GE VERNOVA

PROFICY® SOFTWARE & SERVICES

PROFICY BATCH EXECUTION 5.6

Custom Applications

GE VERNOVA

Proprietary Notice
The information contained in this publication is believed to be accurate and reliable. However, GE Vernova assumes no responsibilities for any errors, omissions or inaccuracies. Information contained in the publication is subject to change without notice.

Trademark Notices

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:
doc@ge.com

Table of Contents

About This Guide

The Custom Applications manual is intended for integrators and programmers who want to develop custom applications or integrate Batch Execution data with their manufacturing operations and Enterprise Resource Planning (ERP) systems by:

- Developing custom VBIS applications.
- Integrating the Batch Execution ActiveX controls into custom client applications.

This manual assumes the reader is proficient in the Microsoft® Visual Basic® or Visual C++™ programming languages.

Reference Documents

For related information, refer to the following document: VBIS Automation Reference.

Introduction

Batch Execution is based on OPC (OLE for process control), COM (Component Object Model), and DCOM (Distributed COM) standards to provide open and vendor-neutral data that you can integrate with other areas of your manufacturing business enterprise. Batch Execution provides the following integration features:

- VBIS OLE Automation interface. A collection of automation interfaces that allows external programs to monitor and control Batch Execution. For example, you can develop a campaign manager application using VBIS.

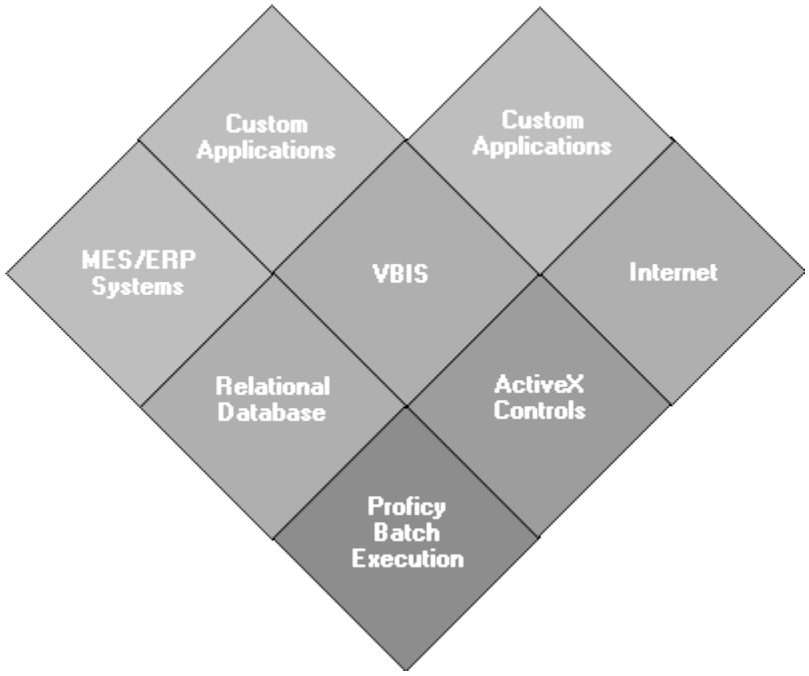
  Refer to the Developing VBIS Applications section for more information on the VBIS automation interface.

- ActiveX controls that you can use to build custom client applications or integrate into custom applications. You can run these controls in any ActiveX container, such as an internet browser, Visual Basic, Visual C++, and the Proficy iFIX WorkSpace.
    - For more information on the BatchList, BatchAdd, BatchRecipeList, BatchOperatorPromptsList, BatchBindingPromptsList, BatchAlarmList, BatchManualPhase, and BatchSFC ActiveX Controls, refer to the following sections:
        - Configuring the Batch Execution ActiveX Controls
        - BatchList ActiveX Control
        - BatchAdd ActiveX Control
        - BatchRecipeList ActiveX Control
        - BatchOperatorPromptsList ActiveX Control

        - BatchBindingPromptsList ActiveX Control
        - BatchAlarmList ActiveX Control
        - BatchCampaignClient ActiveX Control
        - BatchManualPhase ActiveX Control
        - BatchSFC ActiveX Control
        - BatchActivePhaseList ActiveX Control
    - For more information on the EWI ActiveX control, refer to Configuring the EWI ActiveX Control in the WorkInstruction Manual.
- The ability to store recipe and batch data in SQL, Oracle, and Access relational databases. Once the data is stored in your relational database, you can integrate this data into your ERP or MES systems.

  Refer to the Appendix A: VBIS Recipe Data Model and Appendix B: Batch Event Journal Data Model sections for more information on the relational database table structures.

The following figure illustrates the integration components of Batch Execution.



Batch Execution Integration

Before You Begin to Use VBIS

This software is key-protected. VBIS and the ActiveX controls will not work without a key. Be sure to install the key before beginning a VBIS project.

Integrating Batch Execution with External Systems

You can configure Batch Execution to store batch data and recipes in a relational database. This provides open access to your Batch Execution data and lets you integrate this data into your manufacturing operations and your Enterprise Resource Planning (ERP) system. Integrating all business areas of your manufacturing enterprise, from the plant-floor to the corporate-level systems, provides a single set of reliable data that you can use to make informed and timely business decisions.

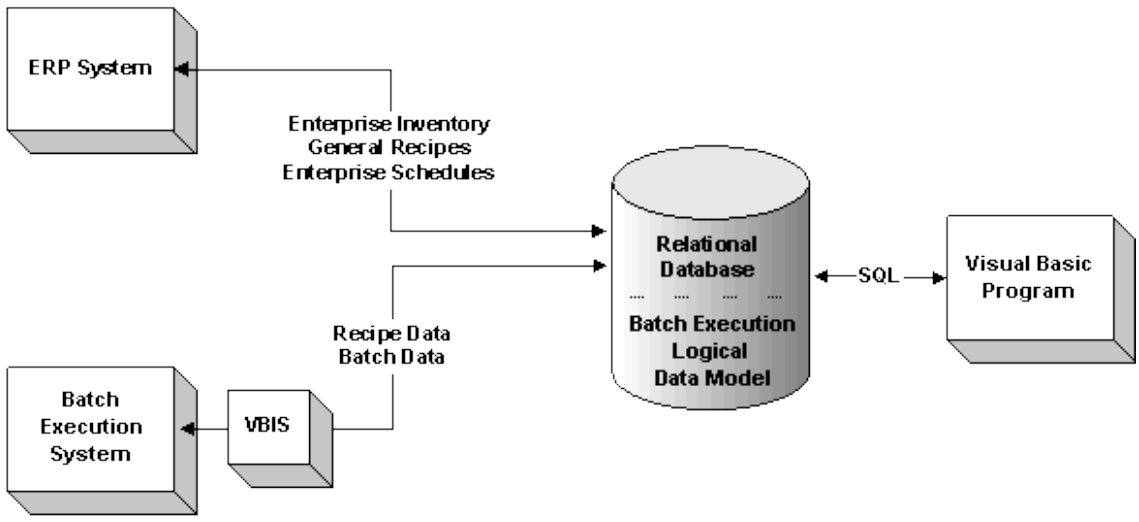
Integrating Batch Execution data with manufacturing systems can be accomplished in several ways. The Using VBIS as an Integration Tool section illustrates one integration path. In this scenario, ERP and Batch Execution data are stored in the same relational database. ERP data, including inventory, general recipes, and schedules are stored in the ERP tables. Batch Execution uses the relational database to store:

- Recipes in the tables comprising the VBIS Logical Data Model (LDM), which is part of the VBIS. The LDM is the table structures and rules that represent the storage of Batch Execution recipes in a relational database.
- Batch event data and transaction logs, by archiving the data to the relational database.

Using VBIS as an Integration Tool

As the following figure illustrates, you can incorporate the functions provided by VBIS to integrate Batch Execution data with your manufacturing enterprise data. For example, you can use the Scheduling Service provided in VBIS to build a Batch Execution campaign manager based on enterprise scheduling data. There are a number of other interfaces provided by VBIS.

