = Services are not running
1 = Services are running

Syntax

```
ReturnValue = MaterialDBVar.TestConnection ()
```

Example

The following sample C# code illustrates how to define, query, and use the wwMaterialDB object to populate a list of ingredients.

This example bypasses the use of the wwMaterials collection.

```
private void QueryMats()
{
wwMaterialDBClass mdbc = new wwMaterialDBClass();
wwMaterialUnits MatUnits=null;
wwMaterials mats;
wwMaterial Material;
if (MatStatus.GetMaterialType() == "Ingradiants"
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
mdbc.QueryMaterialsById(MatStatus.GetMaterialId())
mats = (wwMaterials)mdbc.Materials;
 if (mats.Count != 0)
  {
     Material = (wwMaterial)mats.Item(1);
```

```
      Material.QueryMaterialUnits();
      MatUnits = (wwMaterialUnits)Material.MaterialUnits;
    }
}
```

wwMaterials Object Class

This object is a collection of wwMaterial objects. This collection is populated when you call the QueryMaterials method on a wwMaterialDB object. You can retrieve a reference to this object from the Materials property on the wwMaterialDB object.

Properties

The following property is available for the wwMaterials class.

Count

This property represents the number of materials in the database.

Data Type: Long

Access: R

Methods

This section describes the method available for the wwMaterials class.

Item()

This method returns a specific wwMaterial object based on the passed index.

Syntax

```
Object = MaterialsVar.Item (Index)
```

Parameters

Index
Data Type: Long
The specific entry in the list of lot tracking data. Lists begin with 1.

Example

The following C# sample code illustrates how to define, query, and use the wwMaterials object to populate a list of finished goods.

```
private void QueryMats()
{
wwMaterialDBClass mdbc = new wwMaterialDBClass();
wwMaterialUnits MatUnits=null;
wwMaterials mats;
wwMaterial Material;
if (MatStatus.GetMaterialType() == "Ingradiants"
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
mdbc.QueryMaterialsById(MatStatus.GetMaterialId())
mats = (wwMaterials)mdbc.Materials;
 if (mats.Count != 0)
  {
     Material = (wwMaterial)mats.Item(1);
     MessageBox.Show(Material.Id);
     MessageBox.Show(Material.Name);
   }
}
```

wwMaterialUnit Object Class

You can use the wwMaterialUnit object to access or modify information specific to the unit assignment for a material. These objects are populated when you call the QueryMaterialUnits method on a wwMaterial object. You can retrieve a reference to one of these objects can be retrieved by using the Item property of the wwMaterialUnits collection object.

Properties

The following properties are available for the wwMaterialUnit class.

UnitName

This property contains the name of the assigned unit.

Data Type: String

Access: R

MaterialCLBs

This property contains the wwMaterialCLBs reference.

Data Type: Object

Access: R

TotalMaterialQty

This property contains the total quantity of material in the unit.

Data Type: Double

Access: R

Methods

This section describes the methods available for the wwMaterialUnit class.

AddMaterialCLB()

This method adds a new lot (CLB) of material to the unit. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialUnitVar.AddMaterialCLB ( CampaignID, LotID, BatchID,
Quantity, Month, Day, Year)
```

Parameters

CampaignID
Data Type: String
New Campaign ID of lot

LotID
Data Type: String
New Lot ID of lot

BatchID
Data Type: String
New Batch ID of lot

Quantity
Data Type: Double
New Quantity in lot

Month
Data Type: Integer
Month lot was entered

Day
Data Type: Integer
Day lot was entered

Year
Data Type: Integer
Year lot was entered

Delete()

This method deletes the material assigned to the selected unit. The return code is a Short integer value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = MaterialUnitVar.Delete ()
QueryCLBs()
```

This method updates the MaterialCLBs collection with the latest set of material lots.

Syntax

```
MaterialUnitVar.QueryCLBs ()
```

Example

The following sample C# code illustrates how to define, query, and use the wwMaterialUnit object.

```
private void AddClbs()
 {
   wwMaterialUnits LocMatUnits;
   short retval = 0;
  LocMatUnits = GetMatUnits();//find the code for this in above page
  for (int ut = 1; ut <= LocMatUnits.Count; ut++)
  {
     MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
     if (UnitsLsb.Items.Count > 0)
        if (MatUnit.UnitName == UnitsLsb.SelectedItem.ToString())
                 break;
   }
DateTime dt = dateTimePicker1.Value;
retval = MatUnit.AddMaterialCLB(CampaignTxt.Text, LotText.Text,
BatchTxt.Text, System.Convert.ToDouble(QualityText.Text), (short)dt.Month,
(short)dt.Day, (short)dt.Year);
}
```

wwMaterialUnits Object Class

This object is a collection of wwMaterialUnit objects. This collection is populated when you call the QueryMaterialUnits method on a wwMaterial object. You can retrieve a reference to this object can be retrieved from the MaterialUnits property on the wwMaterial object.

Properties

The following property is available for the wwMaterialUnits class.

Count

This property represents the number of units.

Data Type: Long

Access: R

Methods

This section describes the method available for the wwMaterialUnits class.

Item()

This method returns a specific wwMaterialUnit object based on the passed index.

Syntax

```
Object = MaterialUnitsVar.Item (Index)
```

Parameters

Index
Data Type: Long
The specific entry in the list of lot tracking data. Lists begin with 1.

Example

The following C# sample code illustrates how to define, query, and use the wwMaterialUnits object.

The example populates a list of ingredients and then populates a list with units to which the material has been assigned whenever an item is selected in the list of ingredients. The global variables are defined so that they are available throughout the entire program.

```
private void SetUnitNames()
 {
            UnitsLsb.Items.Clear();
            AvaliableUnitsLsb.Hide();
            CancelBtn.Hide();
            wwMaterialUnits LocMatUnits;
            LocMatUnits = GetMatUnits();
            if (LocMatUnits == null)
                return;
            for (int ut = 1; ut <= LocMatUnits.Count; ut++)
            {
                MatUnit = (wwMaterialUnit)LocMatUnits.Item(ut);
                UnitsLsb.Items.Add(MatUnit.UnitName);
            }
            if (UnitsLsb.Items.Count > 0)
            UnitsLsb.SelectedIndex = 0;
            MaterialIdTxt.Text = MatStatus.GetMaterialId();
}
private wwMaterialUnits GetMatUnits()
{
  wwMaterialUnits MatUnits=null;
if (MatStatus.GetMaterialType() == "Ingradiants")

mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIngredient);
if (MatStatus.GetMaterialType() == "Finished Goods")
          mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeFinishedGood); if
(MatStatus.GetMaterialType() == "Intermediates")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeIntermediate);
if (MatStatus.GetMaterialType() == "Other products")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeOther);
if (MatStatus.GetMaterialType() == "All")
mdbc.QueryMaterialsByType(wwMtrlTypeEnum.wwTypeAll);
     mdbc.QueryMaterialsById("r00-454-4545")
```

```
    mats = (wwMaterials)mdbc.Materials;
     if (mats.Count != 0)
         {
            Material = (wwMaterial)mats.Item(1);
            Material.QueryMaterialUnits();
            MatUnits = (wwMaterialUnits)Material.MaterialUnits;
         }
       return MatUnits;}
```

Enumerated Constants

The following enumerated constants are available in the COM-based programming environment of choice. The enumerations enable programming to be completed without the direct knowledge of specific integer values.

wwMtrlCharValTypeEnum

The enumerated constant wwMtrlCharValTypeEnum has the following names and values.

Name	Value
wwCharRealVal	0
wwCharIntegerVal	1
wwCharStringVal	2

wwMtrlTypeEnum

The enumerated constant wwMtrlTypeEnum has the following names and values.

Name	Value
wwTypeAll	-1
wwTypeIngredient	0
wwTypeIntermediate	1
wwTypeFinishedGood	2
wwTypeByProduct	3
wwTypeOther	4

Error Return Values

The following error values may be returned from the material automation server.

Number	Description
0	No error
1	Database error
5	Memory error
6	Item not found
7	Invalid high and/or low deviation
8	Invalid material type
9	Item already exists
10	Material not found
12	Unit not found
13	Invalid conflict type
14	Invalid date
15	CLB not found
16	Characteristic not found
17	Conversion error

CHAPTER 6

Recipe Database Automation Server

You can use the Recipe Database Automation Server (RecipeEdit.exe) to read and modify the batch control system recipe database.

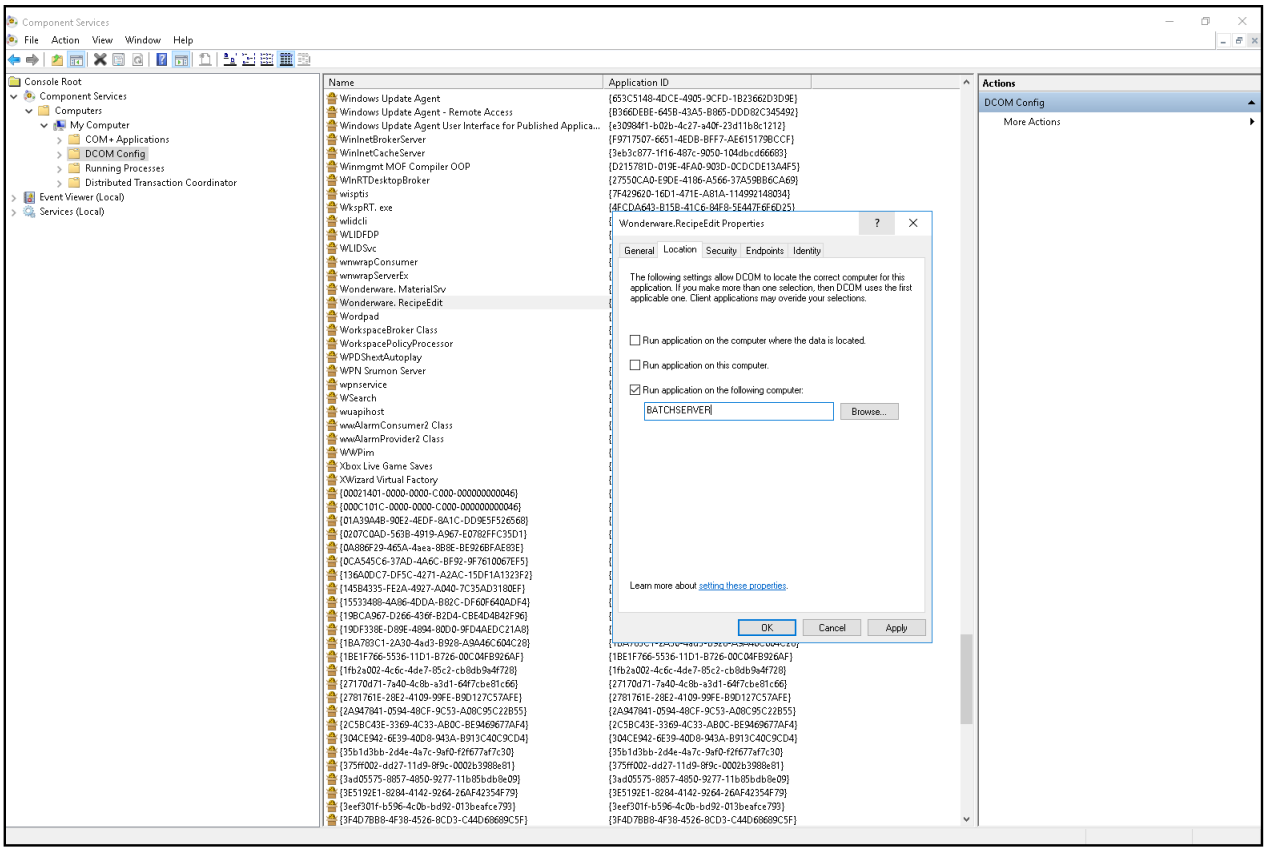
The following list describes some of the activities that you can perform in a custom application using this server:

- Add, change, and delete recipes
- Query and change recipe header information
- Query and change recipe equipment requirements
- Query and change formula inputs defined for a recipe
- Query and change formula outputs defined for a recipe
- Define and modify the formula for a recipe
- Define a recipe procedure

Refer to the appropriate COM-based environment documentation for specific details on installing automation servers within that environment.

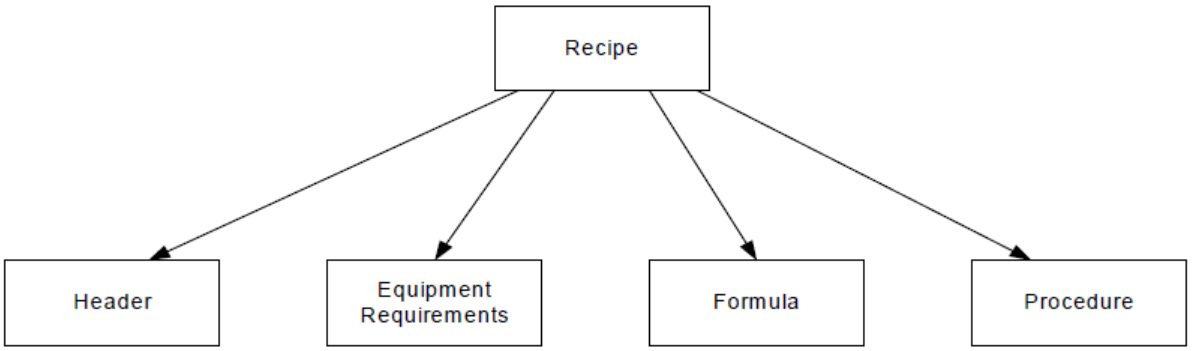
To run a custom application that incorporates functions from the recipe server on a batch client, you must configure the DCOMCNFG utility as shown in the following figures to direct the application to the recipe server on the batch server.

Note: To run the Recipe Database Automation Severs on a 64-bit machine, you must configure the DCOMCNFG utility in 32-bit version. To do this, use the command **mmc comexp.msc /32** in the system Run prompt.



The recipe automation server is comprised of a single object class. This class contains many properties and methods that mimic the functionality of the Recipe Editor accessed from Environment Display. To use the server effectively, it is critical that you thoroughly understand the structure and creation of recipes is critical.

The following diagram illustrates the main components of a recipe.



A recipe has four parts:

- The header contains general information about the recipe such as the name and sizing.
- The equipment requirements define the classes and instances of equipment that are required for the recipe.

- The formula contains the materials and their quantities that are consumed and produced in the recipe.
- The procedure defines the sequence of unit procedures, operations and phases to be processed in the recipe.

Access to each of these parts is possible using this automation server.

You can configure the batch management system for two recipe levels (unit procedures and phases) or three recipe levels (unit procedures, operations, and phases). The Recipe Database Automation Server provides functions to access all parts of the recipe regardless of the number of levels that you define. It is the responsibility of the programmer to account for the number of recipe levels defined and use the appropriate functions accordingly. You can use the MiscGet2Levels method available with the Batch Management ActiveX Control to determine the number of recipe levels defined for the system.

For more information on the Batch Management ActiveX control, see Chapter 4, *Batch Management ActiveX Control* on page 43.

In This Chapter

Recipe Server Class Library

Only one object class is available with the recipe server. The following section provides an alphabetical listing of all of the methods and properties available within this object class including a description of each. Also, code examples written using Visual Basic show how the object can be used.

wwRecipe Object Class

Using the wwRecipe object, information in the batch recipe database can be accessed and modified. This is the only object available in this server and is the only object that a client actually creates. All methods and properties are accessed from this class.

Properties

The following properties are available for the wwRecipe class.

DefaultBatchSize

This property designates the default recipe size.

Data Type: Long

Access: RW

ErrorMessage

This property defines the error string.

Data Type: String

Access: R

GetRecipes

This property defines the list of recipe IDs.

Data Type: Object

Access: R

MaximumBatchSize

This property designates the maximum size of the recipe.

Data Type: Long

Access: RW

MinimumBatchSize

This property designates the minimum size of the recipe.

Data Type: Long

Access: RW

ProductId

This property defines the product ID of the recipe.

Data Type: String

Access: RW

ProductName

This property designates the product name of the recipe.

Data Type: String

Access: RW

RecipeId

This property designates the recipe ID.

Data Type: String

Access: RW

RecipeName

This property designates the name of the recipe.

Data Type: String

Access: RW

Methods

This section describes the methods available for the wwRecipe class.