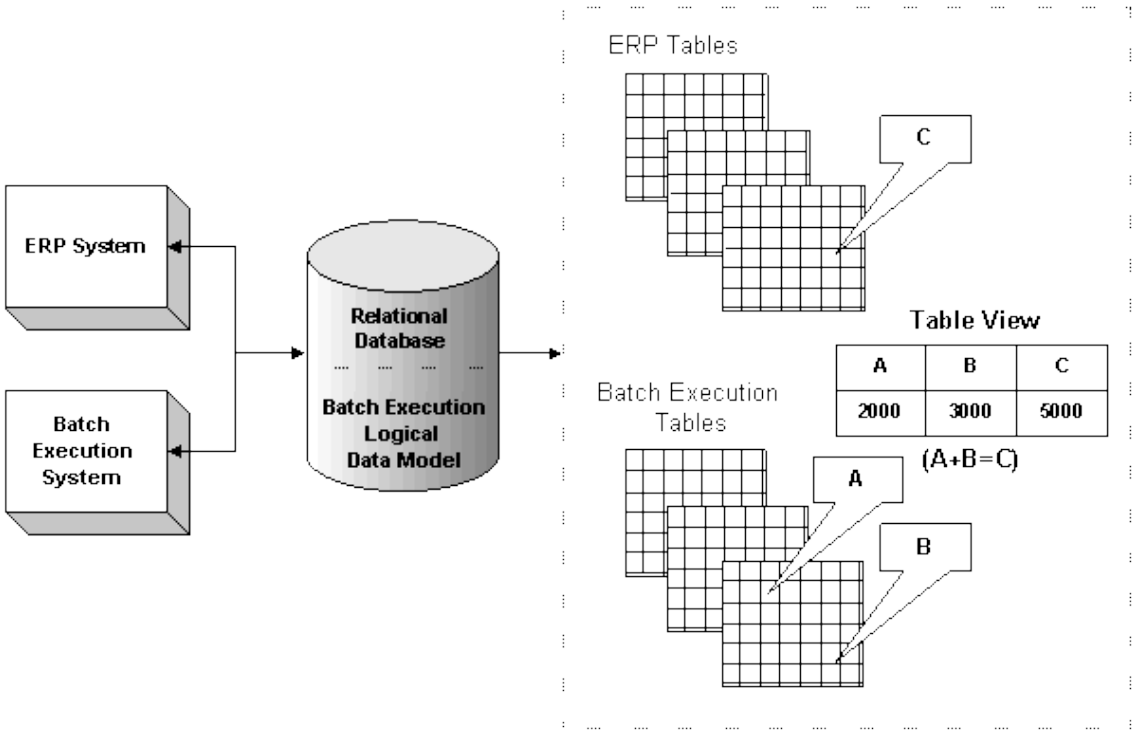
NOTE: *A relational database is not required to use VBIS.*



Typical ERP Integration Path

Storing the ERP and Batch Execution data in the same relational database allows dynamic and immediate data reconciliation. For example, assume Batch Execution produces a 2000 LB batch of toothpaste at Plant A. When the batch completes, Batch Execution archives this amount to the relational database. The ERP system needs to track production amounts for all plants. To update the ERP system, you can create *a table view* to extract the batch amount from the Batch Execution data

and join it to a common location that the ERP system uses to send and retrieve current data. The following figure illustrates this scenario.



Using a Table View to Integrate Batch Execution Data

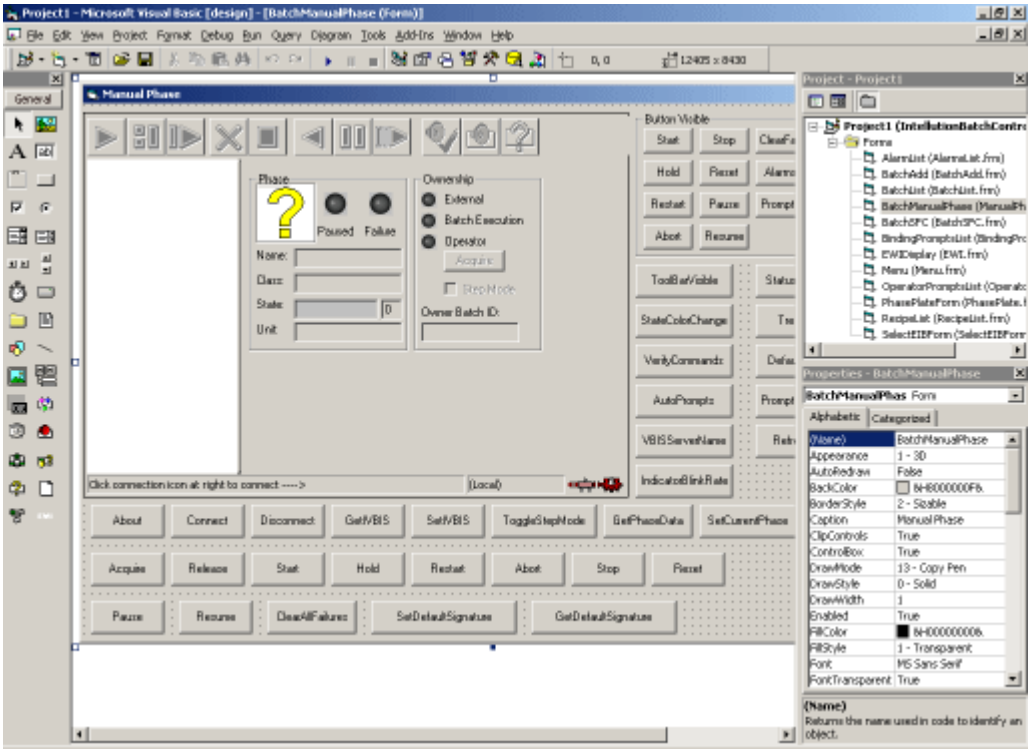
When the data is stored in separate relational databases, you can integrate the data by writing a SQL program to query the Batch Execution and ERP data and then reconcile the data.

Sample Applications for ActiveX Controls and VBIS

You can find the samples of VBIS and the ActiveX controls in the Batch Execution Samples folder. If you installed the product to the default location, this folder is: c:\Program Files\Proficy\Proficy Batch Execution\Samples.

The sample VBIS applications are written in Visual Basic, and include a VBIS API, a sample campaign manager that uses VBIS, plus an EWI test application.

The sample ActiveX control applications are written in both Microsoft Visual Basic and Visual C++. You can use these sample projects to base your own projects on, or as a test project. The following graphic illustrates an example of the Visual Basic project for the ActiveX controls.



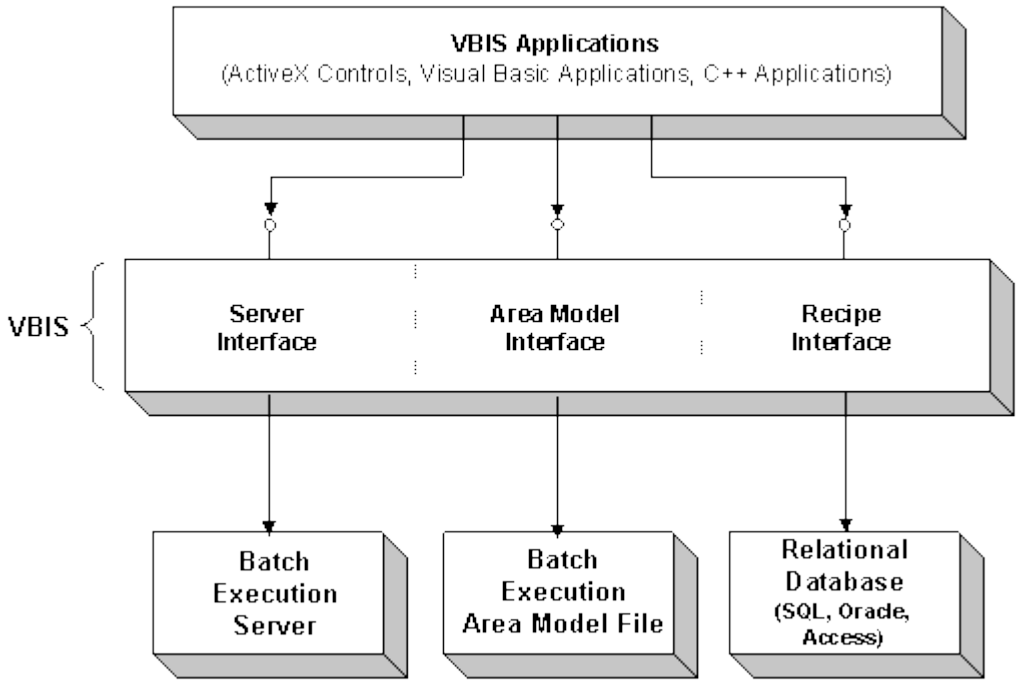
Sample Project Using ActiveX Controls in Visual Basic - BatchManualPhase Control Form

Developing VBIS Applications

VBIS is a collection of automation interfaces that allows external programs to monitor and control Batch Execution. Within VBIS, services are provided in a number of functional areas including recipes, the area model, scheduling, and batch execution. Using VBIS, you can develop custom applications such as:

- A campaign manager.
- A legacy system connection manager.
- A recipe editor that manipulates data from a custom external system.

The following figure illustrates the VBIS architecture.



VBIS Architecture

ActiveX Controls

Batch Execution supplies a set of ActiveX controls that you can use to extend the power of Batch Execution. You can use these controls as an alternative to the Batch Execution Client application. You can "drop" these controls into any ActiveX control container, such as:

- Proficy iFIX WorkSpace
- Web browsers, such as Internet Explorer
- Visual Basic

You can also integrate these controls into custom applications. Each control provides programmability through OLE Automation. This allows you to take advantage of the ActiveX control's features through the Visual Basic or Visual C++ programming languages.

For information on this ActiveX control...	**Refer to...**
BatchList	BatchList ActiveX Control
BatchAdd	BatchAdd ActiveX Control
BatchRecipeList	BatchRecipeList ActiveX Control
BatchOperatorPromptsList	BatchOperatorPromptsList ActiveX Control

For information on this ActiveX control...	Refer to...
BatchBindingPromptsList	BatchBindingPromptsList ActiveX Control
BatchAlarmList	BatchAlarmList ActiveX Control
BatchCampaignClient	BatchCampaignClient ActiveX Control
BatchManualPhase	BatchManualPhase ActiveX Control
BatchSFC	BatchSFC ActiveX Control
BatchActivePhaseList	BatchActivePhaseList ActiveX Control
EWIX	Configuring the EWI ActiveX Control in the WorkInstruction Manual

Configuring and Using the VBIS Server

The VBIS applications that you develop and the Batch Execution ActiveX controls connect to the VBIS Server. The VBIS Server, VBIS.EXE, is installed into the Batch Execution Bin directory, typically C:\Program Files\Proficy\Proficy Batch Execution\BIN. The VBIS Server:

- Comes installed as a Windows service under the account you specify during install. You should not change the account under which the VBIS Server runs. If you do not specify an account during the install, the BatchExecutive account is used by default. The default password for the BatchExecutive account is batchrules.
- Can reside on any node.

The following subsections describe configuration tasks for using the VBIS Server.

- Configuring VBIS Communications to the Batch Execution Server
- Optimizing Multiple VBIS Client Performance
- Configuring Client Communications to VBIS
- How to Register VBIS

Configuring VBIS Communications to the Batch Execution Server

The VBIS Server acts as a client to the Batch Execution Server. Therefore, to configure the VBIS Server to communicate to a local or remote node on which the Batch Execution Server resides, you need to specify the computer that is running the Batch Execution Server.

To specify the computer running the Batch Execution Server:

1. Start the Batch Execution Client.
2. Click the Server Configuration and Defaults button on the toolbar to open the System Configuration and Defaults screen.
3. Click the Remote Server option.
4. Select the Remote Server Configuration tab.
5. Click the Add Server button in the Remote Server Configuration screen. The Add a Remote Server dialog box appears.
6. In the Server Name field, enter the name of the remote server or click the Browse (...) button to select a server.

   NOTE: *For the Batch Execution ActiveX controls, you can specify the VBIS server from the VBIS Server property page.*

7. In the Alias field, enter an alias for the server name. You must enter an alias with every server, even if it is the same name as the server.

   NOTE: *The alias cannot exceed 30 characters. It can contain the numbers: 0-9, the letters: A-Z and a-z, the underscore character, and a space character. The underscore and space characters cannot appear at the beginning of an entry, or alone. Duplicate alias names are not allowed in the list.*

8. Click the Default check box if you want this remote server to act as the default server. You can only assign one server as the default server.
9. Click OK.

Optimizing Multiple VBIS Client Performance

If you have multiple clients (ActiveX controls for VBIS applications) connecting to the VBIS Server that are starting and stopping the VBIS Server on a regular basis, you may want to configure the VBIS Server to always run, rather than making the VBIS clients start and stop the VBIS Server.

To configure the VBIS Server to always run:

NOTE: *These steps assume that VBIS is registered to run as a Windows service. Normally, VBIS is automatically registered to run as a service during the product install. However, you can manually override this setting and run VBIS as a regular server. For more information, refer to the How to Register VBIS section.*

1. From the Windows Control Panel's Administrative Tools, select the Services icon. The Services dialog box appears.
2. Select GE Intelligent Platforms Batch Integration Services in the list box.
3. Click the Startup button. The Service dialog box appears.
4. In the Startup Type area, select the Automatic button.
5. Click OK to return the Services dialog box.
6. Click Close.

Configuring Client Communications to VBIS

The Batch Execution ActiveX Controls and applications developed using the VBIS Automation Interfaces are clients to the VBIS Server. They can reside on the same node as the VBIS Server or on a remote node. The following subsections describe how to configure VBIS clients to communicate with the VBIS Server.

To configure the Batch Execution ActiveX Controls to communicate with the VBIS Server:

1. Open the control in any OLE container.
2. Right-click on the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the VBIS Server property page.
4. Select Local if the VBIS Server is running on the same machine as the control.
5. Select Remote if the VBIS Server is running on a different machine from the control.
6. If you selected Remote, click the Browse button to select the computer name on which the VBIS Server is running. You can also manually enter the machine name, IP address, or other DCOM-compatible machine name.
7. Select the Connect on Startup check box if you want the control to automatically connect to the VBIS Server when the control is instantiated in a run-time environment.
8. In the Refresh Rate field, enter the interval at which you want to refresh data.
9. Click OK to save your changes.

   NOTE: *For more information on the VBIS Server property page, refer to Configuring the Batch Execution ActiveX Controls.*

Configuring Third-Party Applications to Communicate with the VBIS Server

If your third-party applications are running remotely, that is on a different machine than where the VBIS Server is running, you need to configure remote communications to the VBIS Server.

To configure applications programmed in Visual C++, specify a remote machine name in the CoCreateInstanceEx() call. For example:

```
USES_CONVERSION;
COSERVERINFO serverinfo;
COSERVERINFO* pServerInfo;
serverinfo.dwReserved1 = 0;
serverinfo.dwReserved2 = 0;
CString strServerName = "REMOTE_MACHINE";
serverinfo.pwszName = A2W(strServerName);
serverinfo.pAuthInfo = NULL;
pServerInfo = &serverinfo;
MULTI_QI qi = {&IID_IVBIS8, NULL, 0};
hr = CoCreateInstanceEx (CLSID_VBIS8,
                         NULL,
                         CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER,
                         pServerInfo,
                         1,
                         &qi);
```

An example of the Visual Basic code is as follows:

```
Dim VBISApp As VBIS8
CreateObject ("GE.VBIS.7","REMOTE_MACHINE")
```

To configure third-party applications that were programmed in Visual Basic to connect to a remote VBIS Server, you must configure DCOM to point to the remote machine.

NOTE: *You must have Administrator rights to configure these DCOM settings.*

To configure the DCOM to point to the remote machine:

1. Log in to Windows as an administrator, if you are not already.
2. Click the Start button, and point to Run. The Run dialog box appears.
3. Enter dcomcnfg.exe and click OK. The Component Services Microsoft® Management Console (MMC) snap-in appears.
4. Double-click the Component Services folder.
5. Double-click the Computers folder within the Component Services folder.
6. Double-click the My Computer (or computer name) icon.
7. Double-click the DCOM Config folder to display the list of DCOM applications.
8. Right-click the GE Intelligent Platforms Batch Integration Services application, and select Properties from the right-click menu. The Properties dialog box appears.
9. Select the Location tab.
10. Select the *Run application on the following computer* check box, and remove all other check marks.
11. Enter the computer name where the VBIS Server is running. You can use the Browse button to search for the computer on the network.
12. Click Apply.
13. Click OK.

How to Register VBIS

Normally, you do not need to register the VBIS application because it is automatically registered during the product install. However, there may be instances during troubleshooting when you need to re-register your VBIS application. There are two ways to register and run VBIS:

- As a service (the default)
- As a regular server

The steps below explain how to do both.

To register VBIS to run as a service:

1. Confirm that the product key you purchased includes the VBIS feature.
2. From the Windows Task Manager, make sure that VBIS is not running.

3. From a command prompt, navigate to the folder where VBIS.exe is located. If you installed to the default location, this folder is: C:\Program Files\Proficy\Proficy Batch Execution\BIN.
4. Type for the following command to unregister VBIS:

   ```
   vbis /unregister
   ```

5. Type the following command to register VBIS:

   ```
   vbis /service
   ```

   NOTE: *When you start VBIS as a service, no message box appears to indicate if there was a success or failure.*

To register VBIS to run as a regular server:

1. Confirm that the product key you purchased includes the VBIS feature.
2. From the Windows Task Manager, make sure that VBIS is not running.
3. From a command prompt, navigate to the folder where VBIS.exe is located. If you installed to the default location, this folder is: C:\Program Files\Proficy\Proficy Batch Execution\BIN.
4. Type for the following command to unregister VBIS:

   ```
   vbis /unregister
   ```

5. Type the following command to register VBIS:

   ```
   vbis /regserver
   ```

   NOTE: *When you run VBIS as a regular server, VBIS runs under the account you are logged in under. When you log off, VBIS also stops.*

Configuring the VBIS Automation Interface

You can program VBIS applications in either the Visual Basic or Visual C++ programming languages. The following sections describe how to use the VBIS interface within each programming environment:

- Programming with VBIS in Visual Basic
- Programming with VBIS in C++

Programming with VBIS in Visual Basic

VBIS provides a type library that Visual Basic can reference (VBISSRV.TLB). It uses a dual OLE Automation interface that allows both early and late binding. While in the editor, Visual Basic will recognize all the VBIS objects, methods, and properties.