AddAllocateProcessPhase()

This method adds an allocation process phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddAllocateProcessPhase ( ProcessInstance, PhaseLabel)
```

Parameters

ProcessInstance
Data Type: String
Process instance in recipe

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called.

AddAllocateTransferPhase()

This method adds an allocation transfer phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddAllocateTransferPhase ( TransferInstance,
PhaseLabel)
```

Parameters

TransferInstance
Data Type: String
Transfer instance in recipe

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called.

AddInputMaterials()

This method adds materials to the recipe input material list. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddInputMaterials ( MaterialInputIds)
```

Parameters

MaterialInputId
Data Type: Object (string array)
Input material ID

AddOperation()

This method adds an operation to the focal unit procedure. The operation is added below the current focal operation. Focus is automatically on the operation added. It is not necessary to call the SetFocusOperation method. The return code is a Boolean value that can be used for error handling.

Although not required, operation names should be unique to allow for easy access and changing operation focus.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddOperation ( Name )
```

Parameters

Name
Data Type: String
Name of created operation

AddOutputMaterials()

This method adds materials to the recipe output material list. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddOutputMaterials ( MaterialOutputIds)
```

Parameters

MaterialOutputId
Data Type: Object (string array)
Output material ID

AddProcessClass()

This method adds a process class from the Process Model database to the process class list in the recipe. A default process instance of the same name is created and added to the recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddProcessClass (ProcessClass)
```

Parameters

ProcessClass
Data Type: String
Process class to add

AddProcessInstance()

This method adds a unique process instance to a process class. Process instances are defined to be unique not just in their process class but across all classes. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddProcessInstance ( ProcessClass, ProcessInstance)
```

Parameters

ProcessClass
Data Type: String
Process class in recipe

ProcessInstance
Data Type: String
Unique instance name

AddProcessPhase()

This method adds a process phase to the focal operation. This phase must be defined under the operation's process class. Focus is automatically on the phase added. It is not necessary to call the SetFocusPhase method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddProcessPhase (PhaseName,
PhaseLabel)
```

Parameters

PhaseName
Data Type: String
Process phase name

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called.

AddReleaseProcessPhase()

This method adds a release process phase. Focus is automatically on the phase added. It is not necessary to call the SetFocusPhase method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddReleaseProcessPhase ( ProcessInstance, PhaseLabel)
```
Parameters

ProcessInstance
Data Type: String
Process instance in recipe

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called

AddReleaseTransferPhase()

This method adds a release transfer phase. Focus is automatically on the phase added. It is not necessary to call the SetFocusPhase method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.
Syntax

```
ReturnCode = RecipeVar.AddReleaseTransferPhase ( TransferInstance, PhaseLabel)
```
Parameters

TransferInstance
Data Type: String
Transfer instance in recipe

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called.

AddToOpLibrary()

This method adds the current focal operation to the operation library. All equipment and materials are stored with the operation. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.
Syntax

```
ReturnCode = RecipeVar.AddToOpLibrary (Name, Comments)
```
Parameters

Name
Data Type: String
Name of the operation as it is to appear in the operation library

Comments
Data Type: String
Comments to be stored with the operation in the library

AddToUpLibrary()

This method adds the current focal unit procedure to the unit procedure library. All equipment and materials are stored with the unit procedure. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.
Syntax

```
ReturnCode = RecipeVar.AddToUpLibrary (Name, Comments)
```
Parameters

Name
Data Type: String
Name of the unit procedure as it is to appear in the operation library

Comments
Data Type: String
Comments to be stored with the unit procedure in the library

AddTransferInstance()

This method adds a transfer instance to a transfer class. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddTransferInstance ( TransferClass, InstanceName,
SourceInstance, DestinationInstance)
```

Parameters

TransferClass
Data Type: String
Transfer class in recipe

InstanceName
Data Type: String
Unique transfer instance name

SourceInstance
Data Type: String
Process instance for source class

DestinationInstance
Data Type: String
Process instance for destination class

AddTransferPhase()

This method adds a transfer phase to the focal operation. The transfer instance and phase must associated with the process class of the operation. The process class is either the source or destination class for the associated transfer instance. Focus is automatically on the phase added. It is not necessary to call the SetFocusPhase method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddTransferPhase ( TransferInstance, TransferPhase,
PhaseLabel)
```

Parameters

TransferInstance
Data Type: String
Instance of transfer class

TransferPhase
Data Type: String
Phase of transfer class

PhaseLabel
Data Type: String
The phase label is generated automatically by the recipe server and returned when this method is called.

AddUnitProcedure()

This method adds a unit procedure to the recipe procedure. The unit procedure is added below the current focal unit procedure. Focus is automatically on the unit procedure added. It is not necessary to call the SetFocusUnitProcedure method. The return code is a Boolean value that can be used for error handling.

Although not required, unit procedure names should be unique to allow for easy access and changing unit procedure focus.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.AddUnitProcedure (Name, ProcessInstance)
```

Parameters

Name
Data Type: String
Name of created unit procedure

ProcessInstance
Data Type: String
Process instance in recipe

ChangeInputMaterial()

This method substitutes one material for another and reflects this change in all input phases using that material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values"*.

Syntax

```
ReturnCode = RecipeVar.ChangeInputMaterial (OldMatId, NewInputId)
```

Parameters

OldMatId
Data Type: Integer
Existing input material index entry in array of materials. The list of materials is one-based.

NewInputId
Data Type: String
New material to replace existing material

ChangeOutputMaterial()

This method substitutes one material for another and reflects this change in all output phases using that material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ChangeOutputMaterial ( OldOutputId, NewOutputId)
```

Parameters

OldOutputId
Data Type: Integer
Existing output material index entry in array of materials. The list of materials is one-based.

NewOutputId
Data Type: String
New material to replace existing material

ChangeProcessInstance()

This method renames a process instance. The new instance name is modified throughout the recipe procedure. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ChangeProcessInstance ( ProcessInstance,
NewInstanceName)
```

Parameters

ProcessInstance
Data Type: String
Old process instance name

NewInstanceName
Data Type: String
New process instance name

ChangeTransferInstance()

This method modifies a transfer instance name or the source and destination transfer instances. The InstanceName field is required. The NewInstanceName, NewSrcInstance, and NewDestInstance fields are optional and the existing values are used. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ChangeTransferInstance ( InstanceName, NewInstanceName,
NewSrcInstance, NewDestInstance)
```

Parameters

InstanceName
Data Type: String
Transfer Instance defined in recipe

NewInstanceName
Data Type: String
Optional new name of instance

NewSrcInstance
Data Type: String
Optional new source instance for source class

NewDestInstance
Data Type: String
Optional new destination instance for destination process class

ClearAppendtoInstructions()

This method disables the append unit descriptions to instructions property for the focal phase. There is no return value.

Syntax

```
RecipeVar.ClearAppendtoInstructionsDesc ()
```

ClearCommentRequired()

This method disables the comment required property for the focal phase. There is no return value.

Syntax

```
RecipeVar.ClearCommentRequired ()
```

ClearEntryAck()

Disables the phase entry acknowledge property for the focal phase. There is no return value.

Syntax

```
RecipeVar.ClearEntryAck ()
```

ClearEntryAck()

This method disables the phase exit acknowledge property for the focal phase. There is no return value.

Syntax

```
RecipeVar.ClearExitAck ()
```

ClearHeaderComments()

This method removes the header comments from the recipe. There is no return value.
Syntax

```
RecipeVar.ClearHeaderComments ()
```

ClearInstructions()

This method removes the phase instructions from the focal phase. There is no return value.
Syntax

```
RecipeVar.ClearInstructions ()
```

ClearPhaseContinue()

This method disables the continue mode property for the focal phase. There is no return value.
Syntax

```
RecipeVar.ClearPhaseContinue ()
```

ClearPhaseReportName()

This method removes the assigned report name from the focal phase. There is no return value.
Syntax

```
RecipeVar.ClearPhaseReportName
```

ClearProcessInstanceUnit()

This method removes the unit assignment from a process instance. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.ClearProcessInstanceUnit ( ProcessInstance)
```
Parameters

ProcessInstance
Data Type: String
Instance for which to remove unit assignment

ClearProductionApproval()

This method disables recipe production approval. A recipe cannot be run without either approval for production or test. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.ClearProdutionApproval ()
ClearTestApproval()
```

This method disables recipe test approval. A recipe cannot be run without either approval for production or test. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.ClearTestApproval ()
```

Close()

This method closes the currently open recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.Close ()
```

DeleteInputMaterials()

This method deletes input materials. Phases using a deleted material are reset. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteInputMaterials ( InputMaterials)
```

Parameters

InputMaterials
Data Type: Object
List of input materials to delete

DeleteOperation()

This method deletes an operation from the focal unit procedure. The focal operation becomes the previous operation and the focal phase is the phase header. If no operations exist following the deletion, focus returns to the operation header. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteOperation ()
DeleteOpFromLibrary()
```

This method deletes an operation from the operation library. The operation to delete is selected from the list of possible operations that can be obtained by calling the **GetOpLibrary** method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteOpFromLibrary ( OperationIndex)
```

Parameters

OperationIndex
Data Type: Integer
Index entry in array of available operations. The list of operations is one-based.

DeleteOutputMaterials()

This method deletes output materials. Phases using a deleted material are reset. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteOutputMaterials ( OutputMaterials)
```

Parameters

OutputMaterials
Data Type: Object
Output materials to delete

DeletePhase()

This method deletes the phase with focus, or if the phase header has focus, this method deletes all phases. The phase header has focus following a call to SetFocusOperation. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeletePhase ()
```

DeleteProcessClass()

This method deletes a process class from the recipe. All process instances are also deleted. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteProcessClass ( ProcessClass)
```

Parameters

ProcessClass
Data Type: String
Process Class to delete

DeleteProcessInstance()

This method deletes a process instance. Because process instances are unique across all process classes, specification of a process class is not necessary. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteProcessInstance ( ProcessInstance)
```

Parameters

ProcessInstance
Data Type: String
Instance to delete

DeleteRecipe()

This method deletes a recipe from the recipe database. The recipe remains in memory as an untitled recipe but is removed from the list of available recipes. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteRecipe (RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe to delete

DeleteTransferInstance()

This method deletes a transfer instance. Transfer instances are unique across all transfer classes so specification of a transfer class in not necessary. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteTransferInstance ( TransferInstance)
```

Parameters

TransferInstance
Data Type: String
Transfer Instance to delete

DeleteUnitProcedure()

This method deletes a unit procedure from the recipe procedure. The focal unit procedure becomes the previous unit procedure and the focal operation is the operation header. If only two recipe levels are defined, then the focal phase is the phase header. If no unit procedures exist following the deletion, focus returns to the procedure header. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteUnitProcedure ()
```

DeleteUpFromLibrary()

This method deletes a unit procedure from the unit procedure library. The unit procedure to delete is selected from the list of possible unit procedures that can be obtained by calling the **GetUpLibrary** method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.DeleteOpFromLibrary ( UnitProcedureIndex)
```

Parameters

UnitProcedureIndex
Data Type: Integer
Index entry in array of available unit procedures. The list of unit procedures is one-based.

ExportRecipe()

This method exports a recipe from the recipe database to a binary RCP file. Only the recipes that exist in the recipe database can be exported. All exported recipes are placed in the config_A directory, by default, and have an .rcp extension. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ExportRecipe (RecipeId, Overwrite)
```

Parameters

RecipeId
Data Type: String
Name of recipe to export

Overwrite
Data Type: Boolean
Flag to determine if the exported recipe should overwrite an existing recipe file if one is found in the config_A directory. A value of True overwrites an existing recipe file. A value of False preserves an existing recipe file.

ExportRecipeXML()

This method exports a recipe from the recipe database to an XML file. Only the recipes that exist in the recipe database can be exported. All exported recipes are placed in the config_A directory, by default, and have an .xml extension. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *Error Return Values* .

Syntax

```
ReturnCode = RecipeVar.ExportRecipeXML (RecipeId, Overwrite)
```

Parameters

RecipeId
Data Type: String
Name of recipe to export

Overwrite
Data Type: Boolean
Flag to determine if the exported recipe should overwrite an existing recipe file if one is found in the config_A directory. A value of True overwrites an existing recipe file. A value of False preserves an existing recipe file.

GetAllProcessInstances()

This method queries the recipe for all defined process instances. An array of string values containing the process instances is returned.
Syntax

```
ProcessInstances = RecipeVar.GetAllProcessInstances ()
```

GetAllTransferInstances()

This method queries the recipe for all transfer instances. An array of string values containing the transfer instances is returned.
Syntax

```
TransferInstances = RecipeVar.GetAllTransferInstances ()
```

GetApprovalHistory()

This method queries the approval history for a particular version of the currently open recipe. The version is selected from the list of all versions that can be obtained by calling the **GetRecipeHistory** method. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetApprovalHistory ( VersionIndex, Approval2, Approval3,
Approval4, Approval5, Test, State, Type)
```

Parameters

VersionIndex
Data Type: Integer
Index entry in array of versions. The list of versions is one-based.

Approval2
Data Type: String
User name of individual who was the second to approve the selected version for production

Approval3
Data Type: String
User name of individual who was the third to approve the selected version for production

Approval4
Data Type: String
User name of individual who was the fourth to approve the selected version for production

Approval5
Data Type: String
User name of individual who was the fifth to approve the selected version for production

Test
Data Type: String
User name of individual approved the selected version for test

State
Data Type: String
State of the recipe for the selected version

Type
Data Type: String
Type of the recipe for the selected version

GetConnectionInfo()

This method queries a specified connection and returns the transfer class, source unit, and destination unit.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetConnectionInfo ( ConnectionName, TransferClass,
SourceUnit, DestinationUnit)
```

Parameters

Connection
Data Type: String
Unique name of the connection

TransferClass
Data Type: String
Transfer class of the specified connection

SourceUnit
Data Type: String
Name of the destination unit for the specified connection

GetConnections()

This method queries for all connections. An array of string values containing the connection names is returned.

Syntax

```
Connections = RecipeVar.GetConnections ()
```

GetDocumentInfo()

This method queries the document viewing requirements for the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetDocumentInfo (DocPath, DocDesc, DocViewReq)
```

Parameters

DocPath
Data Type: String
Path to the document

DocDesc
Data Type: String
Description of the assigned document

DpcViewReq
Data Type: Integer
Document viewing acknowledgement setting:
0 = None
1 = On entry
2 = Prior to exit

GetFocusOperation()

This method queries the focal operation name. The operation name is returned as a string.

Syntax

```
OperationName = RecipeVar.GetFocusOperation ()
```

GetFocusPhase()

This method queries the focal phase label. The phase label is returned as a string.
Syntax

```
PhaseLabel = RecipeVar.GetFocusPhase ()
```

GetFocusUnitProcedure()

This method queries the focal unit procedure name. The unit procedure name is returned as a string.
Syntax

```
UnitProcedureName = RecipeVar.GetFocusUnitProcedure ()
```

GetHeaderComments()

This method queries the header comments for the currently open recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetHeaderComments (Comments)
```
Parameters

Comment
Data Type: String
Comments defined in the header of the recipe

GetInputMaterialChars()

This method queries the characteristics defined for a specified input. The characteristics and their values are returned in arrays that can be traversed for information. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetInputMaterialChars ( MaterialId,
CharacteristicNames, CharacteristicValues)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

CharacteristicName
Data Type: Object
Array of names for the characteristics defined for the selected material

CharacteristicValues
Data Type: Object
Array of values for the characteristics defined for the selected material

GetInputMaterialDescription()

This method queries the description defined for a specified input. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetInputMaterialDescription ( MaterialId, Description)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Description
Data Type: String
Input material description

GetInputMaterialInfo()

This method queries information concerning a specified input. The information returned allows identification of duplicate materials. The UnitofMeasure and the ValueType may be defined to uniquely identify input materials. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetInputMaterialInfo (MatId, MaterialId,
MaterialUnitOfMeasure, MaterialValueType)
```

Parameters

MatId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

MaterialId
Data Type: String
Input material ID

MaterialUnitOfMeasure
Data Type: String
Input material unit of measure value

MaterialValueType
Data Type: Integer
Input material value type

GetInputMaterialPhaseLabels()

This method queries the recipe procedure for a list of phases assigned to the specified input material. An array of string values containing the input phase labels is returned.

Syntax

```
InputPhaseLabels = RecipeVar.GetInputMaterialPhaseLabels (InputId)
```

Parameters

InputId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

GetInputMaterials()

This method queries the recipe for a list of input materials ids. An array of string values containing the input materials is returned.

Syntax

```
InputMaterials = RecipeVar.GetInputMaterials ()
GetInputMaterialTotal()
```

This method queries the total amount of input material used in the recipe. The total is returned in the Total parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetInputMaterialTotal ( MaterialId, Total)
```

Parameters

MaterialId
Data Type: Integer

Input material index entry in array of materials. The list of materials is one-based.