To work with VBIS objects within your Visual Basic program, you must first include it as a reference into your Visual Basic program.

To include the VBIS reference in Visual Basic:

1. On the Project menu, click References.
2. Check for the entry labeled Intellution VisualBatch Integrated Services.

3. If you don't find this reference, click the Browse button.
4. Navigate to the Program Files\Proficy\Proficy Batch Execution\TOOLS\VBIS directory.
5. Select VBISSRV.TLB and click OK.

You will now be able to use the VBIS defined types within your Visual Basic program.

Programming with VBIS in C++

The interface from C++ to VBIS8 interface requires you to include the following three files that are supplied with VBIS8:

- VBISSRV_I.C
- VBISSRV.H
- VBISSRV.TLB

These files provide the interface IDs and the interface method and property declarations. They are located in the Tools folder. If you installed to the default location, this folder is: C:\Program Files\Proficy\Proficy Batch Execution\Tools\VBIS.

NOTE: *VBIS8 is the top-level interface in the VBIS automation interface hierarchy.*

VBIS Automation Interface Hierarchy

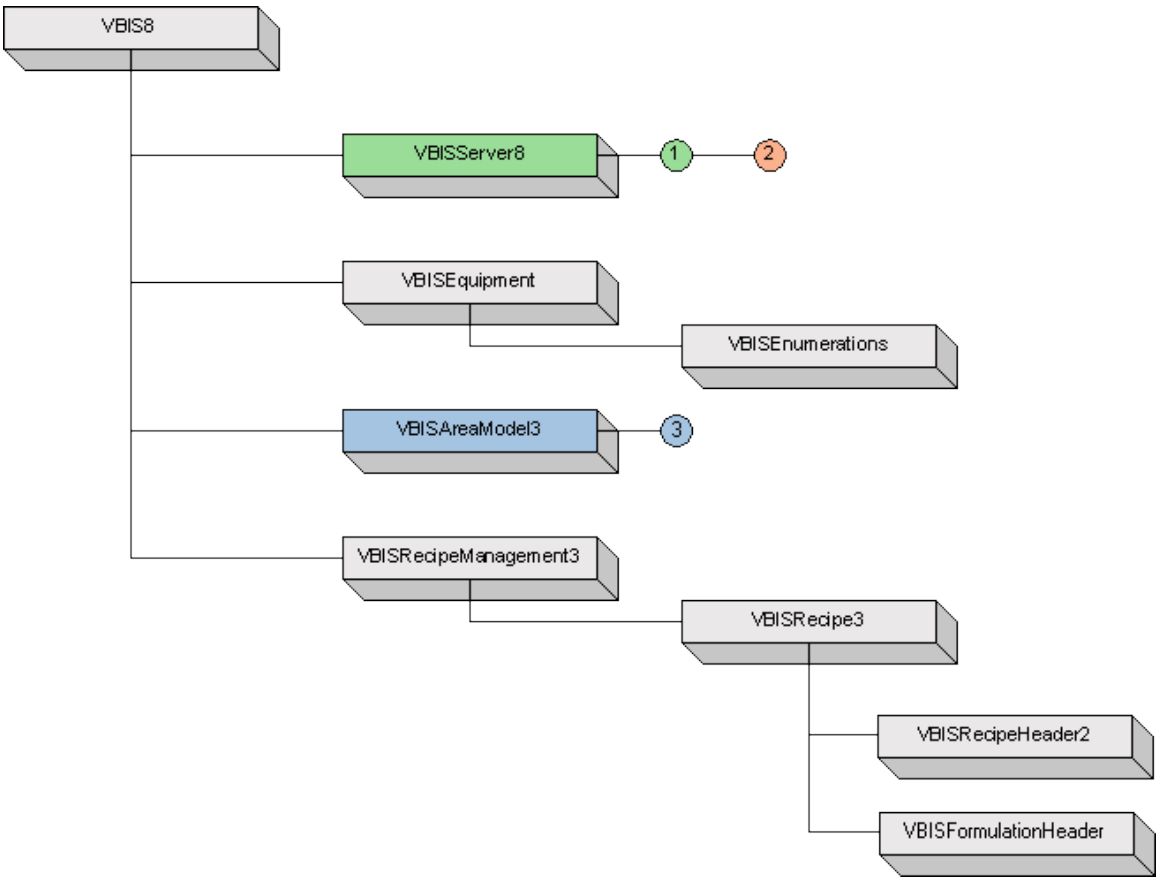
The VBIS automation interface is made up of a hierarchy of interfaces that provides access to Batch Execution. The following sections describe each interface hierarchy.

For More Information

For details on each interface, the properties and methods used by each interface, plus numerous code examples, refer to the VBIS Automation Reference. This Help is integrated into Visual Basic. This allows you to get context-sensitive Help for VBIS functions while you are coding. To get Help on VBIS while you are in Visual Basic, select a VBIS function and press F1.

VBIS8 Interface Hierarchy

The VBIS automation interface is comprised of several hierarchical objects. The top-level object is the VBIS8 interface. From VBIS8 you can instantiate its lower-level objects. The following figure illustrates this hierarchy.



VBIS8 Interface Hierarchy

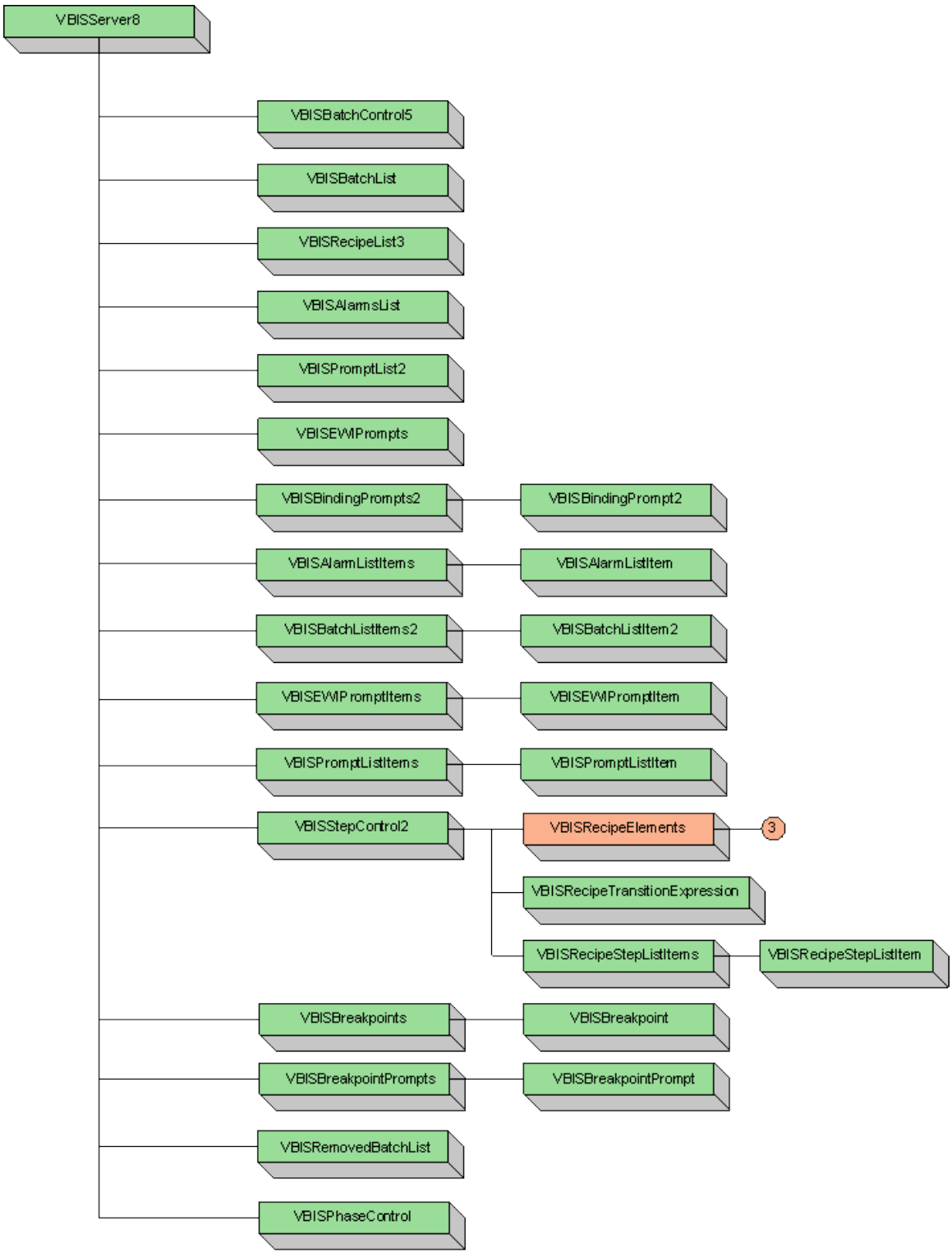
The following table describes each interface in the VBIS8 hierarchy.

VBIS8 Interfaces	
Interface	**Description**
VBISServer8	Used to communicate with the Batch Execution Server.
VBISEquipment	Provides access to enumerations defined in the area model.
VBISEnumerations	Provides the list (collection) of enumerations for the parameter, if applicable.
VBISAreaModel3	Provides access to the equipment defined in the area model.
VBISRecipeManagement3	Allows you to create and maintain recipes.

VBIS8 Interfaces	
Interface	**Description**
VBISRecipe3	Provides access to the recipe data stored in the Batch Execution Server.
VBISRecipeHeader2	Provides access to the recipe header defined in the Batch Execution Recipe Editor.

VBISServer8 Interface Hierarchy

As the following figure illustrates, the VBISServer8 interface provides access to interfaces that let you manipulate the Batch Execution Server.



VBISServer8 Hierarchy

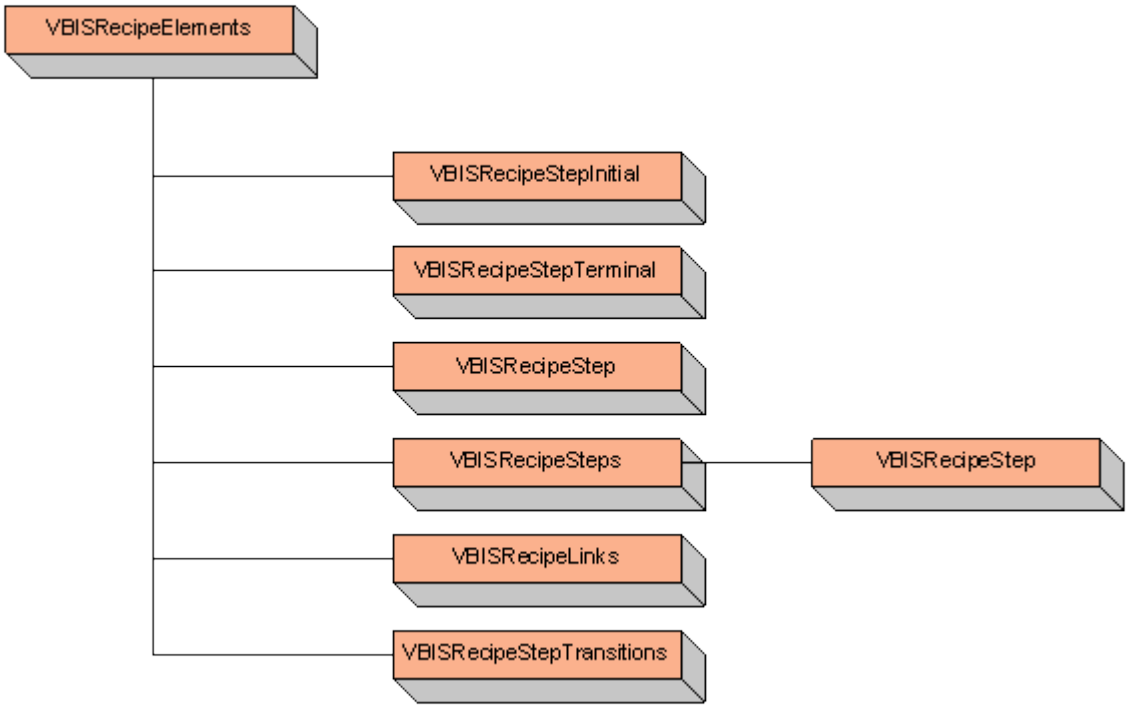
The following table describes each interface in the VBISServer8 hierarchy.

VBISServer8 Interfaces	
Interface	**Description**
VBISBatchControl5	Provides access to and control of batches executing on the Batch Execution Server. Using this object, you can add and control batches in the Batch Execution Client's batch list or a third-party client application.
VBISBatchList	Provides access to batch list data stored in the Batch Execution Server.
VBISRecipeList3	Provides access to recipe list data stored in the Batch Execution Server.
VBISAlarmsList	Provides access to alarm list data stored in the Batch Execution Server.
VBISPromptList2	Provides access to operator prompt list data stored in the Batch Execution Server.
VBISEWIPrompts	Provides access to the EWI prompts stored in the Batch Execution Server.
VBISBindingPrompts2	A collection of VBISBindingPrompt2 objects.
VBISBindingPrompt2	Provides access to binding prompts.
VBISAlarmListItem	Provides access to alarm list data stored in the Batch Execution Server.
VBISAlarmListItems	A collection of VBISAlarmListItem objects.
VBISBatchListItem2	Provides access to batch list data stored in the Batch Execution Server.
VBISBatchListItems2	A collection of VBISBatchListItem2 objects.

VBISServer8 Interfaces	
Interface	**Description**
VBISEWIPromptItem	Provides access to EWI prompts stored in the Batch Execution Server.
VBISEWIPromptItems	A collection of VBISEWIPromptItem objects.
VBISPromptListItem	Provides access to operator prompt list data stored in the Batch Execution Server.
VBISPromptListItems	A collection of VBISPromptListItem objects.
VBISStepControl2	Provides access to the manual phase control to phases.
VBISRecipeElements	Describes all of the elements of the sequential function chart (SFC). See VBISRecipeElements Interface Hierarchy for more information.
VBISRecipeStepListItem	Provides access to the SFC steps that describe the logic of the recipe.
VBISRecipeStepListItems	A collection of VBISRecipeStepListItem objects.
VBISRecipeTransitionExpession	Provides access to the transitions in a recipe.
VBISBreakpoints	A collection of VBISBreakpoint objects.
VBISBreakpoint	Provides access to the transition breakpoints.
VBISBreakpointPrompts	A collection of VBISBreakpointPrompt objects.
VBISBreakpointPrompt	Provides access to the transition breakpoint prompt.
VBISRemovedBatchList	Provides access to the list of batches removed from the campaign.
VBISPhaseControl	Provides access to the phase control interface object.

VBISRecipeElements Interface Hierarchy

The following figure illustrates the VBISRecipeElements interface. This interface describes the elements of the sequential function chart (SFC).