Total
Data Type: Double
Total of material in recipe

GetInputMaterialValue()

This method queries the value of an input material. The material value is returned in the Value parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetInputMaterialValue ( MaterialId, Value)
```

Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
Value assigned to material

GetInputParmInfo()

This method queries all of the defined parameter information for an input parameter associated with the focal phase. The information returned includes everything about the input parameter as defined for the phase. This method is typically called following the **GetParmType** method once an input parameter is located. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetInputParmInfo (ParmName, Value, Tolerance, HighDev,
LowDev, MaterialId, ValueType, Total, UnitOfMeasure)
```

Parameters

ParmName
Data Type: String
Input parameter name

Value
Data Type: Double
Input material value

Tolerance
Data Type: Integer
Input tolerance setting:
0 = General
1 = Recipe
2 = None

HiDev
Data Type: Double
Input material high deviation

LowDev
Data Type: Double
Input material low deviation

MaterialId
Data Type: String
Input material ID

ValueType
Data Type: Integer
Input value type setting:
0 = Percent
1 = Actual

Total
Data Type: Boolean
Flag to determine if the material is totaled in the formula. A value of True indicates totaling enabled. A value of False indicates totaling disabled.

UnitofMeasure
Data Type: String
Input material unit of measure

GetInstructions()

This method queries the focal phase for all defined instructions. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetInstructions (Instructions, AppendToInstructions)
```

Parameters

Instructions
Data Type: String
Instructions defined for the phase

AppendToInstructions
Data Type: Boolean
Flag to determine if append unit/connection description is enabled for the phase. A value of True indicates that append unit or connection description is enabled. A value of False indicates that append unit or connection description is disabled.

GetMaterialDBUofM()

This method queries the default unit of measure for a material from the materials database. The unit of measure value is returned as a string.

Syntax

```
MaterialUofM = RecipeVar.GetMaterialDBUofM (MaterialId)
```

Parameters

MaterialId
Data Type: String
Material ID

GetMaterials()

This method queries the materials database and returns a list of ids for the specified type. An array of string values containing the materials is returned.

Syntax

```
ListMaterials = RecipeVar.GetMaterials (MaterialType)
```

Parameters

MaterialType
Data Type: wwMaterialType
Material type enumeration. For MaterialType enumeration constants, see *"Enumerated Constants" on page 383*.

GetOperations()

This method queries the focal unit procedure for all operations. An array of string values containing the operations is returned.

Syntax

```
Operations = RecipeVar.GetOperations ()
```

GetOpLibrary()

This method queries the recipe database for all available operations in the operation library. Two arrays are returned. One array contains the names of the operations. The second array contains that operation comments. These arrays can be used to load and delete operations from the library. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetOpLibrary (OpName, OpComment)
```

Parameters

OpName
Data Type: Object
Array of names for all of the operations in the operation library

OpComment
Data Type: Object
Array of comments for all of the operations in the operation library

GetOutputMaterialChars()

This method queries the characteristics defined for a specified output. The characteristics and their values are returned in arrays that can be traversed for information. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetOutputMaterialChars ( MaterialId,
CharacteristicNames, CharacteristicValues)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

CharacteristicName
Data Type: Object
Array of names for the characteristics defined for the selected material.

CharacteristicValues
Data Type: Object
Array of values for the characteristics defined for the selected material

GetOutputMaterialDescription()

This method queries the description defined for a specified output. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetOutputMaterialDescription ( MaterialId, Description)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

Description
Data Type: String
Output material description

GetOutputMaterialInfo()

This method queries information concerning a specified output. The information returned allows identification of duplicate materials. The **UnitofMeasur**e or the **ValueType** may be defined to uniquely identify output materials. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetOutputMaterialInfo (MatId, MaterialId,
MaterialUnitOfMeasure, MaterialValueType)
```

Parameters

MatId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

MaterialId
Data Type: String
Output material ID

MaterialUnitOfMeasure
Data Type: String
Output material unit of measure value

MaterialValueType
Data Type: Integer
Output material value type

GetOutputMaterialPhaseLabels()

This method queries the recipe procedure for a list of phases assigned to an output material. An array of string values containing the output phase labels is returned.
Syntax

```
OutputPhaseLabels = RecipeVar.GetOutputMaterialPhaseLabels (OutputId)
```

Parameters

OutputId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

GetOutputMaterials()

This method queries the recipe for a list of output materials ids. An array of string values containing the output materials is returned.
Syntax

```
OutputMaterials = RecipeVar.GetOutputMaterials ()
```

GetOutputMaterialTotal()

This method queries the total amount of output material used in the recipe. The total is returned in the Total parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetOutputMaterialTotal ( MaterialId, Total)
```

Parameters

MaterialId
Data Type: Integer

Output material index entry in array of materials. The list of materials is one-based.

Total
Data Type: Double
Total material in recipe

GetOutputMaterialValue()

This method queries the value of an output material. The material value is returned in the Value parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetOutputMaterialValue ( MaterialId, Value)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
Assigned value to material

GetOutputParmInfo()

This method queries all of the defined parameter information for an output parameter associated with the focal phase. The information returned includes everything about the output parameter as defined for the phase. This method is typically called following the **GetParmType** method once an output parameter is located. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetOutputParmInfo (ParmName, Value, MaterialId,
ValueType, Total, UnitOfMeasure)
```

Parameters

ParmName
Data Type: String
Output parameter name

Value
Data Type: Double
Output material value

MaterialId
Data Type: String
Output material ID

ValueType
Data Type: Integer
Input value type setting:
0 = Percentl
1 = Actual

Total
Data Type: Boolean
Flag to determine if the material is totaled in the formula. A value of True indicates that totaling is enabled. A value of False indicates that totaling is disabled.

UnitofMeasure
Data Type: String
Output material unit of measure

GetParameterDescription()

This method queries the description defined in the Process Model database of a specific parameter in the focal phase. The name of the phase parameter is required as an input to the function. The parameter description is returned as a parameter of the function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetParameterDescription ( ParmName, Description)
```

Parameters

ParmName
Data Type: String
Name of the parameter for which the description is desired

Description
Data Type: String
Description of the parameter

GetParmElementInfo()

This method queries all of the configuration information defined in the Process Model database for each element of the selected parameter for the focal process or transfer phase. The name of the phase parameter is required as an input to the function. The parameter type, data class, and three arrays of parameter element information are returned as parameters of the function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

The **ElementTypes** array contain the appropriate enumeration value (wwParamElementType) for the parameter element names (such as Target and Actual). The **ElementViews** array contains the appropriate enumeration value, (wwPhaseParamViewType) for the Enable Display, Edit Allowed, or Edit Required setting for each parameter element. The **ElementValues** array contains the default values for each parameter element where applicable. The three arrays contain information based on the type and data class of the selected parameter and all information is in the same relative position in each array.

For example, if an analog process variable parameter is selected, the first position in the **ElementTypes** array is 0 for the Target element, the second for the **Actual** element, the third position in the **ElementTypes** array is 2 for the **High Deviation** element, and so forth for all six possible elements of an analog process variable parameter (see the table that follows). However, for an analog input parameter, there would be seven entries in the **ElementTypes** array that correspond to the seven elements available for an input parameter (see the tables that follow).

Further, the values in the **ElementViews** and **ElementValues** arrays correspond to the appropriate element type in the **ElementTypes** array. Thus, for an analog process variable parameter, the first position in the **ElementTypes** array is 0 for the **Target** element, the first position in the ElementViews array is 1, 3, or 7 for the **Enable Display, Edit Allowed, or Edit Required** setting for the Target element, and the first position in the **ElementValues** array is the default value for the **Target** element. However, for an analog input parameter, the first position in the **ElementTypes** array is 0 for the **Target** element, the first position in the ElementViews array is 1, 3, or 7 for the Enable Display, Edit Allowed, or Edit Required setting for the **Target** element, but the first position in the **ElementValues** array is empty because there is no default value for the **Target** element of an input parameter.

The following tables show the basic structure of the three arrays for each possible parameter type and data class combination.

Process Variable Parameters (Analog)

The following table shows the basic structure of the analog process variable parameters.

Element Name	Types Array Entry	Views Array Entry	Values Array Entry
Target	0	0, 1, 3, or 7	Default value
Actual	1	0, 1, 3, or 7	Always blank
High Deviation	2	0, 1, 3, or 7	Default value
Low Deviation	3	0, 1, 3, or 7	Default value
High Limit	4	0, 1, 3, or 7	Default value
Low Limit	5	0, 1, 3, or 7	Default value

Process Variable Parameters (Discrete or String)

The following table shows the basic structure of the discrete or string process variable parameters.

Element Name	Types Array Entry	Views Array Entry	Values Array Entry
Target	0	0, 1, 3, or 7	Default value
Actual	1	0, 1, 3, or 7	Always blank

Input Parameters (Analog Only)

The following table shows the basic structure of the input parameters.

Element Name	Types Array Entry	Views Array Entry	Values Array Entry
Target	0	0, 1, 3, or 7	No value returned
Actual	1	0, 1, 3, or 7	No value returned
High Deviation	2	0, 1, 3, or 7	No value returned
Low Deviation	3	0, 1, 3, or 7	No value returned
Preact	4	0, 1, 3, or 7	No value returned
Lot Code	5	0, 1, 3, or 7	No value returned
Material ID	6	0, 1, 3, or 7	No value returned

Output Parameters (Analog Only)

The following table shows the basic structure of the output parameters,

Element Name	Types Array Entry	Views Array Entry	Values Array Entry
Target	0	0, 1, 3, or 7	No value returned
Actual	1	0, 1, 3, or 7	No value returned
Material ID	2	0, 1, 3, or 7	No value returned

Syntax

```
ReturnCode = RecipeVar.GetParmElementInfo (ParmName, ParmType, DataClass,
ElementTypes, ElementViews, ElementValues)
```

Parameters

ParmName
Data Type: String
Name of the parameter for which the element information is desired

ParmType
Data Type: Integer
Phase parameter type setting:
0 = Input
1 = Process variable
2 = Output

DataClass
Data Type: Integer
Phase parameter data class setting:
0 = Analog
1 = Discrete
2 = String
3 = Enumeration

ElementTypes
Data Type: Object
Array of elements available for the selected parameter. Each entry is an integer that corresponds to the wwParamElementType enumeration. For Element Type enumeration constants, see *"wwParamElementType" on page 384*.

ElementViews
Data Type: Object
Array of display and edit settings for the appropriate parameter elements. Each entry is an integer that corresponds to the wwPhaseParamViewType enumeration. For Element View enumeration constants, see *"wwPhaseParamViewType" on page 386*.

ElementValues
Data Type: Object
Array of default values for the appropriate parameter elements. Each entry is a string that corresponds to the default value defined in the Process Model Editor for that corresponding element.

GetParmEnumValues()

This method queries the enumeration values available for a selected parameter. An array is returned that contains the enumeration values. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetParmEnumValues (ParmName, EnumValues)
```

Parameters

ParmName
Data Type: String
Name of the parameter

EnumValue
Data Type: Object
Array of enumeration values available for the selected parameter

GetParmType()

This method queries the selected parameter for its type. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetParmType (ParmName, ParmType)
```

Parameters

ParmName
Data Type: String
Name of the parameter

ParmType
Data Type: Integer
Phase parameter type setting:
0 = Input
1 = Process variable
2 = Output

GetPhaseDescription()

This method queries the description defined in the Process Model database for the focal phase. The phase description is returned as a parameter of the function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetPhaseDescription ( Description)
```

Parameters

Description
Data Type: String
Description of the focal phase

GetPhaseInfo()

This method queries the focal phase for all defined property information. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetPhaseInfo (ReportName, EntryAck, ExitAck,
CommentRequired, ContinueMode)
```

Parameters

ReportName
Data Type: String
Name of report assigned to the phase

EntryAck
Data Type: Integer
Phase entry acknowledgement setting:
0 = Entry acknowledge
3 = Done by Entry acknowledge
7 = Check by Entry acknowledge

ExitAck
Data Type: Integer
Phase entry acknowledgement setting:
8 = Exit acknowledge
24 = Done by Exit acknowledge
56 = Check by Exit acknowledge

CommentRequired
Data Type: Boolean
Flag to determine if comment required property is enabled for the phase. A value of True indicates comment required enabled. A value of False indicates comment required disabled.

ContinueMode
Data Type: Boolean
Flag to determine if continue mode property is enabled for the phase. A value of True indicates continue mode enabled. A value of False indicates continue mode disabled.

GetPhaseLabels()

This method queries the focal operation and returns a list of the operation's phase labels. An array of string values containing the phase labels is returned.
Syntax

```
PhaseLabels = RecipeVar.GetPhaseLabels ()
```

GetPhaseParams()

This method queries the focal phase and returns a list of all phase parameters. An array of string values containing the phase parameters is returned.
Syntax

```
PhaseParams = RecipeVar.GetPhaseParams ()
```

GetPhases()

This method queries the focal operation and returns a list of phase names. An array of string values containing the phases is returned.
Syntax

```
Phases = RecipeVar.GetPhases ()
```

GetPhaseType()

This method queries the type for the focal phase. A Short return value provides the phase type. The value can be interpreted as follows:

0 = Process
1 = Allocate process
2 = Release process
3 = Transfer
4 = Allocate transfer
5 = Release transfer
Syntax

```
PhaseType = RecipeVar.GetPhaseType ()
```

GetProcessClasses()

This method queries the Process Model database for all process classes. An array of string values containing the process classes is returned.

Syntax

```
ProcessClasses = RecipeVar.GetProcessClasses ()
```

GetProcessClassUnits()

This method queries the Process Model database and returns a list of units assigned to a process class. An array of string values containing the assigned units is returned.