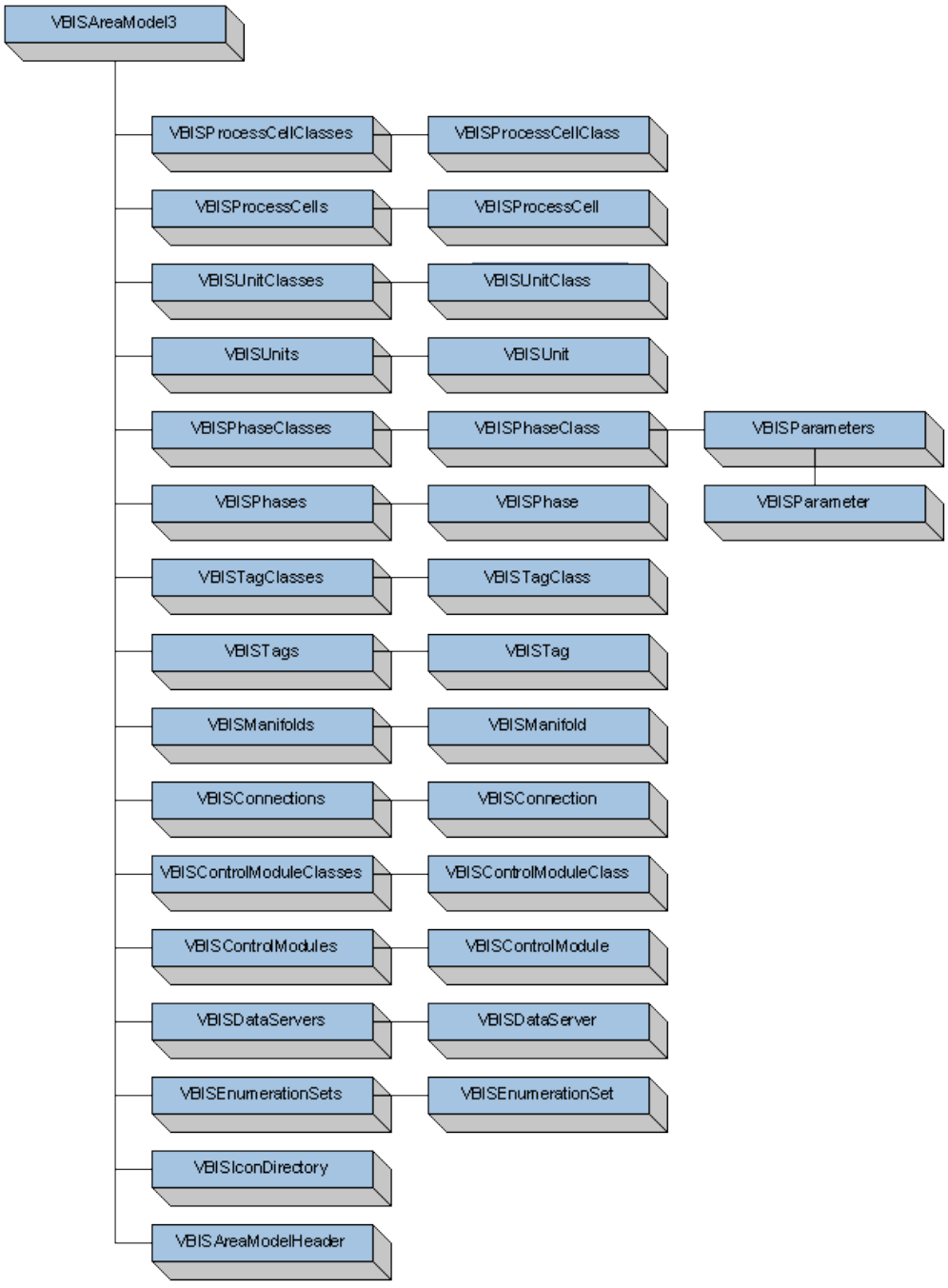
VBISRecipeElements Hierarchy

The following table describes each interface in the VBISRecipeElements hierarchy.

VBISRecipeElements Interfaces	
Interface	**Description**
VBISRecipeStepInitial	Defines the starting point of a sequential function chart.
VBISRecipeStepTerminal	Defines the ending point of a sequential function chart.
VBISRecipeStep	Defines a parent step that contains derivative steps.
VBISRecipeSteps	Defines a collection of children steps that describe the logic of the recipe.
VBISRecipeLinks	Defines a collection of links from steps to transitions, includes Jacobson links, transitions, Or/And Divergences and Convergences, and so on.
VBISRecipeStepTransitions	Defines a collection of transitions. Transitions define when a recipe moves from one step to another in the sequential function chart.

VBISAreaModel3 Interface Hierarchy

The following figure illustrates the VBISAreaModel3 interface. This interface provides access to all components of the Batch Execution area model. Along with the individual sets of equipment, the VBISAreaModel3 interface contains equipment relationships. These relationships represent the equipment hierarchy in the Batch Execution area model. For example, a particular VBISProcessCellClass object contains a list of all the VBISProcessCells it contains, and the VBISUnitClasses object contains a list of all the VBISUnits it contains, and so on.



VBISAreaModel3 Interface Hierarchy

The following table describes interfaces in the VBISAreaModel3 hierarchy.

VBISAreaModel3 Interfaces	
Interface	**Description**
VBISProcessCellClasses	A collection of VBISProcessCellClass objects defined in the area model.
VBISProcessCellClass	Provides access to process cell classes defined in the area model.
VBISProcessCells	A collection of VBISProcessCell objects.
VBISProcessCell	Provides access to process cells in the area model.
VBISUnitClasses	A collection of VBISUnitClass objects defined in the area model.
VBISUnitClass	Provides access to unit classes in the area model.
VBISUnits	A collection of VBISUnit objects.
VBISUnit	Provides access to units defined in the area model.
VBISPhaseClasses	A collection of VBISPhaseClass objects.
VBISPhaseClass	Provides access to equipment phases defined in the area model.
VBISPhases	A collection of VBISPhase objects.
VBISPhase	Provides access to equipment phases in the area model.
VBISParameters	A collection of VBISParameter objects.
VBISParameter	Provides access to equipment phase parameters defined in the area model.
VBISTagClasses	A collection of VBISTagClass objects.
VBISTagClass	Provides access to equipment phase tag classes defined in the area model.
VBISTags	A collection of VBISTag objects.

VBISAreaModel3 Interfaces	
Interface	**Description**
VBISTag	Provides access to equipment phase tags defined in the area model.
VBISManifolds	A collection of VBISManifold objects.
VBISManifold	Provides access to manifold objects defined in the area model.
VBISConnections	A collection of VBISConnection objects.
VBISConnection	Provides access to connections defined in the area model.
VBISControlModuleClasses	A collection of VBISControlModuleClass objects.
VBISControlModuleClass	Provides access to control module classes defined in the Batch Execution area model.
VBISControlModules	A collection of VBISControlModule objects.
VBISControlModule	Provides access to control modules defined in the Batch Execution area model.
VBISDataServers	A collection of VBISDataServer objects.
VBISDataServer	Provides access to the OPC data servers defined in the area model.
VBISEnumerationSets	A collection of VBISEnumerationSet objects.
VBISEnumerationSet	Provides access to enumeration sets defined in the area model.
VBISIconDirectory	Provides access to the icon (bitmap) directories in the area model.
VBISAreaModelHeader	Provides access to audit trail data collected for the area model.

Understanding Collections

A collection is a way of grouping a set of related items of an unknown quantity. Collections are used in Visual Basic to keep track of many things, such as the loaded forms in your program (the Forms collection), or all the controls on a form (the Controls collection). You can access these collections in a standard way that allows you to enumerate over each element within the collection. Collection objects in Visual Basic support the "for each" mechanism.

The VBIS automation interface implements collection objects. For example, the VBISProcessCells object is a collection of VBISProcessCell objects. You can use the VBISProcessCells object to enumerate over each process cell in the VBISProcessCell object.

If you plan on using multiple clients (ActiveX controls for VBIS applications), use the collection objects instead of the record set objects. The collection objects are designed to support multiple clients. The record set objects are no longer the recommended way to interact with VBIS.

For more information on collections, refer to the Visual Basic documentation.

Configuring the Batch Execution ActiveX Controls

Batch Execution supplies a set of ActiveX controls that you can use to extend the power of Batch Execution. ActiveX controls technology rests on a foundation of core technologies such as COM (Component Object Model), DCOM (Distributed COM), and OLE (Object Linking and Embedding) automation. The Batch Execution ActiveX controls also provide enabling technology that support compliance with 21 CFR Part 11.

You can integrate the Batch Execution ActiveX controls into custom applications. Each control provides programmability through OLE Automation to take advantage of the control's features through the Microsoft's Visual Basic and Visual C++ programming languages, or any other COM-compatible development environment.

ActiveX Control Modes

The Batch Execution ActiveX controls run as clients of VBIS. There are two modes of use for the ActiveX controls: run time and design time. Run time is when the control is active and interacting with the Batch Execution Server. At run time, there must be a connection to VBIS, either locally or remotely over a network. Design time is when a developer configures the properties of the control. The control may be embedded in a control container or be included as part of an application. A connection to the VBIS Server is not necessary during design time.

OLE Containers

You can embed the ActiveX controls in any OLE-compliant container, such as:

- Proficy iFIX WorkSpace
- Web browsers, such as Internet Explorer
- Visual Basic
- Visual C++

Property Pages

The controls provide property pages that designers or operators can use to view and change the control's properties.

The properties that you can define for the Batch Execution BatchList, BatchAdd, BatchRecipeList, BatchOperatorPromptsList, BatchBindingPromptsList, and BatchAlarmList ActiveX controls are:

- Columns
- Sort Order
- VBIS Server
- Miscellaneous
- Security
- Electronic Signature (available on most controls)
- Command Buttons (available only on specific controls)
- Recipe Filters (available for BatchList only)
- Colors
- Fonts

The following sections describe each property page.

For information on BatchManualPhase property pages, refer to the BatchManualPhase ActiveX Control section. For information on BatchSFC property pages, refer to the BatchSFC ActiveX Control section.