Syntax

```
AssignedUnit = RecipeVar.GetProcessClassUnits ( ProcessClass)
```

Parameters

ProcessClass
Data Type: String
Process class in model database

GetProcessInstanceAttributes()

This method queries the attributes defined for a specified process instance in the recipe. Three arrays are returned that includes the names, minimum values, and maximum values for each attribute. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetProcessInstanceAttributes ( ProcessInstance,
AttributeNames, AttributeMins, AttributeMaxs)
```

Parameters

ProcessInstance
Data Type: String
Name of process instance for which attributes are desired

Attribute
Data Type: Object
Array of names for the attributes defined for the selected process class

AttributeMins
Data Type: Object
Array of minimum values for the attributes defined for the selected process class

AttributeMaxs
Data Type: Object
Array of maximum values for the attributes defined for the selected process class

GetProcessInstances()

This method queries the process class instances defined for a recipe. An array of string values containing the process instances is returned.

Syntax

```
ProcessInstances = RecipeVar.GetProcessInstances ( ProcessClass)
```

Parameters

ProcessClass
Data Type: String
Class to query for its instances

GetProcessInstanceUnit()

This method queries a process instance for its unit assignment. The unit name is returned as a string.

Syntax

```
UnitName = RecipeVar.GetProcessInstanceUnit ( ProcessInstance)
```

Parameters

ProcessInstance
Data Type: String
Instance to query for unit assignment

GetProcessPhaseDescByClass()

This method queries the description defined in the Process Model database for a specific process phase. The names of the process class and process phase are required as input to the function. The phase description is returned as a parameter of the function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetProcessPhaseDescByClass ( ProcessClass,
ProcessPhase, Description)
```

Parameters

ProcessClass
Data Type: String
Name of the process class as defined in the Process Mode

ProcessPhase
Data Type: String
Name of the process phase as defined in the Process Model

Description
Data Type: String
Description of the process phase requested

GetProcessPhaseDescByInstance()

This method queries the description defined in the Process Model database for a specific process phase based on the phase instance in a recipe. The names of the recipe process instance and process phase are required as input to the function. The phase description is returned as a parameter of the function. This method differs from **GetProcessPhaseDescByClass** in that this method requires the process instances be added to a recipe prior to the retrieval of the description. The **GetProcessPhaseDescByClass** method does not require any prior recipe configuration. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetProcessPhaseDescByInstance ( ProcessInstance,
ProcessPhaseName, PhaseDescription)
```

Parameters

ProcessInstance
Data Type: String
Name of the process instance as defined in the recipe

ProcessPhase
Data Type: String
Name of the process phase as defined in the Process Model

Description
Data Type: String
Description of the process phase requested

GetProcessPhases()

This method queries the phases from the Process Model database for a specific process class. The name of the process class is required as an input to the function. An array of process phases is returned.

Syntax

```
ProcessPhases = RecipeVar.GetProcessPhases ( ProcessClass)
```

Parameters

ProcessClass
Data Type: String
Name of the process class as defined in the Process Mode

GetProcessVariables()

This method queries all recipe phase labels, phases, and phase formula parameters. Three arrays are returned by this function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetProcessVariables (Labels, Phases, Parameters)
```

Parameters

Labels
Data Type: Object
Array of all labels for phases in the recipe

Phases
Data Type: Object
Array of all phases in the recipe

Parameters
Data Type: Object
Array of all phase formula parameters in the recipe

GetProcessVarParmInfo()

This method queries all the defined parameter information for a process variable parameter associated with the focal phase. The information returned includes everything about the process variable parameter as defined for the phase. This method is typically called following the **GetParmType** method after a process variable parameter is located. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetProcessVarParmInfo ( ParmName, Value, ToleranceType,
HighDev, LowDev, UnitofMeasure)
```

Parameters

Labels
Data Type: String
Process variable parameter name

Value
Data Type: String
Process variable value

ToleranceType
Data Type: Integer
Process variable tolerance setting:
0 = General
1 = Recipe
2 = None

HighDev
Data Type: Double
Process variable high deviation

LowDev
Data Type: Double

Process variable low deviation

UnitOfMeasure
Data Type: String
Process variable unit of measure

GetRecipeApprovals()

This method queries the recipe approval status for the currently open recipe. The approval flags are returned when this method is called. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetRecipeApprovals (Production, Test)
```

Parameters

Production
Data Type: Boolean
Flag to determine if recipe is approved for production. A value of True indicates that approved for production is enabled. A value of False indicates that approved for production is disabled.

Test
Data Type: Boolean
Flag to determine if recipe is approved for test. A value of True indicates that approved for test is enabled. A value of False indicates that approved for test is disabled.

GetRecipeHistory()

This method queries the history for the currently open recipe. The number of versions is returned as a parameter. In addition, three arrays are returned that includes the dates, authors, and comments for each version. The arrays can be traversed using the number of versions information. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetRecipeHistory (NumVersions, Dates, Authors, Comments)
```

Parameters

NumVersion
Data Type: Integer
Number of recipe versions available

Dates
Data Type: Object
Array of dates for each recipe version

Authors
Data Type: Object
Array of authors for each recipe version

Comments
Data Type: Object
Array of comments for each recipe version

GetRecipeProcessClasses()

This method queries all process classes assigned to a recipe. An array of string values containing the process classes is returned.

Syntax

```
ProcessClasses = RecipeVar.GetRecipeProcessClasses ()
```

GetState()

This method queries the state defined for the current open recipe. The name of the state assigned to the recipe is returned as a parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetState (StateName)
```

Parameters

StateName
Data Type: String
Name of state assigned to recipe

GetStateInfo()

This method queries the information for a specific state in the recipe database. The name of the state is required and the description, default status, schedule status, and read only information is returned in the parameters. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetStateInfo (StateName, Description, DefaultState,
ScheduleState, ReadOnly)
```

Parameters

StateName
Data Type: String
Name of state for which information is desired

Description
Data Type: String
Description of specified state

DefaultState
Data Type: Boolean
Flag to determine if state is the default for the recipe database. A value of True indicates the default state. A value of False indicates not the default state.

ScheduleState
Data Type: Boolean
Flag to determine if state allows scheduling. A value of True indicates that recipes with this state can be scheduled. A value of False indicates that recipes with this state cannot be scheduled.

ReadOnly
Data Type: Boolean
Flag to determine if recipe with this state is read only or read/write. A value of True indicates read only. A value of False indicates read and write.

GetStates()

This method queries all states defined in the recipe database. The states are returned in an object.

Syntax

```
RecipeStates = RecipeVar.GetStates ()
```

GetTrainAttributes()

This method queries the train for the defined attributes, values, and data classes. An array of string values containing the attribute names is returned in the names argument. An array of values is returned in the values argument. An array of ValueTypes is returned in the ValueTypes argument. The function returns a boolean value: True if successful, and False if there is an error.

Syntax

```
Dim Names as Object
Dim Values as Object
```

```
Dim ValueTypes as Object
Dim Train as String
ReturnCode = RecipeVar.GetTrainAttributes (Train, Names, Values, ValueTypes)
```

Parameters

Train
Data Type: String
Train to query for its attributes

Names
Data Type: Object
Returned names of the attributes

Values
Data Type: Object Array
Returned Values of attributes

ValueTypes
Data Type: Object Array
Returned Value Types of attributes:
0 = Real
1 = Integer
2 = String

GetTrains()

This method queries the system for the defined trains. An array of string values containing the train names is returned.

Syntax

```
Trains = RecipeVar.GetTrains ()
```

GetTrainUnits()

This method queries the train for its contained units. An array of string values containing the unit names is returned.

Syntax

```
Units = RecipeVar.GetTrainUnits (Train)
```

Parameters

Train
Data Type: String
Train to query for its units

GetTransferClasses()

This method queries the recipe for the defined transfer classes. Transfer classes are added to the recipe when both the source and destination process classes are defined in the recipe. An array of string values containing the transfer classes is returned.

Syntax

```
TransferClasses = RecipeVar.GetTransferClasses ()
```

GetTransferInstances()

This method queries the transfer class instances defined for a recipe. A transfer class and default instances are added to a recipe automatically when both the source and destination process classes are defined for the recipe. An array of string values containing the transfer instances is returned.

Syntax

```
TransferInstances = RecipeVar.GetTransferInstances ( TransferClass)
```

Parameters

TransferClass
Data Type: String

Class to query for its instances

GetTransferInstanceSrcDest()

This method queries a transfer instance for its source and destination instances. These source and destination instances are defined under the source and destination process classes for the transfer class. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetTransferInstanceSrcDest ( TransferInstance, Source, Destination)
```

Parameters

TransferInstance
Data Type: String
Transfer Instance to query for source and destination instances

Source
Data Type: String
Source instance of Transfer

Destination
Data Type: String
Destination instance of Transfer

GetTransferPhaseDescByClass()

This method queries the description defined in the Process Model database for a specific transfer phase. The names of the transfer class and transfer phase are required as input to the function. The phase description is returned as a parameter of the function. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetTransferPhaseDescByClass ( TransferClass, TransferPhase, Description)
```

Parameters

TransferClass
Data Type: String
Name of the transfer class as defined in the Process Model

TransferPhase
Data Type: String
Name of the transfer phase as defined in the Process Model

Description
Data Type: String
Description of the transfer phase requested

GetTransferPhaseDescByInstance()

This method queries the description defined in the Process Model database for a specific transfer phase based on the phase instance in a recipe. The names of the recipe transfer instance and transfer phase are required as input to the function. The phase description is returned as a parameter of the function. This method differs from **GetTransferPhaseDescByClass** in that this method requires the transfer instances be added to a recipe prior to the retrieval of the description. The **GetTransferPhaseDescByClass** method does not require any prior recipe configuration. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetTransferPhaseDescByInstance ( TransferInstance,
TransferPhaseName, PhaseDescription)
```

Parameters

TransferInstance
Data Type: String
Name of the transfer instance as defined in the Process Model

TransferPhase
Data Type: String
Name of the transfer phase as defined in the Process Model

Description
Data Type: String
Description of the transfer phase requested

GetTransferPhases()

This method queries the phases from the Process Model database for a specific transfer class. The name of the transfer class is required as an input to the function. An array of transfer phases is returned.

Syntax

```
TransferPhases = RecipeVar.GetTransferPhases ( TransferClass)
```

Parameters

TransferClass
Data Type: String
Name of the transfer class as defined in the Process Model

GetType()

This method queries the type defined for the current open recipe. The name of the type assigned to the recipe is returned as a parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetType (TypeName)
```

where:

Parameters

TypeName
Data Type: String
Name of type assigned to recipe

GetTypeInfo()

This method queries the information for a specific type in the recipe database. The name of the type is required and the description and default status information is returned in the parameters. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.GetTypeInfo (TypeName, Description, DefaultType)
```

Parameters

TypeName
Data Type: String
Name of type for which information is desired

Description
Data Type: String
Description of specified type

DefaultType
Data Type: Boolean
Flag to determine if type is the default for the recipe database. A value of True indicates the default type. A value of False indicates not the default type.

GetTypes()

This method queries all types defined in the recipe database. The types are returned in an object.
Syntax

```
RecipeTypes = RecipeVar.GetTypes ()
```

GetUnitProcedureProcessPhases()

This method queries the focal unit procedure and returns a list of process phases. This list represents all process phases for the unit procedure's process instance. An array of string values containing the process phases is returned.
Syntax

```
ProcessPhases = RecipeVar.GetUnitProcedureProcessPhases ()
```

GetUnitProcedures()

This method queries the recipe procedure for all unit procedures. An array of string values containing the unit procedures is returned.
Syntax

```
UnitProcedures = RecipeVar.GetUnitProcedures ()
```

GetUnitProcedureTransferPhases()

This method queries the Process Model database for transfer instances and phases associated with the focal unit procedure. These lists represent the instances and phases for the unit procedure's process instance. The transfer instances and phase are returned to the TransferInstances and TransferPhases parameters. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetUnitProcedureTransferPhases ( TransferInstances,
TransferPhases)
```

Parameters

TransferInstances
Data Type: Object
List of transfer instances

TransferPhases
Data Type: Object
List of transfer phases

GetUpLibrary()

This method queries the recipe database for all available unit procedures in the unit procedure library. Two arrays are returned. One array contains the names of the unit procedures. The second array contains that unit procedure comments. These arrays can then be used to load and delete unit procedures from the library. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.GetUpLibrary (UpNames, UpComments)
```

Parameters

UpName
Data Type: Object
Array of names for all of the unit procedures in the unit procedure library

UpComments
Data Type: Object
Array of comments for all of the unit procedures in the unit procedure library

ImportRecipe()

This method imports a recipe from a file into the recipe database. Recipes that are exported from the recipe database and have an .rcp extension can be imported using this method. Also, recipes that have an .xml extension and are in the proper format can be imported. The return code is a Boolean value that can be used for error handling. You should also check the SMC Logger or LogViewer for more details if you get an error.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ImportRecipe (FileName, Overwrite)
```

Parameters

FileName
Data Type: String
Name of recipe file to import. This name can include the full path to the file as well.

UpComments
Data Type: Boolean
Flag to determine if the imported recipe should overwrite an existing recipe if one is found in the recipe database. A value of True overwrites an existing recipe. A value of False preserves an existing recipe.

LoadOpFromLibrary()

This method loads an operation from the operation library into the focal unit procedure. The operation is placed after the current focal operation. The operation to load is selected from the list of possible operations that can be obtained by calling the **GetOpLibrary** method. If the proper equipment has not been assigned to the recipe, the operation does not load. The InsertMaterials parameter can optionally be used to automatically add any materials defined in the operation to the recipe formula. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.LoadOpFromLibrary ( OperationIndex, InsertMaterials)
```

Parameters

OperationIndex
Data Type: Integer
Index entry in array of available operations. The list of operations is one-based.

UpComments
Data Type: Boolean
Flag to determine if the materials associated with the operation to be loaded should be automatically added to the recipe formula. A value of True adds the materials. A value of False does not add the materials. The operation is added even if the materials are not added to the formula.

LoadUpFromLibrary()

This method loads a unit procedure from the unit procedure library into the recipe procedure. The unit procedure is placed after the current focal unit procedure. The unit procedure to load is selected from the list of possible unit procedures that can be obtained by calling the **GetUpLibrary** method. If the proper equipment has not been assigned to the recipe, the unit procedure cannot load. The InsertMaterials parameter can optionally be used to automatically add any materials defined in the unit procedure to the recipe formula. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.LoadUpFromLibrary ( UnitProcedureIndex,
InsertMaterials)
```

Parameters

UnitProcedureIndex
Data Type: Integer
Index entry in array of available unit procedures. The list of unit procedures is one-based.

InsertMaterials
Data Type: Boolean
Flag to determine if the materials associated with the unit procedure to be loaded should be automatically added to the recipe formula. A value of True adds the materials. A value of False does not add the materials. The unit procedure is added even if the materials are not added to the formula.

NewRecipe()

This method initializes and clears the recipe in memory. The Construction of recipes requires either calling **NewRecipe** or **Open**. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.NewRecipe ()
```

Open()

This method opens a recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.Open (RecipeId)
```

Parameters

RecipeId
Data Type: String
Recipe in database to open

ResetPhaseFocus()

This method focuses on a phase not within the current focused operation and having that phase become the focal phase. If you are using **SetFocusPhase**, the phase label specified must part of the current focal operation. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.RecipeVar.ResetPhaseFocus ( Label)
```

Parameters

Label
Data Type: String
Phase label

Save()

This method saves the currently opened recipe to the recipe database. The Author field is required. Version comments are optional. The version information is automatically incremented by the server. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.RecipeVar.Save (Author, VersionComments)
```

Parameters

Author
Data Type: String

Modifier of recipe

VersionComments
Data Type: String
Optional comments recorded with recipe version record.

SaveAs()

This method saves a recipe to the recipe database using a unique Recipe Id. The Author and Id fields are required. Version comments are optional. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.RecipeVar.SaveAs (RecipeId, Author, VersionComments)
```

Parameters

RecipeId
Data Type: String
Unique recipe identifier

Author
Data Type: String
Creator of recipe

VersionComments
Data Type: String
Optional comments recorded with recipe version record

SetAppendtoInstructions()

This method enables the appending unit descriptions to phase instructions property for the focal phase. There is no return value.

Syntax

```
RecipeVar.RecipeVar.SetAppendtoInstructions ()
```

SetCommentRequired()

This method enables the comment required property for the focal phase. There is no return value.

Syntax

```
RecipeVar.RecipeVar.SetCommentRequired ()
```

SetDocumentInfo()

This method assigns a document and the viewing requirements to the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.RecipeVar.SetDocumentInfo ( DocPath, DocDesc,
DocViewReq)
```

Parameters

DocPath
Data Type: String
Path to the document

DocDesc
Data Type: String
Description of assigned document

DocViewReq
Data Type: Integer
Document viewing acknowledgement setting:

0 = None
1 = On entry
2 = Prior to exit

SetEntryAck()

This method enables the phase entry acknowledge property for the focal phase. There is no return value.
Syntax

```
RecipeVar.SetEntryAck (EntryAck)
```
Parameters

EntryAck
Data Type: wwPhaseEntry
Phase entry acknowledgement enumeration. For EntryAck enumeration constants, see *"Enumerated Constants" on page 383*.

SetExitAck()

This method enables the phase exit acknowledge property for the focal phase. There is no return value.
Syntax

```
RecipeVar.SetExitAck (ExitAck)
```
Parameters

ExitAck
Data Type: wwPhaseExit
Phase exit acknowledgement enumeration. For ExitAck enumeration constants, see *"Enumerated Constants" on page 383*.

SetFocusOperation()

This method sets focus to a specified operation. Invalid operation names do not reset either operation or phase focus. Since operation names need not be unique, it always sets focus to the first operation found with the name. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetFocusOperation ( OperationName)
```
Parameters

OperationName
Data Type: String
Operation Name in unit procedure

SetFocusPhase()

This method sets focus to a new phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetFocusPhase (PhaseLabel)
```
Parameters

PhaseLabel
Data Type: String
Unique label for selected phase

SetFocusUnitProcedure()

This method sets focus to a specified unit procedure. Invalid unit procedure names do not reset either unit procedure, operation or phase focus. Since unit procedure names need not be unique, it always sets focus to the first unit procedure found with the name. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetFocusUnitProcedure ( UnitProcedureName)
```

Parameters

UnitProcedureName
Data Type: String
Unit Procedure Name in recipe procedure

SetFocusUpPhase()

This method sets the focus on a phase within a unit procedure. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetFocusUpPhase (UnitProcedure, PhaseLabel)
```

Parameters

UnitProcedure
Data Type: String
Name of focal unit procedure

PhaseLabel
Data Type: String
Phase label within unit procedure

SetHeaderComments()

This method defines header comments. The function does not append comments to the currently defined header comments so the comments passed to the function become the new header comments. There is no return value.

Syntax

```
RecipeVar.SetHeaderComments (HeaderComments)
```

Parameters

HeaderComments
Data Type: String
Null terminated string containing all comments

SetInputHighDev()

This method assigns a new high deviation value to an input material. The material must have Recipe tolerances (wwRecipeTolerance). The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputHighDev (MaterialId, HighDev)
```

Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

HighDev
Data Type: Double
New high deviation

SetInputLowDev()

This method assigns a new low deviation value to an input material. The material must have Recipe tolerances (wwRecipeTolerance). The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputLowDev (MaterialId, LowDev)
```

Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

LowDev
Data Type: Double
New low deviation

SetInputParmMaterial()

This method assigns a material and value to an input phase parameter. The material Id must be defined as an input material for the recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputParmMaterial ( Parameter, MaterialId, Value)
```

Parameters

Parameter
Data Type: String
Parameter of focal phase

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
Phase parameter value

SetInputParmValue()

This method assigns a value to the input phase parameter for the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputParmValue (ParmName, Value)
```

Parameters

ParmName
Data Type: String
Input phase parameter name

Value
Data Type: Double
New value for parameter

SetInputTolerance()

This method assigns a tolerance type to an input material. The default type is General (wwGeneralTolerance). The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputTolerance (MaterialId, Tolerance)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Tolerance
Data Type: wwToleranceType
Input tolerance enumeration. For Tolerance enumeration constants, see *"Enumerated Constants" on page 383*.

SetInputTotal()

This method enables or disables the totaling of an input material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetInputTotal (MaterialId, Total)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Total
Data Type: Integer
Flag to disable or enable totaling of material:
0 = Not totaled
1 = Totaled

SetInputUnitOfMeasure()

This method modifies the unit of measure for an input material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetInputUnitOfMeasure ( MaterialId, NewMaterialUofM)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

NewMaterialUofM
Data Type: String
New material unit of measure value

SetInputValue()

This method assigns a new value to an input material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetInputValue (MaterialId, Value)
```
Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
New value for material

SetInputValueType()

This method assigns a new value type to an input material. The default value type is Percent (wwPercentValueType). The return code is a Boolean value that can be used for error handling.

Note: The ValueType parameter is used to distinguish between multiple instances of the same material. It is possible to have more than two instances of a material.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetInputValueType (MaterialId, ValueType)
```

Parameters

MaterialId
Data Type: Integer
Input material index entry in array of materials. The list of materials is one-based.

ValueType
Data Type: wwParmValueType
Input value type enumeration. For ValueType enumeration constants, see *"Enumerated Constants" on page 383*.

SetInstructions()

This method assigns phase instructions to the focal phase. There is no return value.

Syntax

```
RecipeVar.SetInstructions (Instructions)
```

Parameters

Instructions
Data Type: String
String of instructions to be assigned to a phase

SetOutputParmMaterial()

This method assigns a material and value to an output phase parameter. The material Id must be defined as an output material for the recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetOutputParmMaterial ( ParmName, MaterialId, Value)
```

Parameters

ParmName
Data Type: String
Parameter Name in focal phase

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
Phase parameter value

SetOutputParmValue()

This method assigns a value to the output phase parameter for the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetOutputParmValue (ParmName, Value)
```

Parameters

ParmName
Data Type: String
Parameter Name in focal phase

Value
Data Type: Double
Phase parameter value

SetOutputTotal()

This method enables or disables the totaling of an output material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetOutputTotal (MaterialId, Total)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

Total
Data Type: Integer
Flag to disable or enable totaling of material:
0 = Not totaled
1 = Totaled

SetOutputUnitOfMeasure()

This method modifies the unit of measure for an output material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetOutputUnitOfMeasure ( MaterialId, NewMaterialUofM)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

NewMaterialUofM
Data Type: String
New material unit of measure value

SetOutputValue()

This method assigns a new value to an output material. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetOutputValue (MaterialId, Value)
```

Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

Value
Data Type: Double
New value for material

SetOutputValueType()

This method assigns a new value type to an input material. The default value type is Percent (wwPercentValueType). The return code is a Boolean value that can be used for error handling.

Note: The ValueType parameter may be used to distinguish between multiple instances of the same material. It is possible to have more than two instances of a material.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetOutputValueType (MaterialId, ValueType)
```
Parameters

MaterialId
Data Type: Integer
Output material index entry in array of materials. The list of materials is one-based.

ValueType
Data Type: wwParmValueType
Output value type enumeration. For ValueType enumeration constants, see *"Enumerated Constants" on page 383*.

SetPhaseContinue()

This method enables the continue mode property for the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetPhaseContinue ()
```

SetPhaseLabel()

This method changes the label for the focal phase. The function assures uniqueness of phase label names. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetPhaseLabel (OldLabel, NewLabel)
```
Parameters

OldLabel
Data Type: String
Existing phase label

NewLabel
Data Type: String
New unique phase label

SetPhaseReportName()

This method assigns a report name to the focal phase. There is no return value.
Syntax

```
RecipeVar.SetPhaseReportName (ReportName)
```
Parameters

ReportName
Data Type: String
Report name defined in report database

SetProcessInstanceAttribute()

This method assigns a value to a process instance attribute. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetProcessInstanceAttribute ( ProcessInstance,
AttributeName, AttributeMin, AttributeMax)
```

Parameters

ProcessInstance
Data Type: String
Instance containing attribute

AttributeName
Data Type: String
Attribute name from model database

AttributeMin
Data Type: Double
New minimum attribute value

AttributeMax
Data Type: Double
New maximum attribute value

SetProcessInstanceSelMode()

This method defines the unit selection mode for a process instance. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetProcessInstanceSelMode ( ProcessInstance,
SelectionMode)
```

Parameters

ProcessInstance
Data Type: String
Process Instance to assign selection mode

SelectionMode
Data Type: wwUnitSelectionType
Instance selection mode enumeration. For SelectionMode enumeration constants, see *"Enumerated Constants" on page 383*.

SetProcessInstanceUnit()

This method assigns a unit to a process instance. The unit must be assigned to the process instance's process class. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetProcessInstanceUnit ( ProcessInstance, Unit)
```

Parameters

ProcessInstance
Data Type: String
Process Instance to receive unit assignment

Unit
Data Type: String
Unit name from process model database

SetProcessVarParmValue()

This method assigns a value to a process variable parameter for the focal phase. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetProcessVarParmValue ( ParmName, Value)
```

Parameters

ParmName
Data Type: String
Process variable name

Value
Data Type: String
New parameter value

SetProcessVarTolerance()

This method assigns a tolerance type to a process variable parameter. The default type is General (wwGeneralTolerance). The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetProcessVarTolerance ( Parameter, Tolerance, LowDev,
HighDev)
```

Parameters

Parameter
Data Type: String
Process variable parameter

Tolerance
Data Type: wwToleranceType
Process variable tolerance enumeration. For Tolerance enumeration constants, see *"Enumerated Constants" on page 383*.

LowDev
Data Type: Double
New low deviation

HighDev
Data Type: Double
New high deviation

SetProductionApproval()

This method enables recipe production approval. A recipe cannot be run without either approval for production or test. The return code is a Boolean value that can be used for error handling.

Note: This function should only be called after the recipe has been saved using the Save or SaveAs methods. Any change made to a recipe automatically disables the approved for production flag.

For more information on the error codes returned, see *"Error Return Values" on page 388*.
Syntax

```
ReturnCode = RecipeVar.SetProductionApproval ()
```

SetState()

This method assigns a valid state to a recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetState (StateName)
```

Parameters

StateName
Data Type: String
Name of state to assign to recipe

SetStatetoDefault()

This method assigns the default state to the recipe. If no default state is defined in the database, the recipe is not modified. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetStatetoDefault ()
```

SetTestApproval()

This method disables recipe test approval. A recipe cannot be run without either approval for production or test. The return code is a Boolean value that can be used for error handling.

Note: This function should only be called after the recipe has been saved using the Save or SaveAs methods. Any change made to a recipe automatically disables the approved for test flag.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetTestApproval ()
```

SetType()

This method assigns a valid type to a recipe. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetType (Name)
```

Parameters

Name
Data Type: String
Name of type to assign to recipe.

SetTypetoDefault()

This method assigns the default type to the recipe. If no default type is defined in the database, the recipe is not modified. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.SetTypetoDefault ()
```

ValidateRecipe()

This method validates the recipe and returns any error messages. If valid, function returns True and one message stating recipe is valid. The validation messages are returned to the ValidationMsgs parameter. The return code is a Boolean value that can be used for error handling.

For more information on the error codes returned, see *"Error Return Values" on page 388*.

Syntax

```
ReturnCode = RecipeVar.ValidateRecipe (ValidationMsgs)
```

Parameters

ValidationMsgs
Data Type: Object
One message for each error or a single message for valid recipe.

Examples

Because there is only a single object class in the recipe server and also due to the number of methods and properties available, specific examples were not created for each function. However, examples were created for each functional area of the recipe server and are included below.

General Recipe Manipulation

The following sample code illustrates loading a list of recipes, selecting a recipe by double-clicking the desired recipe in the list, and saving the recipe as a new recipe. The global variable for RecipeVar is defined so that it is available throughout the entire program. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure.

```
' Define required global variable
Dim RecipeVar As New wwRecipe
Private Sub Form_Load()
    ' Define required local variable
    Dim Recipes As Object
    Dim i As Integer
    lstRecipes.Clear
    ' Query for the available recipes and add to list
    Recipes = RecipeVar.GetRecipes
    For i = LBound(Recipes) To UBound(Recipes)
        lstRecipes.AddItem Recipes(i)
    Next i
End Sub
Private Sub lstRecipes_DblClick()
    ' Define required local variable
    Dim ReturnCode As Boolean
    ' Open, save, and close the selected recipe
    ReturnCode = RecipeVar.Open(lstRecipes.Text)
    ReturnCode = RecipeVar.SaveAs("New Recipe ID", "Recipe Author", _
       "Recipe Version Comments")
    ReturnCode = RecipeVar.Close
End Sub
```

Changing a Recipe Header

The following sample code illustrates opening the recipe with ID "Recipe123", changing a few of the header fields, saving the recipe with the same ID, and then approving the recipe for production. The global variable for RecipeVar is defined so that it is available throughout the entire program. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure.

```
' Define required global variable
Dim RecipeVar As New wwRecipe
Private Sub Form_Load()
    ' Define required local variable
    Dim ReturnCode As Boolean
    ' Open the desired recipe
    ReturnCode = RecipeVar.Open("Recipe123")
    ' Change the appropriate header fields
    RecipeVar.ProductId = "New Product ID"
```

```
    RecipeVar.ProductName = "New Product Name"
    RecipeVar.MinimumBatchSize = 100
    RecipeVar.MaximumBatchSize = 1000
    RecipeVar.DefaultBatchSize = 500
    ' Save and approve the recipe
    ReturnCode = RecipeVar.Save("Recipe Author", "Recipe Version Comments")
    ReturnCode = RecipeVar.SetProductionApproval
    ReturnCode = RecipeVar.Close
End Sub
```

Defining Recipe Equipment Requirements

The following sample code illustrates how to create a new recipe, add several process classes, add a second process instance, add the appropriate new transfer instances, and save the recipe as "Recipe123". The global variable for RecipeVar is defined so that it is available throughout the entire program. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure.

```
' Define required global variable
Dim RecipeVar As New wwRecipe
Private Sub Form_Load()
    ' Define required local variable
    Dim ReturnCode As Boolean
    ' Create a new recipe
    ReturnCode = RecipeVar.NewRecipe
    ' Add the appropriate process classes
    ReturnCode = RecipeVar.AddProcessClass("BULKTKS")
    ReturnCode = RecipeVar.AddProcessClass("REACTORS")
    ReturnCode = RecipeVar.AddProcessClass("MIXTANKS")
    ' Change name of default Reactors process instance
    ReturnCode = RecipeVar.ChangeProcessInstance("REACTORS", _
       "FIRST_REACTOR")
    ' Add a second reactor process instance
    ReturnCode = RecipeVar.AddProcessInstance("REACTORS", _
       "SECOND_REACTOR")
    ' Change name of default Bulks to Reactors transfer instance
    ReturnCode = RecipeVar.ChangeTransferInstance("BULKRXS", _
       "BULKS_FIRST_RX", "BULKTKS", "FIRST_REACTOR")
    ' Add a second Bulks to Reactors transfer instance
    ReturnCode = RecipeVar.AddTransferInstance("BULKRXS", _
       "BULKS_SECOND_RX", "BULKTKS", "SECOND_REACTOR")
    ' Change name of default Reactors to Mix Tanks transfer instance
    ReturnCode = RecipeVar.ChangeTransferInstance("RXSMIXTK", _
       "FIRST_RX_MIXTKS", "FIRST_REACTOR", "MIXTANKS")
    ' Add a second Reactors to Mix Tanks transfer instance
    ReturnCode = RecipeVar.AddTransferInstance("RXSMIXTK", _
       "SECOND_RX_MIXTKS", "SECOND_REACTOR", "MIXTANKS")
    ' Save the recipe
    ReturnCode = RecipeVar.SaveAs("Recipe123", "Recipe Author", _
       "Recipe Version Comments")
    ReturnCode = RecipeVar.Close
End Sub
```

Creating a Recipe Formula

The following sample code illustrates how to create a new recipe, add several ingredients, change the properties of each material accordingly, and save the recipe as "Recipe123". This example assumes that the materials already exist in the material database. Also, two instances of a particular material are added in this example and the second instance is modified appropriately. The global variable for RecipeVar is defined so that it is available throughout the entire program. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure.