For information on configuring properties programmatically, refer to the following sections:

- BatchList ActiveX Control
- BatchAdd ActiveX Control
- BatchRecipeList ActiveX Control
- BatchOperatorPromptsList ActiveX Control
- BatchBindingPromptsList ActiveX Control
- BatchAlarmList ActiveX Control
- BatchCampaignClient ActiveX Control
- BatchManualPhase ActiveX Control
- BatchSFC ActiveX Control
- BatchActivePhaseList ActiveX Control

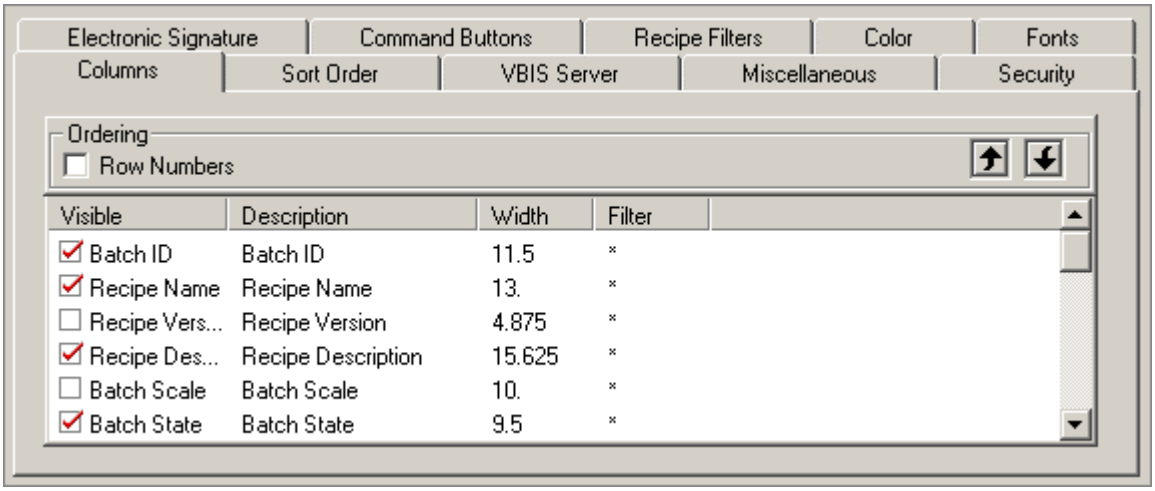
Columns Property Page

Each ActiveX control contains a Columns property page. The Columns property page lets you configure the following properties:

- Which columns of data appear in the control.
- The header text for the columns.
- The width of the columns.

- The filter setting for the columns.
- Whether or not to display row numbers.
- The order of the columns.

The Columns property page, shown in the following figure, is available at both design time and run time. The columns listed in the property page depend on which ActiveX control you are configuring. This figure shows the BatchListControl's Columns property page.



Columns Property Page

The following table lists the items that are available on the Columns property page.

NOTE: *Refer to the section for the specific control for description and syntax information on the individual control's Column properties.*

Columns Property Page		
Item	**Description**	**Associated Property**
Row Numbers check box	When selected, row numbers are displayed for the control.	RowNumbersVisible
Up and Down Arrow buttons	Lets you change the order of the columns in the control.	None
Visible check box	Lets you select which columns you want to appear in the control.	*Column*Visible
Visible (data column heading)	Lets you edit the name of the column.	None

Columns Property Page		
Item	**Description**	**Associated Property**
Description	The name of the data column.	None
Width	Lets you set the width for the column.	*Column*Width
Filter	Lets you set the filter for the column.	*ColumnFilter*
Run check box (Available on the BatchActivePhaseList control)	When selected, phases in the Run and Restarting state display in this grid.	None
Hold check box (Available on the BatchActivePhaseList control)	When selected, phases in the Holding and Held state display in this grid.	None
Stop check box (Available on the BatchActivePhaseList control)	When selected, phases in the Stopping and Stopped state display in this grid.	None
Abort check box (Available on the BatchActivePhaseList control)	When selected, phases in the Aborting and Aborted state display in this grid.	None

Setting Columns

The steps below describe how to set the column properties for an ActiveX control.

▸To set the column properties for a control:

1. Open the control in an OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Columns property page.
4. To display row numbers, select the Row Numbers check box.
5. Select the check box next to each column that you want to appear in the control.
6. To change the column name:
    a. Select the column you want to edit.

b. Click the text you want to change under Visible.

c. Enter the column header text that you want to appear on the control.

7. To change the width of the column:

    a. Click the column you want to edit.

    b. Click the width you want to edit under Width.

    c. Enter width for the column.

8. To change the filter for the column:

    a. Click the column you want to edit.

    b. Click the filter you want to edit under Filter.

    c. Enter the filter you want to apply to the column.

Filtering Data

The filter is a string that specifies what type of data is to be displayed. If the data displayed in any column does not correspond with the filter string specified, then the entire row will not be displayed. The following table describes each filter string.

Columns Property Page		
This Filter...	**Does This...**	**Example**
*	Shows all items for the column	If you enter ***** for the Batch ID column, all Batch IDs are displayed. Additionally, you can use the * filter as follows: ***Batch*** This filter will display all batches that have Batch in the Batch ID. ***Batch** This filter will display all batches that have Batch as the last five characters in its ID. **Batch*** This filter will display all batches that have Batch as the first five characters in its ID.

Columns Property Page		
This Filter...	**Does This...**	**Example**
?	Matches exactly one character.	Entering **Batch?** in the Batch ID column will display all batches with IDs that start with Batch and have one character following it. For example, it might display the following batches: Batch1 Batch2 BatchA
|	OR Logical operator allows multiple filter conditions.	You can use the | character to create several filter conditions separated by OR logical operators. Entering **A*| B* |C*** for the BatchID column displays all of the batches starting with A or B or C.
!	Allows you to search for the opposite condition.	Entering **!B*** lists all items that do not begin with the letter B. Entering **!*BAT*** lists all items except those that contain the phrase BAT.
\	Allows you to treat the ?, *, !, and \ characters as non-filtering characters.	Entering **Batch*?** in the Batch ID column might display batches with the following IDs: Batch*1 Batch*2 Batch*A If a wildcard (*, ?, or !) is the first character in a search string, use the backslash to escape the character. For example: "*B*" or "\?B*" or "\!B*" are possible expressions.
Null (no filter)	Matches nothing	If you do not enter a filter for the Batch ID column, none are displayed. ***NOTE:*** *Entering a space character as a filter works differently than entering nothing. For example, entering a space will match columns such as Start Time, that contain a space prior to starting a batch.*

Filtering Guidelines

The following are guidelines to use when filtering data in the ActiveX controls.

Case-Sensitivity

Filtering is case-sensitive. For example, if you enter a filter such as Batch?, it will display the following batches:

- BatchA
- Batch1
- Batch2

Whereas, the following batches will *not* be displayed:

- batchA
- batch1
- batch2

Empty Data

If data was not entered for a column, a match occurs only if you enter a filter of *. For example, assume you enter the following filter for the recipe description column:

```
Toothpaste*
```

In this case, the recipe with the empty description will not be displayed.

Combining Filters

You can combine filters to achieve more complex filtering capabilities. For example, assume the following filter is entered for the Batch ID column:

```
*Batch?
```

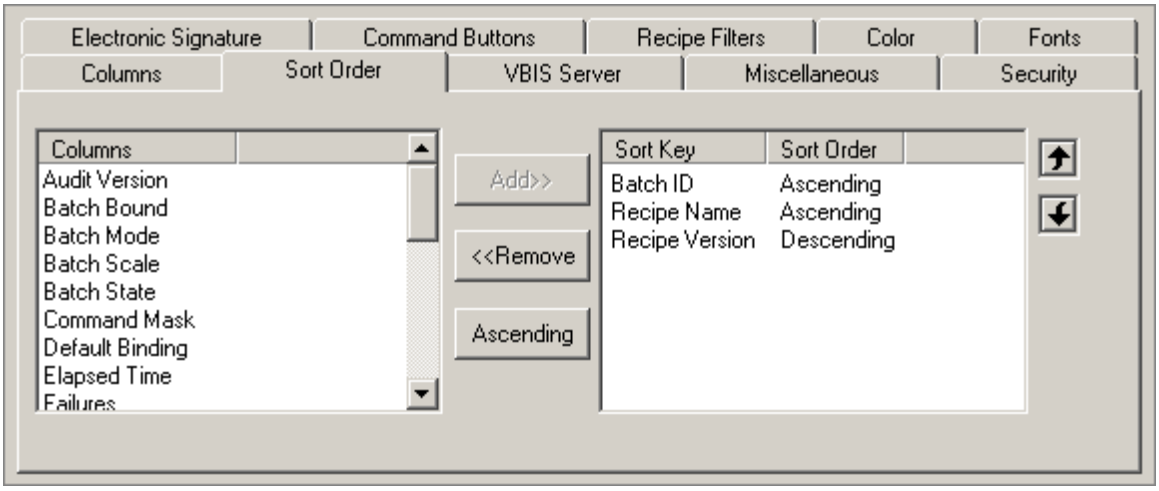
This filter might display the following batches:

- MondayBatchA
- TuesdayBatch1

Sort Order Property Page

Each ActiveX control contains a Sort Order property page. The Sort Order property page lets you configure how you want the data sorted in the control. The sort order property page is available at design time and at run time.

The columns on this page depend on which control you are configuring. The following figure shows the Sort Order property page for the BatchList control.



Sort Order Property Page

The following table lists the items that are available on the Sort Order property page.