```
' Define required global variable
Dim RecipeVar As New wwRecipe
Private Sub Form_Load()
    ' Define required local variables
    Dim ReturnCode As Boolean
    Dim InputMaterials(4) As String
    ' Create a new recipe
    ReturnCode = RecipeVar.NewRecipe
    ' Add required input materials to array
    InputMaterials(0) = "I0001"
    InputMaterials(1) = "I0002"
    InputMaterials(2) = "I0003"
    InputMaterials(3) = "I0003"
    ' Add the ingredient array to the recipe formula inputs
    ReturnCode = RecipeVar.AddInputMaterials(InputMaterials)
    ' Change the properties for input material I0001. Lists are One-based.
    ReturnCode = RecipeVar.SetInputValue(1, 50)
    ReturnCode = RecipeVar.SetInputValueType(1, wwActualValueType)
    ReturnCode = RecipeVar.SetInputTolerance(1, wwGeneralTolerance)
    ReturnCode = RecipeVar.SetInputHighDev(1, 3)
    ReturnCode = RecipeVar.SetInputLowDev(1, 4)
' Change the properties for input material I0002. Lists are One-based.
    ReturnCode = RecipeVar.SetInputValue(2, 25)
    ReturnCode = RecipeVar.SetInputValueType(2, wwActualValueType)
    ReturnCode = RecipeVar.SetInputTolerance(2, wwGeneralTolerance)
    ReturnCode = RecipeVar.SetInputHighDev(2, 2)
    ReturnCode = RecipeVar.SetInputLowDev(2, 1)
    ' Change the properties for the second instance
    ' of input material I0003. Lists are One-based.
    ReturnCode = RecipeVar.SetInputValue(4, 2.5)
    ReturnCode = RecipeVar.SetInputValueType(4, wwPercentValueType)
    ReturnCode = RecipeVar.SetInputTolerance(4, wwGeneralTolerance)
    ReturnCode = RecipeVar.SetInputHighDev(4, 1.5)
    ReturnCode = RecipeVar.SetInputLowDev(4, 2.5)
    ' Save the Recipe
    ReturnCode = RecipeVar.SaveAs("Recipe123", "Recipe Author", _
       "Recipe Version Comments")
End Sub
```

Building a Recipe Procedure

The following sample code illustrates how to open an existing recipe named "Recipe123", add a recipe procedure using three recipe levels, and save the recipe. This example assumes that the equipment and materials already exist in the recipe. The global variable for RecipeVar is defined so that it is available throughout the entire program. While not shown in these examples, specific application code can be added to provide error messages if the return code for any of the function calls indicates a failure.

The recipe consists of the following operations and phases:

Unit Procedure	Operation	Equipment Instance	Phase
Add Ingredients	Bulk Adds	Reactors	BulkAdd (I0001) BulkAdd (I0002) ManualAdd (I0003)
	Manual Adds	Reactors	
Process	Process	Reactors	AgitOn Heat (250 deg. C) Soak (250 deg. C, 30 min.) Cool (80 deg. C) AgitOff
Discharge	Transfer	MixTanks	Package (FG0001)

```
' Define required global variable
Dim RecipeVar As New wwRecipe
Private Sub Form_Load()
    ' Define required local variables
    Dim ReturnCode As Boolean
    Dim PhaseLabel As String
    ' Open the desired recipe
    ReturnCode = RecipeVar.Open("Recipe123")
    ' Add first unit procedure, operations, and phases.
    ' Lists are One-based.
    ReturnCode = RecipeVar.AddUnitProcedure("Add Ingredients", _
       "FIRST_REACTOR")
    ReturnCode = RecipeVar.AddOperation("Bulk Adds")
    ReturnCode = RecipeVar.AddTransferPhase("BULKS_FIRST_RX", "BULKADD", _
       PhaseLabel)
    ReturnCode = RecipeVar.SetInputParmMaterial("QUANTITY", 1, 50)
    ReturnCode = RecipeVar.AddTransferPhase("BULKS_FIRST_RX", "BULKADD", _
       PhaseLabel)
    ReturnCode = RecipeVar.SetInputParmMaterial("QUANTITY", 2, 25)
    ReturnCode = RecipeVar.AddOperation("Manual Adds")
    ReturnCode = RecipeVar.AddProcessPhase("MAN_ADD", PhaseLabel)
    ReturnCode = RecipeVar.SetInputParmMaterial("QUANTITY", 4, 25)
    ' Add second unit procedure, operation, and phases.
    ' Lists are One-based.
    ReturnCode = RecipeVar.AddUnitProcedure("Process", "FIRST_REACTOR")
    ReturnCode = RecipeVar.AddOperation("Process")
    ReturnCode = RecipeVar.AddProcessPhase("AGIT_ON", PhaseLabel)
    ReturnCode = RecipeVar.SetProcessVarParmValue("SPEED", 75)
    ReturnCode = RecipeVar.AddProcessPhase("HEAT", PhaseLabel)
    ReturnCode = RecipeVar.SetProcessVarParmValue("TEMP", 250)
    ReturnCode = RecipeVar.AddProcessPhase("SOAK", PhaseLabel)
    ReturnCode = RecipeVar.SetProcessVarParmValue("TEMP", 250)
    ReturnCode = RecipeVar.SetProcessVarParmValue("TIME", 30)
    ReturnCode = RecipeVar.AddProcessPhase("COOL", PhaseLabel)
    ReturnCode = RecipeVar.SetProcessVarParmValue("TEMP", 80)
    ReturnCode = RecipeVar.AddProcessPhase("AGIT_OFF", PhaseLabel)
    ' Add third unit procedure, operation, and phase. Lists are One-based.
    ReturnCode = RecipeVar.AddUnitProcedure("Discharge", "MIXTANKS")
```

```
    ReturnCode = RecipeVar.AddOperation("Discharge")
    ReturnCode = RecipeVar.AddTransferPhase("FIRST_RX_MIXTKS", "PACKAGE", _
       PhaseLabel)
    ReturnCode = RecipeVar.SetOutputParmMaterial("QUANTITY", 1, 100)
    ' Save the Recipe & Close it
    ReturnCode = RecipeVar.Save("Recipe Author", "Recipe Comments")
    ReturnCode = RecipeVar.SetProductionApproval
    ReturnCode = RecipeVar.Close
End Sub
```

Enumerated Constants

The following enumerated constants are available in the COM-based programming environment of choice. The enumerations enable programming to be completed without the direct knowledge of specific integer values.

In the Visual Basic environment, the enumerations appear as popup selection options. The example below shows the document viewing enumeration set as it appears in Visual Basic.

wwDocViewAck

The enumerated constant wwDocViewAck has the following names and values..

Name	Value
wwDocViewAckNone	0
wwDocViewEntryAck	1
wwDocViewExitAck	2

wwMaterialType

The enumerated constant wwMaterialType has the following names and values.

Name	Value
wwMatIngredient	0
wwMatIntermediate	1

Name	Value
wwMatFinishedGood	2
wwMatByProduct	3
wwMatOther	4

wwParamElementType

The enumerated constant wwParamElementType has the following names and values.

Name	Value
wwParamElemTarget	0
wwParamElemActual	1
wwParamElemHIDev	2
wwParamElemLODev	3
wwParamElemPreact	4
wwParamElemLotcode	5
wwParamElemHILim	4
wwParamElemLOLim	5
wwParamElemInputID	6
wwParamElemOutputID	2

wwParmValueType

The enumerated constant wwParamValueType has the following names and values.

Name	Value
wwPercentValueType	0
wwActualValueType	1

wwPhaseEntry

The enumerated constant wwPhaseEntry has the following names and values.

Name	Value
wwPhaseEntryAck	1
wwPhaseEntryDoneBy	3
wwPhaseEntryCheckBy	7

wwPhaseExit

The enumerated constant wwPhaseExit has the following names and values.

Name	Value
wwPhaseExitAck	8
wwPhaseExitDoneBy	24
wwPhaseExitCheckBy	56

wwPhaseParamDataClass

The enumerated constant wwPhaseParamDataClass has the following names and values.

Name	Value
wwAnalogParam	0
wwDiscreteParam	1
wwStringParam	2
wwEnumParam	3

wwPhaseParamType

The enumerated constant wwPhaseParamType has the following names and values.

Name	Value
wwInputParam	0
wwProcVarParam	1

Name	Value
wwOutputParam	2

wwPhaseParamViewType

The enumerated constant wwPhaseParamViewType has the following names and values.

Name	Value
wwEnableView	1
wwEnableChange	3
wwRequireChange	7

wwPhaseType

The enumerated constant wwPhaseType has the following names and values.

Name	Value
wwProcessPhase	0
wwAllocateProcessPhase	1
wwReleaseProcessPhase	2
wwTransferPhase	3
wwAllocateTransferPhase	4
wwReleaseTransferPhase	5

wwToleranceType

The enumerated constant wwPhaseToleranceType has the following names and values.

Name	Value
wwGeneralTolerance	0
wwRecipeTolerance	1
wwNoTolerance	2

wwTrainAttributeType

The enumerated constant wwTrainAttributeType has the following names and values.

Name	Value
wwTrainAttributeReal	0
wwTrainAttributeInt	1
wwTrainAttributeChar	2

wwUnitSelectionType

The enumerated constant wwUnitSelectionType has the following names and values.

Name	Value
wwUnitAutomatic	0
wwUnitManual	1

Error Return Values

The following error values may be returned from the recipe automation server:

0 = Fail

1 = Success

You can use the error values in conjunction with the ErrorMessage property to display error messages. For example, you can use the following code in Visual Basic for error handling.

```
Private Sub Form_Load()
    ' Define required variables
    Dim RecipeVar As New wwRecipe
    Dim ReturnCode As Boolean
    ' Run the desired method
    ReturnCode = RecipeVar.AddProcessClass("Invalid_Class")
    ' If the method fails, display a message box with the error
    '  message. The method can also be used as the If clause
    If Not (ReturnCode) Then
       MsgBox RecipeVar.ErrorMessage
    End If
End Sub
```

CHAPTER 7

Batch Function Interface Type Libraries

The Batch Function Interface (BFI) Type Libraries provide an interface specification to the procedures and data available from Batch Manager when using the COM BFI.

To completely understand the COM-based batch function interface type libraries, you must understand how Batch Manager runs. The batch function interface enables you to extend the functionality of Batch Manager by creating custom logic and having Batch Manager exercise this custom logic as it performs equipment allocation and batch execution. This custom logic can be included with certain strategic batch points of processing or hooks and is run by Batch Manager.

You use the batch function interface is through the type libraries described in this chapter and a COM-based programming environment such as Visual Basic.NET, C++ and C#.

Important: The COM-based batch function interface enables you to interact directly with the running of Batch Manager. Any errors or problems created using this interfaces can potentially hinder or halt the execution of Batch Manager. It is strongly recommended that you completely test any logic created using this interface approach and that all logic is created by appropriately trained personnel.

The following hooks are available using the COM-based batch function interface.

- Batch Initialization
- Batch Prepare
- Batch Complete
- Unit Procedure Prepare
- Unit Procedure Complete
- Operation Prepare
- Operation Complete
- Phase Prepare
- Phase Complete
- Evaluate Units for Allocation
- Evaluate Connections for Allocation
- Equipment Allocation Changes
- Logging Equipment Status Changes

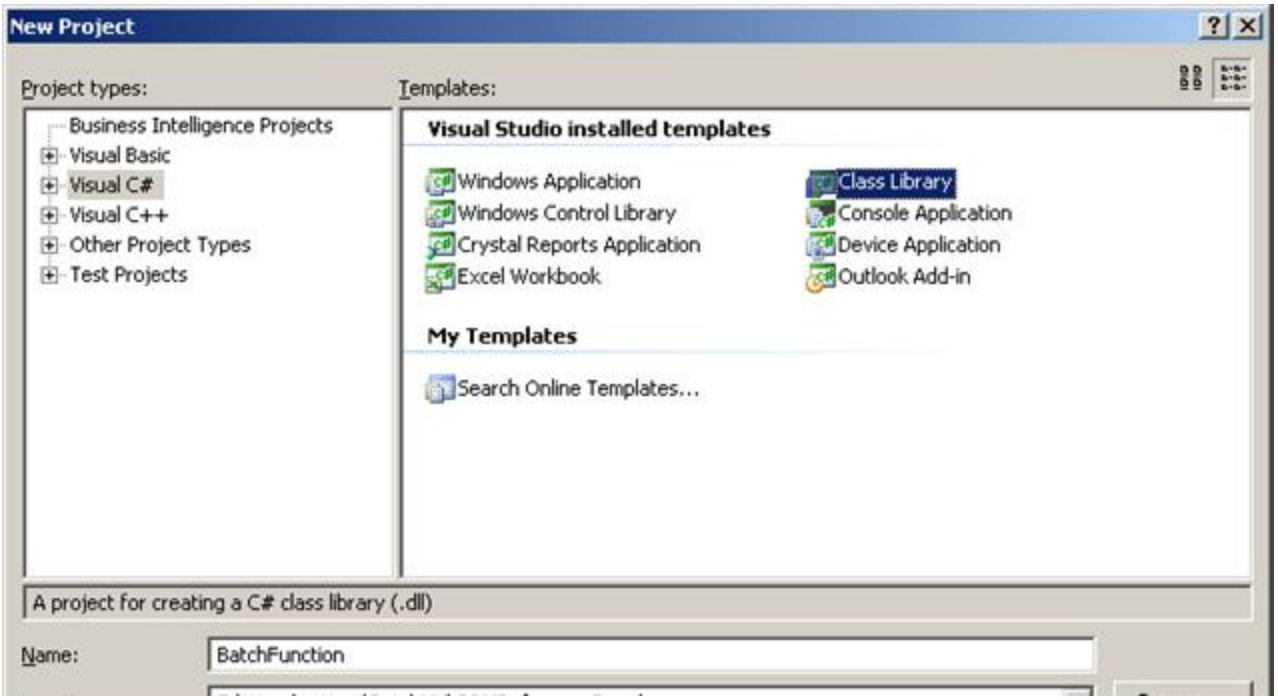
In This Chapter

Enabling the COM-Based Batch Function Interface

To use the COM-based batch function interface, complete the following steps. These steps require that you have properly installed the batch control system. This procedure uses Visual C# as the selected programming environment for all examples.

To enable the COM-based batch function interface

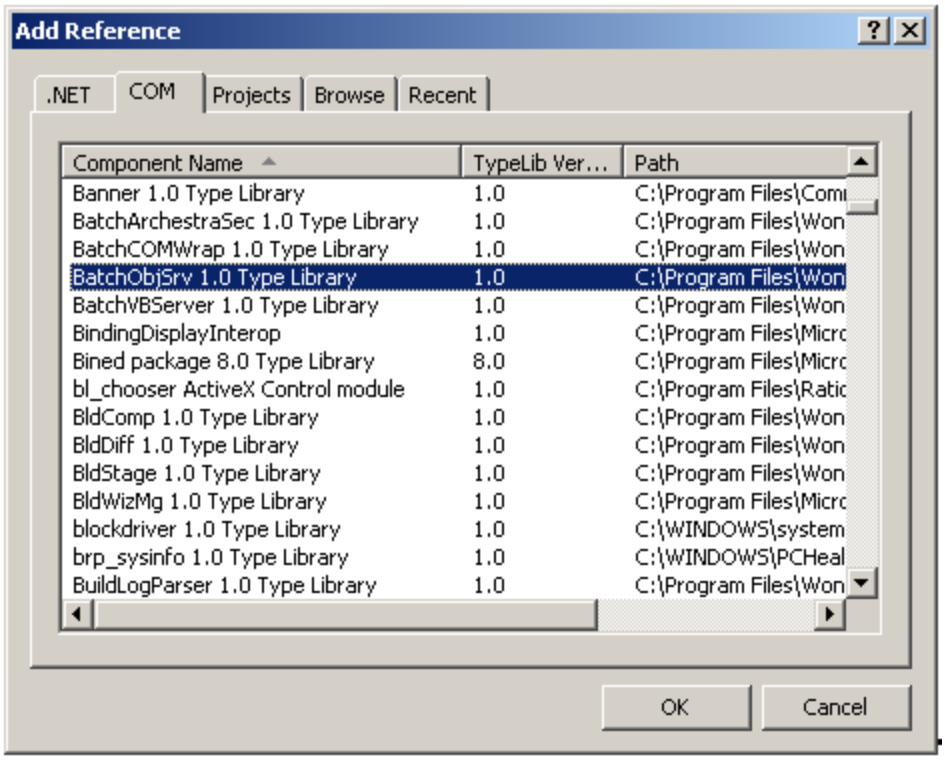
1. Start the COM-based programming environment of choice and begin a project to create an in-process server. In Visual C#, the project must be created as a class library.



2. Enable the batch function interface type libraries in the programming environment. The type libraries are:
   - BatchObjSrv 1.0 Type Library
   - BatchVBServer 1.0 Type Library

   Each library file is loaded during batch installation.

In Visual C#, the type libraries are enabled in the **References** dialog box.

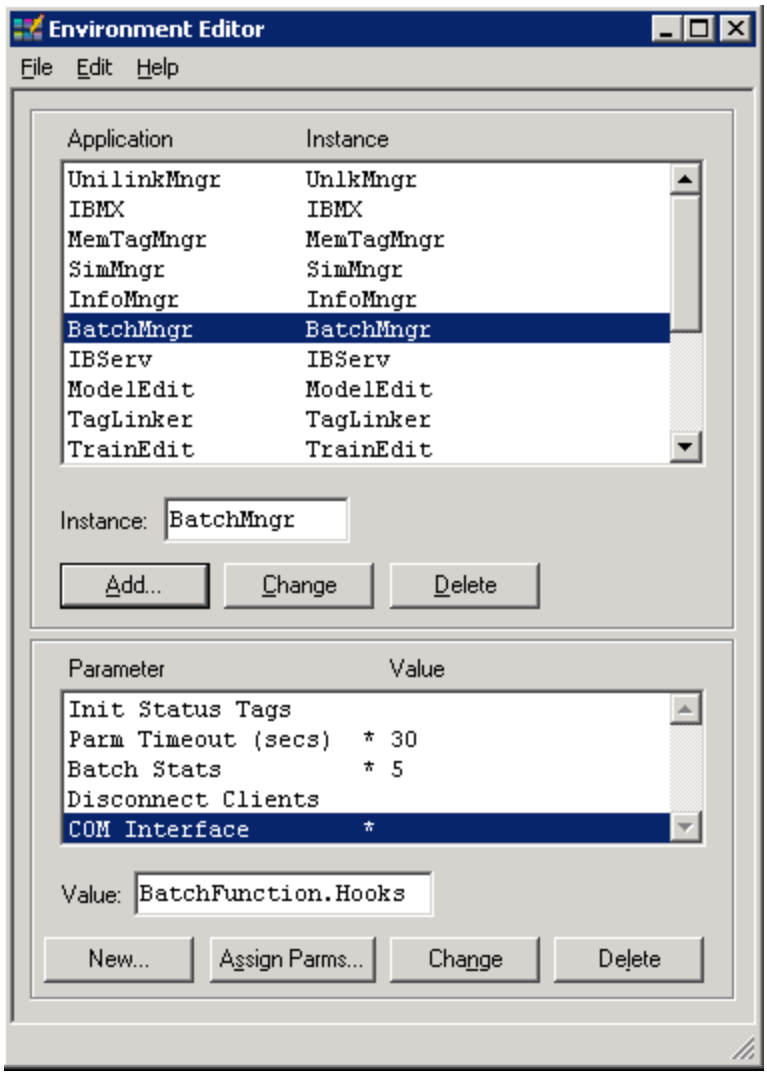


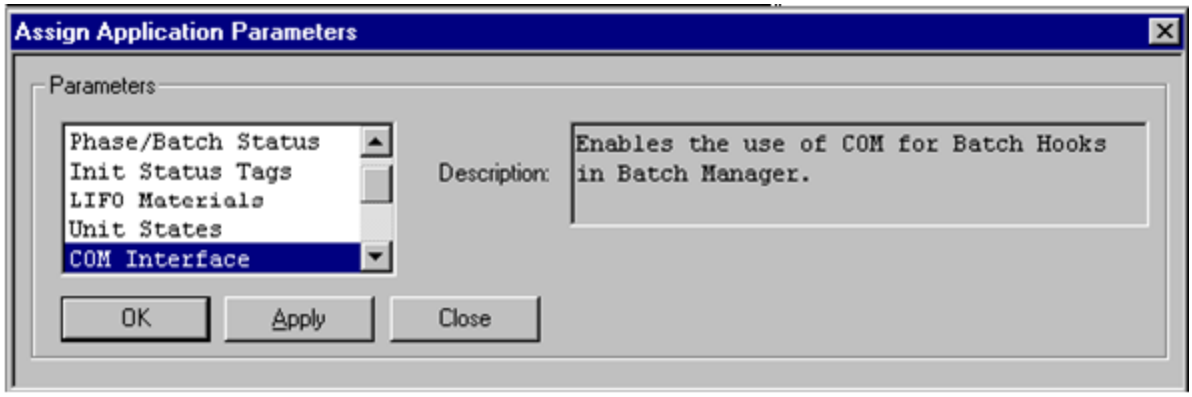
3. Create the DLL using the interface exposed by the type libraries.

   For more information on creating the batch function interface server, see *"Creating the Server"*.

4. Assign and configure the COM Interface application parameter for Batch Manager in the **Environment Editor** dialog box. The value assigned to this parameter corresponds to the project and class names defined in the programming environment. In the dialog boxes shown in the following figures, the project name is BatchFunction and the class name is Hooks. You must separate these names with a period (.) when you enter them as the application parameter value. The DLL in this example is named BatchFunction.dll.

Note: The value of the COM Interface parameter name is case-sensitive. The case of the value field should match the case of the project defined in the COM-based programming environment.





Batch Hooks Type Library

The following pages represent an alphabetical listing of all of the methods and properties available within the object classes defined in this type library including a description of each.

BatchHook Object Class

The batch hook object class contains the basic procedure calls that are issued by Batch Manager during normal operation. The available methods correspond to each of the available routines. In most cases, Batch Manager passes one or more objects to a routine as parameters. The routine logic can selectively use or alter some of this information and then pass the objects back to Batch Manager for continued operation. Several of the procedures require return values to direct Batch Manager operation.

Properties

There are no properties.

Methods

This section describes the methods available for the BatchHook class.

Batch Complete

The BatchHook_BatchComplete subroutine is called by Batch Manager at the completion of each batch.

Syntax

```
Private Sub BatchHook_BatchComplete(ByVal pIBatch As Object)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

Batch Initialization

The BatchHook_BatchInit function is called by Batch Manager for each batch that is initialized. This function requires a return value that indicates whether initialization should continue or stop. A return value of zero does not allow initialization. A return value of one permits initialization. The default return value is zero. When initializing more than one batch using the Initialize All option, this batch initialization routine is called for each batch.

Syntax

```
Private Sub BatchHook_BatchInit(ByVal pIBatch As Object) As Long
End Function
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

Batch Prepare

The BatchHook_BatchPrepare function is called by Batch Manager at the start of each batch. This function requires a return value that indicates whether the batch should continue with the start or not start at all. A return value of zero does not allow the batch to start. A return value of one permits the batch to start. The default return value is zero. A message property exists that you can use to define a custom error message that is displayed when the return value is zero. The message is not displayed if the return value is one.

Syntax

```
Private Sub BatchHook_BatchPrepare(ByVal pIBatch As Object) As Long
End Function
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

Equipment Allocation Change

The BatchHook_EquipAllocChange subroutine is called by Batch Manager when a unit or connection is allocated or released.

Syntax

```
Private Sub BatchHook_EquipAllocChange(ByVal pIBatch As Object, ByVal
pIEquipment As Object)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

pIEquipment
Data Type: BOEquipment
Equipment data provided by Batch Manager

Evaluate Connection for Allocation

The BatchHook_EvalConnAllocation function is called by Batch Manager when connection allocation is required. This function requires a return value that indicates whether the batch should allocate the specified connection. A return value of zero does not allocate the connection. A return value of one permits the connection to be allocated. The default return value is zero.

Syntax

```
Private Function BatchHook_EvalConnAllocation(ByVal pIBatch As Object, ByVal
pIEquipment As Object) As Long
End Function
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

pIEquipment
Data Type: BOEquipment
Equipment data provided by Batch Manager

Evaluate Unit for Allocation

The BatchHook_EvalUnitAllocation function is called by Batch Manager when unit allocation is required. This function requires a return value that indicates whether the batch should allocate the specified unit. A return value of zero does not allocate the unit. A return value of one permits the unit to be allocated. The default return value is zero.

Syntax

```
Private Function BatchHook_EvalUnitAllocation(ByVal pIBatch As Object, ByVal
pIEquipment As Object) As Long
End Function
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

pIEquipment
Data Type: BOEquipment
Equipment data provided by Batch Manager

Log Equipment Status Change

The BatchHook_LogEquipStatus subroutine is called by Batch Manager when the status of a unit or segment is changed and the information about the change is being written to history.

Syntax

```
Private Sub BatchHook_LogEquipStatus(ByVal pIEquipStatus As Object)
End Sub
```

Parameters

pIEquipmentStatus
Data Type: BOEquipStatus
Equipment status data provided by Batch Manager

Operation Complete

The BatchHook_OperationComplete subroutine is called by Batch Manager at the completion of each operation.

Syntax

```
Private Sub BatchHook_OperationComplete(ByVal pIBatch As Object, ByVal opname
As String, ByVal processinstance As String)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

opname
Data Type: String
Name of the operation

processinstance
Data Type: String
Name of the process instance assigned to the operation

Operation Prepare

The BatchHook_OperationPrepare subroutine is called by Batch Manager at the start of each operation.

Syntax

```
Private Sub BatchHook_OperationPrepare(ByVal pIBatch As Object, ByVal opname As
String, ByVal processinstance As String)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

opname
Data Type: String
Name of the operation

processinstance
Data Type: String
Name of the process instance assigned to the operation

Phase Complete

The BatchHook_PhaseComplete subroutine is called by Batch Manager at the completion of each phase.

Syntax

```
Private Sub BatchHook_PhaseComplete(ByVal pIBatch As Object, ByVal pIPhase As
Object, ByVal pIEquipment As Object)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

pIPhase
Data Type: BOPhase
Phase data provided by Batch Manager

pIEquipment
Data Type: BOEquipment
Equipment data provided by Batch Manager

Phase Prepare

The BatchHook_PhasePrepare subroutine is called by Batch Manager at the start of each phase.
Syntax

```
Private Sub BatchHook_PhasePrepare(ByVal pIBatch As Object, ByVal pIPhase As
Object, ByVal pIEquipment As Object)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

pIPhase
Data Type: BOPhase
Phase data provided by Batch Manager

pIEquipment
Data Type: BOEquipment
Equipment data provided by Batch Manager

Routine States

During startup, the BatchHook_RoutineStates subroutine is called by the Batch Manager to determine which hooks have been enabled. It is optionally called by the Batch Manager when it is shutting down. If the Batch Manager is shutting down, the pIRoutines parameter will either be NULL or empty. You must specifically request to have this method called on termination by enabling the wwRoutineStatesOnTerm constant during initialization.
Syntax

```
Private Sub BatchHook_RoutineStates(ByVal pIRoutines As Object)
End Sub
```

Parameters

pIRoutines
Data Type: BORoutines
Routine enabling object provided by Batch Manager

Unit Procedure Complete

The BatchHook_UnitProcedureComplete subroutine is called by Batch Manager at the completion of each unit procedure.
Syntax

```
Private Sub BatchHook_UnitProcedureComplete(ByVal pIBatch As Object, ByVal
unitprocedurename As String, ByVal processinstance As String)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

unitprocedurename
Data Type: String
Name of unit procedure

processinstance
Data Type: String
Name of the process instance assigned to the unit procedure

Unit Procedure Prepare

The BatchHook_UnitProcedurePrepare subroutine is called by Batch Manager at the start of each unit procedure.

Syntax

```
Private Sub BatchHook_UnitProcedurePrepare(ByVal pIBatch As Object, ByVal
unitprocedurename As String, ByVal processinstance As String)
End Sub
```

Parameters

pIBatch
Data Type: BOBatch
Batch data provided by Batch Manager

unitprocedurename
Data Type: String
Name of unit procedure

processinstance
Data Type: String
Name of the process instance assigned to the unit procedure

Batch Objects Type Library

The following pages represent an alphabetical listing of all of the methods and properties available within the object classes defined in this type library including a description of each.

BOBatch Object Class

The batch object class contains properties and methods that can be used to view information about the batch. Batch Manager writes this object as a parameter to several of the hook routines and reads the object when the routine is complete.

Properties

The following properties are available for the BOBatch class.

Batch

This property identifies the batch.

Data Type: String

Access: R

Campaign

This property identifies the campaign.

Data Type: String

Access: R

Lot

This property identifies the lot.

Data Type: String

Access: R

Message

This property is a custom error message that is only available for use with the BatchHook_BatchPrepare function when its return value is zero.

Data Type: String

Access: RW

Mode

This property is a batch mode enumeration. For Batch Mode enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchModeConstants

Access: R

PhaseCount

This property identifies the number of phases in the batch.

Data Type: Long

Access: R

RecipeId

This property identifies the recipe.

Data Type: String

Access: R

Size

This property identifies the batch size.

Data Type: Long

Access: R

Status

This property is a batch status enumeration. For Batch Status enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchStatusConstants

Access: R

Train

This property identifies the name of the train.

Data Type: String

Access: R

Methods

This section describes the method available for the BOBatch class.

Phase()

Populates a BOPhase object variable with information regarding a specific phase instance.
Syntax

```
BOPhaseObjectVar = BOBatchObjectVar.Phase ( index )
```

Parameters

index
Data Type: Long
Instance in phase list

BOEquipment Object Class

The batch equipment object class contains properties and methods that can be used to view information about the equipment being used by the batch. Batch Manager writes this object as a parameter to several of the hook routines and reads the object when the routine is complete.

Properties

The following properties are available for the BOEquipment class.

Instance

This property identifies the instance name.

Data Type: String

Access: R

Name

This property identifies the name of the equipment.

Data Type: String

Access: R

Segment Count

This property identifies the number of segments in the current connection, if applicable.

Data Type: Integer

Access: R

Type

This property identifies the equipment type enumeration. For Type enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchEquipmentTypes

Access: R

TypeRequest

This property identifies the equipment allocation type enumeration. For TypeRequest enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchEquipmentRequest

Access: R

Methods

This section describes the method available for the BOEquipment class.

Segment()

Returns the name of a specific segment instance.

Syntax

```
StringVar = BOEquipmentObjectVar.Segment ( index )
```

Parameters

index
Data Type: Long
Instance in phase list

BOEquipStatus Object Class

The batch equipment status object class contains properties and methods that can be used to view information about the equipment being logged to history following a status change. Batch Manager writes this object as a parameter to one of the hook routines and reads the object when the routine is complete.

Properties

The following properties are available for the BOEquipStatus class.

CheckBy

This property identifies the Check By User ID.

Data Type: String

Access: RW

Comment

This property identifies a user-defined comment for historical records.

Data Type: String

Access: RW

CurrRecipeId

This property identifies the current recipe.

Data Type: String

Access: R

DateTime

This property identifies the data and time value when the equipment status changed.

Data Type: Long

Access: R

DoneBy

This property identifies the Done By user ID.

Data Type: String

Access: RW

Field1 through Field 8

These properties identify user-defined fields.

Data Type: String

Access: RW

LastRecipeId

This property identifies the last recipe.

Data Type: String

Access: R

NewStatus

This property identifies the new equipment status.

Data Type: String

Access: R

OldStatus

This property identifies the old equipment status.

Data Type: String

Access: R

UnitorSegment

This property identifies the unit or segment equipment name.

Data Type: String

Access: R

Methods

None

BOParm Object Class

The batch parameter object class contains properties and methods that can be used to view information about the formula parameters associated with a specific phase in the batch. Batch Manager writes this object as a parameter to several of the hook routines and reads the object when the routine is complete.

Properties

The following properties are available for the BOParm class.

Actual

This property identifies the actual parameter value.

Data Type: String

Access: RW

DataClass

This property is a process variable parameter data class enumeration. For DataClass enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchProcVarDataClasses

Access: R

HighDev

This property identifies the parameter high deviation value.

Data Type: Double

Access: RW

HighLimit

This property identifies the parameter high limit value.

Data Type: Double

Access: RW

LotCode

This property identifies the parameter lot code value.

Data Type: String

Access: R

LowDev

This property identifies the parameter low deviation value.

Data Type: Double

Access: RW

LowLimit

This property identifies the parameter low limit value.

Data Type: Double

Access: RW

MaterialId

This property identifies the parameter material identification value.

Data Type: String

Access: RW

MaterialUofM

This property identifies the parameter material unit of measurement value.

Data Type: String

Access: RW

Name

This property identifies the parameter name.

Data Type: String

Access: R

Preact

This property identifies the input parameter preact value.

Data Type: Double

Access: RW

Target

This property identifies the target parameter value.

Data Type: String

Access: RW

ToleranceType

This property identifies the parameter material tolerance enumeration. For ToleranceType enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchToleranceType

Access: RW

Total

This property identifies the parameter material total quantity flag.

Data Type: Long

Access: RW

Type

This parameter identifies the parameter type enumeration. For Type enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchParmTypes

Access: R

ValueType

This parameter identifies the parameter material value type enumeration. For ValueType enumeration constants, see *"Enumerated Constants" on page 417*.

Data Type: BatchParmValueTypes

Access: R

Methods

None

BOPhase Object Class

The batch phase object class contains properties and methods that can be used to view information about the phases associated with a specific batch. Batch Manager writes this object as a parameter to several of the hook routines and reads the object when the routine is complete.

Properties

The following properties are available for the BOPhase class.

AppendtoInstructions

This property appends the unit and connection description from the Process Model database to the current phase instruction. A value of one enables this property.

DataType: Integer

Access: RW

CommentRequired

This property enables the comment required property for the current phase. A value of one enables this property.

DataType: Integer

Access: RW

Continue

This property enables the continue mode property for the current phase. A value of one enables this property.

DataType: Integer

Access: RW

DocViewPath

This property assigns a document to the current phase.

DataType: String

Access: RW

DocViewType

This property identifies the document viewing type enumeration. For DocViewType enumeration constants, see *"Enumerated Constants" on page 417*.

DataType: BatchDocViewAck

Access: RW

EntryAck

This property identifies the phase entry acknowledgment enumeration. For EntryAck enumeration constants, see *"Enumerated Constants" on page 417*.

DataType: BatchPhaseEntry

Access: RW

ExitAck

This property identifies the phase exit acknowledgment enumeration. For ExitAck enumeration constants, see *"Enumerated Constants" on page 417*.

DataType: BatchPhaseExit

Access: RW

InstanceName

This property identifies the equipment instance name.

DataType: String

Access: R

Instructions

This property identifies the instructions to add to current phase. There is no limit to the amount of text that can be added as an instruction.

DataType: String

Access: RW

Label

This property identifies the phase label.

DataType: String

Access: R

Name

This property identifies the phase name.

DataType: String

Access: R

ParamCount

This property identifies the number of formula parameters in the current phase.

DataType: Long

Access: R

Report

This property identifies the name of the report to assign to the current phase.

DataType: String

Access: R

Type

This property identifies a phase type enumeration. For Type enumeration constants, see *"Enumerated Constants" on page 417*.

DataType: BatchPhaseTypes

Access: R

Methods

This section describes the method available for the BOPhase class.

Param()

Populates a BOParm object variable with information regarding a specific formula parameter instance.

Syntax

```
BOParmObjectVar = BOPhaseObjectVar.Parm ( index )
```

Parameters

index
Data Type: Long
Instance in parameter list

BORoutines Object Class

The batch routines object class contains a single property that is used to enable one or more of the batch hook routines. Batch Manager writes this object as a parameter to this RoutineStates hook and reads the object when the hook is complete.

Properties

The following property is available for the BORoutines class.

RoutineState

This property identifies the batch hooks routine states enumeration. For RoutineState enumeration constants, see *"Enumerated Constants" on page 417*.

DataType: BatchRtnConstants

Access: RW

Property Cross-Reference Matrixes

The previous sections described the properties and methods that are available in the batch function interface type libraries. However, not all methods and properties are available for each routine in which the object is accessible. The tables that follow summarize the availability of methods and properties based on the routine from which the objects exist and the type of information being viewed.

BOBatch Object

The BOBatch object is passed to nine of the batch hook routines. However, the Phase method included with the BOBatch object returns data only when used within the batch prepare and batch complete routines. If called within one of the other routines, no data is returned.

BOBatch Object	Batch	Campaign	Lot	Message	Mode	Phase	PhaseCount	RecipeId	Formula	Size	Status	Train
BatchHook_BatchInit	X	X	X		X			X	X	X	X	X
BatchHook_BatchPrepare	X	X	X	X	X	X	X	X	X	X	X	X
BatchHook_BatchComplete	X	X	X	X	X	X	X	X	X	X	X	X
BatchHook_PhasePrepare	X	X	X		X			X	X	X	X	X
BatchHook_PhaseComplete	X	X	X		X			X	X	X	X	X
BatchHook_OperationPrepare	X	X	X		X			X	X	X	X	X
BatchHook_OperationComplete	X	X	X		X			X	X	X	X	X
BatchHook_UnitProcedurePrepare	X	X	X		X			X	X	X	X	X
BatchHook_UnitProcedureComplete	X	X	X		X			X	X	X	X	X
BatchHook_EvalUnitAllocation	X	X	X		X			X	X	X	X	X
BatchHook_EvalConnAllocation	X	X	X		X			X	X	X	X	X
BatchHook_EquipAllocChange	X	X	X		X			X	X	X	X	X

BOPhase Object

The BOPhase object is passed to four of the batch hook routines. However, the properties available depend on the type of phase defined in the recipe and the routine in question. When working in a batch routine, all properties and methods are available for process and transfer phases. Only a subset of the properties is available for recipe allocation and release phases. When working with the phase routines, only a limited number methods and properties are available.

BOPhase Object	AppendtoInstructions[1]	CommentRequired1	Continue1	DocViewPath	DocViewType	EntryAck1	ExitAck1	InstanceName	Label	Name	Param	ParamCount	Report	Type
BatchHook_BatchPrepare	X	X	X	X	X	X	X	X	X	X	X	X	X	X
BatchHook_BatchComplete	X	X	X	X	X	X	X	X	X	X	X	X	X	X

BOPhase Object	AppendtoInstructions[1]	CommentRequired1	Continue1	DocViewPath	DocViewType	EntryAck1	ExitAck1	InstanceName	Label	Name	Param	ParamCount	Report	Type
BatchHook_PhasePrepare										X	X	X		
BatchHook_PhaseComplete										X	X	X		

1 Available for process and transfer type phases only

BOParm Object

The BOParm object is passed to four of the batch hook routines. However, the properties available vary greatly depending on the type and data class of formula parameter defined for the phase as well as the routine in question. When working in a batch routine, all properties and methods are available with the exception of actual values, parameter limits, lot code, and preact. When working with the phase routines, nearly all of the methods and properties are available.

BOParm Object	Actual[1,2,3]	DataClass	HighDev[1,4]	HighLimit[4]	LotCode[1]	LowDev[1,4]	LowLimit[6]	MaterialId[1,2]	MtrlUofM[1,2]	Name[1,2,3]	Preact[1]	Target[1,2,3]	ToleranceType[1,3]	Total[1,2]	Type[1,2,3]	ValueType[1,2]
BatchHook_Batch Prepare		X	X			X		X	X	X		X	X	X	X	X
BatchHook_Batch Complete		X	X			X		X	X	X		X	X	X	X	X
BatchHook_Phase Prepare	X	X	X	X	X	X	X	X	X	X	X	X			X	
BatchHook_Phase Complete	X	X	X	X	X	X	X	X	X	X	X	X			X	

1 Available for input parameters

2 Available for output parameters

3 Available for all process variable parameters

4 Available for analog process variable parameters only

BOEquipment Object

The BOEquipment object is passed to five of the batch hook routines. However, the properties available vary greatly depending on the routine in question. When working in a phase routine, only the equipment name is available. When working in an allocation routine, availability is determined by the type of equipment being evaluated and the allocation being performed.

BOEquipment Object	Instance	Name	Segment	SegmentCount	Type	TypeRequest
BatchHook_PhasePrepare		X	X	X	X	
BatchHook_PhaseComplete		X				
BatchHook_EvalUnitAllocation	X	X			X	X
BatchHook_EvalConnAllocation	X	X	X	X	X	X
BatchHook_EquipAllocChange		X	X	X	X	X

BOEquipStatus Object

The BOEquipStatus object is passed to only one of the batch hook routines. All properties are available in this routine.

BOEquipStatus Object	CheckBy	Comment	CurrRecipeId	DateTime	DoneBy	Field1	Field2	Field3	Field4	Field5	Field6	Field7	Field8	CurrRecipeID	LastRecipeID	CurrFormulaName	LastFormulaName	NewStatus	OldStatus	UnitorSegment
BatchHook_LogEquipStatus	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

BORoutines Object

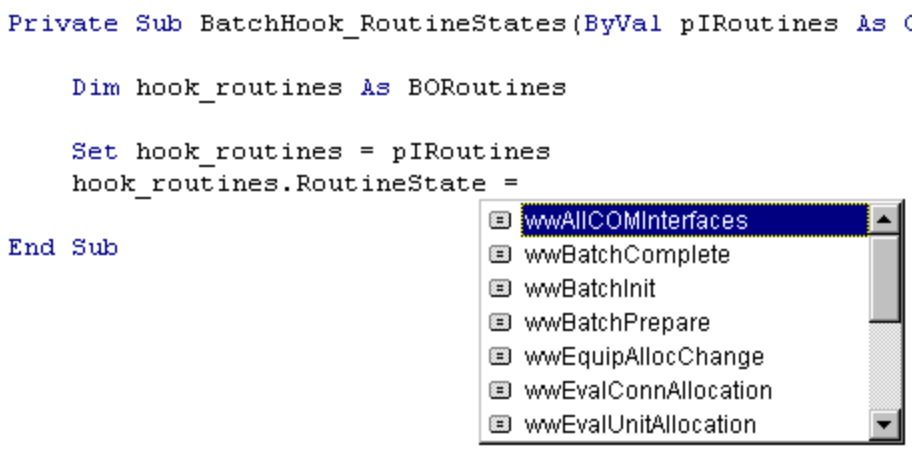
The BORoutines object is passed to only one of the batch hook routines. There is only one property for this object and it is available.

BORoutines Object	RoutineState
BatchHook_RoutineStates	X

Enumerated Constants

The following enumerated constants are available in the COM-based programming environment of choice. The enumerations enable programming to be completed without the direct knowledge of specific integer values.

In the Visual Basic environment, the enumerations appear as popup selection options. The following diagram shows the BatchRtnConstants enumeration set as it appears in Visual Basic.



BatchDocViewAcks

The following enumerated constants and values apply to the BatchDocViewAcks data type.:

wwDocViewEntryAck	1
wwDocViewExitAck	2

BatchEquipmentRequests

The following enumerated constants and values apply to the BatchEquipmentRequests data type.

wwAllocate	1
wwRelease	2

BatchEquipmentTypes

The following enumerated constants and values apply to the BatchEquipmentTypes data type.

wwUnit	0
wwConnection	1

BatchModeConstants

The following enumerated constants and values apply to the BatchModeConstants data type.

wwAutomatic	0
wwSemiAutomatic	1
wwManual	2

BatchParmTypes

The following enumerated constants and values apply to the BatchParmType data type.

wwInputParm	0
wwProcVarParm	1
wwOutputParm	2

BatchParmValueTypes

The following enumerated constants and values apply to the BatchParmValueType data type.

wwPercentValueType	0
wwActualValueType	1

BatchPhaseEntry

The following enumerated constants and values apply to the BatchPhaseEntry data type.

wwPhaseEntryAck	1
wwPhaseEntryDoneBy	3
wwPhaseEntryCheckBy	7

BatchPhaseExit

The following enumerated constants and values apply to the BatchPhaseExit data type.

wwPhaseExitAck	8
wwPhaseExitDoneBy	24
wwPhaseExitCheckBy	56

BatchPhaseTypes

The following enumerated constants and values apply to the BatchPhaseTypes data type.

wwProcessPhase	0
wwAllocateProcessPhase	1
wwReleaseProcessPhase	2
wwTransferPhase	3
wwAllocateTransferPhase	4
wwReleaseTransferPhase	5

BatchProcVarDataClasses

The following enumerated constants and values apply to the BatchProcVarDataClasses data type.

wwAnalogDataClass	0
wwDiscreteDataClass	1
wwStringDataClass	2
wwEnumDataClass	3

BatchRtnConstants

The following enumerated constants and values apply to the BatchRtnConstants data type.

wwBatchInit	1
wwBatchPrepare	2
wwBatchComplete	4
wwPhaseComplete	8
wwPhasePrepare	16
wwOperationPrepare	32
wwOperationComplete	64

wwEvalUnitAllocation	128
wwEvalConnAllocation	256
wwEquipAllocChange	1024
wwLogEquipStatus	2048
wwUnitProcedurePrepare	4096
wwUnitProcedureComplete	8192
wwAllCOMInterfaces	15871
wwRoutineStatesOnTerm	16384

BatchStatusConstants

The following enumerated constants and values apply to the BatchStatusConstants data type.

wwOpen	0
wwReady	1
wwWait	2
wwRun	3
wwDone	4
wwInterlock	5
wwAborted	6
wwHeld	7
wwAborting	8
wwContinue	9
wwLocking	10
wwLocked	11

BatchToleranceTypes

The following enumerated constants and values apply to the BatchToleranceTypes data type.

wwGeneralTolerance	0
wwRecipeTolerance	1
wwNoTolerance	2

Creating the Server

This section describes the fundamentals and documents some tips that you can use while creating the in-process server using the batch function interface type libraries.

You can create the server with any COM-based programming environment. This includes Visual Basic.NET, C++, and C#.

Creating the server requires that you define each of the batch hook routines, and enable the routines that are to be used.

Important: All batch hooks **must** be included in the server logic, but not all are required to be enabled. Compilation errors are generated if you omit one or more batch hook routines.

If other batch databases are to be accessed from within a batch hook routine, it is recommended that all objects and applications are created and started during server initialization rather than at the time required in the routine. This minimizes the delay to return control to Batch Manager.

For an example of accessing batch databases within the batch function interface, refer to the Material Quantity Check at Batch Initialization example later in this section.

The COM-based batch function interface provides a very powerful mechanism for programmatically interacting with the execution of Batch Manager. However, errors in the server interface causes Batch Manager to shut down. Therefore, troubleshooting the in-process server is very important to the success of the interface. A very helpful tip for troubleshooting is to enable the **Allow Service to Interact with Desktop** option for the Batch Manager service. This permits message boxes and other graphical user interface controls to be used in the application for diagnostic purposes. Be sure to remove all message boxes from the application when properly defined. All such controls halt Batch Manager processing until they are acknowledged.

Examples

The following examples illustrate how the batch function interface can be defined in the C# programming environment.

No Routines Enabled

This example contains the absolute minimum code required to use the COM-based batch function interface. In this example, all of the routines are present as required, but none of the routines are enabled.

```
using BATCHOBJSRVLib;
using BATCHVBSERVERLib;
namespace BatchFunction
{
      public class Hooks :IBatchHook
      {
            public int BatchInit(object pIBatch)
            {
            }
            public void BatchComplete(object pIBatch)
            {
            }
            public int BatchPrepare(object pIBatch)
            {
            }
            public void EquipAllocChange(object pIBatch, object pIEquipment)
            {
            }
            public int EvalConnAllocation(object pIBatch, object pIEquipment)
            {
            }
            public int EvalUnitAllocation(object pIBatch, object pIEquipment)
            {
```

```
            }
            public void LogEquipStatus(object pIEquipStatus)
            {
            }
            public void OperationComplete(object pIBatch, string opname, string
processinstance)
            {
            }
            public void OperationPrepare(object pIBatch, string opname, string
processinstance)
            {
            }
            public void PhaseComplete(object pIBatch, object pIPhase, object
pIEquipment)
            {
            }
            public void PhasePrepare(object pIBatch, object
      pIPhase, object pIEquipment)
            {
            }
            public void RoutineStates(object pIRoutines)
            {
            }
            public void UnitProcedureComplete(object pIBatch, string
unitprocedurename, string processinstance)
            {
            }

            public void UnitProcedurePrepare(object pIBatch, string
unitprocedurename, string processinstance)
            {
            }
      }
}
```

All Routines Enabled

This example contains the absolute minimum code required to use the COM-based batch function interface with all of the routines enabled. The logic required to continue Batch Manager processing is also provided in the functions that require return values.

```
using BATCHOBJSRVLib;
using BATCHVBSERVERLib;
namespace BatchFunction
{
      public class Hooks :IBatchHook
      {
            public int BatchInit(object pIBatch)
            {
                  return 1;
            }
            public void BatchComplete(object pIBatch)
            {
            }
            public int BatchPrepare(object pIBatch)
            {
                  return 1;
            }
```

```
            public void EquipAllocChange(object pIBatch, object pIEquipment)
            {
            }
            public int EvalConnAllocation(object pIBatch, object pIEquipment)
            {
                  return 1;
            }
            public int EvalUnitAllocation(object pIBatch, object pIEquipment)
            {
                  return 1;
            }
            public void LogEquipStatus(object pIEquipStatus)
            {
            }
            public void OperationComplete(object pIBatch, string opname, string
processinstance)
            {
            }
            public void OperationPrepare(object pIBatch, string opname, string
processinstance)
            {
            }
            public void PhaseComplete(object pIBatch, object pIPhase, object
pIEquipment)
            {
            }
            public void PhasePrepare(object pIBatch, object pIPhase, object
pIEquipment)
            {
            }
            public void RoutineStates(object pIRoutines)
            {
                  // Check to see if you are terminating
                  if (pIRoutines == null)
                  {
                  // You are terminating- Do cleanup and exit
                  return;
                  }
                  // Define required local variable
                  BORoutines hook_routines = null;
                  // Enable all of the required routines
                  hook_routines = (BORoutines)pIRoutines;
                  hook_routines.RoutineState =
BatchRtnConstants.wwAllCOMInterfaces |
      BatchRtnConstants.wwRoutineStatesOnTerm;
            }
            public void UnitProcedureComplete(object pIBatch, string
unitprocedurename, string processinstance)
            {
            }

            public void UnitProcedurePrepare(object pIBatch, string
unitprocedurename, string processinstance)
            {
            }
      }
}
```