Sort Order Property Page		
Item	**Description**	**Associated Properties**
Columns list box	Displays the columns that you can select. You can select up to three columns for sorting.	None
Add button	Lets you add up to three columns from the Columns list to the Sort Key list.	None
Remove button	Lets you delete a column from the Sort Key list.	None
Ascending/ Descending button	Lets you change the sort order for columns from ascending to descending and vice-versa.	None
Up and Down arrow buttons	Lets you prioritize the columns in the sort.	None

NOTE: *Developers can programmatically sort data using the SetSortKeys Method. Refer to the BatchList ActiveX Control section for information on this method.*

Configuring Sort Order Properties

The steps that follow explain how to configure the sort order for an ActiveX control.

To configure the sort order for a control:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Sort Order property page.
4. In the Columns list box, select the columns you want to sort by and click the Add button to add them to the sort list. You can add up to three columns to the sort list.
5. In the sort list, select a column and click the Up or Down arrows to move the column up or down in the sort priority.
6. In the sort list, select a column and click the Ascending or Descending button to sort the column in ascending or descending order.
7. Click OK to save your changes.

Sorting Guidelines

The following are guidelines for sorting data in the ActiveX controls. There can be a maximum of three columns selected for sorting.

Sort Priority

If batches have identical information in the first column of information sorted, the sort proceeds to the second sort key column of information in the list. For example, assume the following sort order is defined.

Sort Key	Sort Order
Batch ID	Ascending
Recipe Name	Ascending
Recipe Version	Ascending

In this case, if the same Batch ID is assigned to two batches, then they are sorted by Recipe Name. For example:

	Batch ID	Recipe Name	Recipe Version
1	BATCH1	MAKE_MOUTHWASH	2.0

	Batch ID	**Recipe Name**	**Recipe Version**
2	BATCH1	MAKE_TOOTHPASTE	1.0

Now assume that the batches have identical Batch IDs and Recipe Names. In this case they are sorted by Recipe Version, the third sort key.

	Batch ID	**Recipe Name**	**Recipe Version**
1	BATCH1	MAKE_TOOTHPASTE	1.0
2	BATCH1	MAKE_TOOTHPASTE	2.0

Now assume that Descending was selected for the Recipe Version sort order, as shown below:

Sort Key	**Sort Order**
Batch ID	Ascending
Recipe Name	Ascending
Recipe Version	Descending

In this case, the list is sorted as follows:

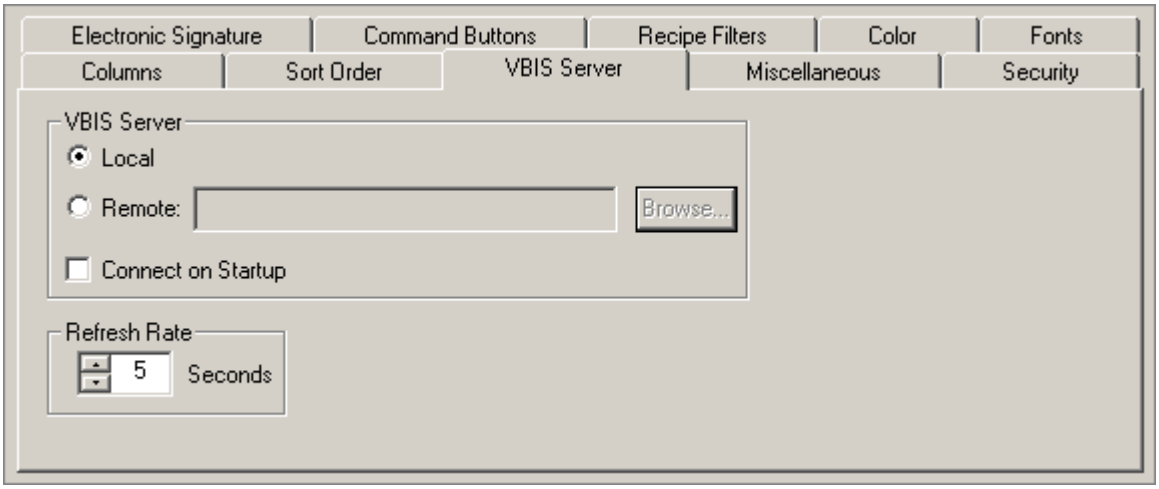
	Batch ID	**Recipe Name**	**Recipe Version**
1	BATCH1	MAKE_TOOTHPASTE	2.0
2	BATCH1	MAKE_TOOTHPASTE	1.0

VBIS Server Property Page

Each ActiveX control contains a VBIS Server property page. The VBIS Server property page:

- Lets you configure the VBIS Server settings for the ActiveX control, such as whether the VBIS Server is local or remote.
- Is available at design time and may be enabled at run time using the Security property page.

The following figure shows the VBIS Server property page. This page is the same for all controls.



VBIS Server Property Page

The following table lists the items that are available on the VBIS Server property page.

VBIS Server Property Page		
Item	**Description**	**Associated Property**
Local button	When selected, indicates that the VBIS Server is running on the same computer as the ActiveX control.	VBISServerName
Remote button	When selected, indicates that the VBIS Server is running on a different computer from the ActiveX control.	VBISServerName
Remote field	When Remote is selected, lets you specify the computer on which the VBIS Server is running.	VBISServerName
Browse button	When Remote is selected, lets you browse through the network to select the computer on which the VBIS Server is running.	None
Connect on Startup check box	When selected, the ActiveX control automatically connects to the VBIS Server when the control is instantiated in a run-time environment.	ConnectAtStartup

VBIS Server Property Page		
Item	**Description**	**Associated Property**
Refresh Rate field	Lets you specify the interval, in seconds, that data is updated in the control. Valid refresh rates range from 0 to 120 seconds. The default value for the Refresh Rate is 5 on all of the ActiveX controls, except the BatchAdd and BatchRecipeList control. For the BatchAdd and BatchRecipeList controls, the default refresh rate is 0, since the information in these lists changes less frequently. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually. ***NOTE:*** *On computers with 166 Megahertz or less, you may experience problems if you set the refresh rate to less than 3 seconds.*	RefreshRate

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Server properties.*

Configuring VBIS Server Settings

The steps that follow explain how to configure the VBIS Server settings for an ActiveX control.

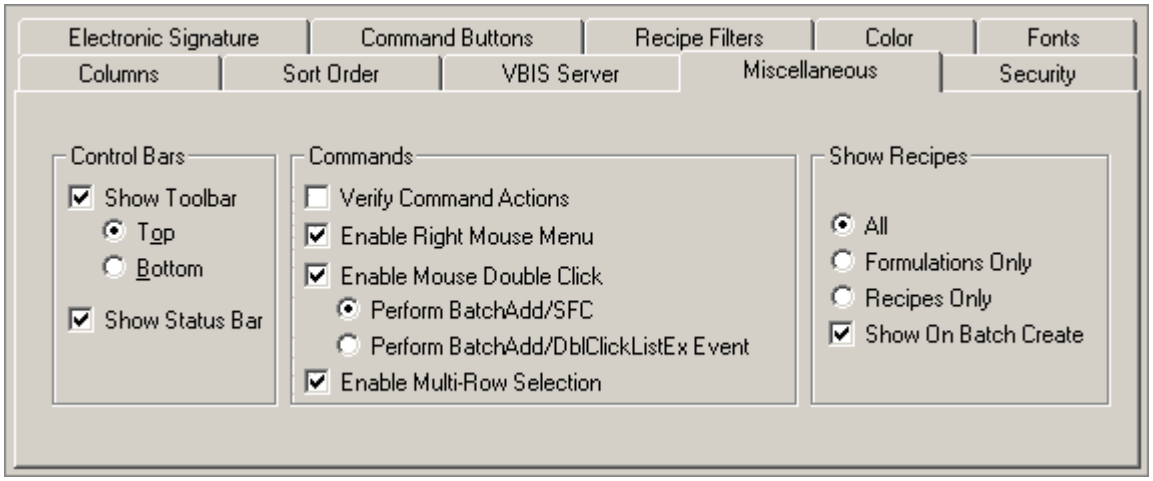
To configure the VBIS Server settings for a control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the VBIS Server property page.
4. Select Local if the VBIS Server is running on the same computer as the control.
5. Select Remote if the VBIS Server is running on a different computer from the control.
6. If you selected Remote, click the Browse button to select the computer name on which the VBIS Server is running. You can also manually enter the computer name, IP address, or other DCOM-compatible computer name.
7. Select the Connect on Startup check box if you want the control to automatically connect to the VBIS Server when the control is instantiated in a run-time environment.
8. In the Refresh Rate field, enter the interval at which you want to refresh data.
9. Click OK to save your changes.

NOTE: *You can also click the VBIS Server name in the status bar to display the property page.*

Miscellaneous Property Page

Each ActiveX control contains a Miscellaneous property page. The Miscellaneous property page lets you configure a variety of settings for the control, such as whether or not to display the toolbar. The Miscellaneous property page is available at design time and may be enabled at run time using the Security property page. The selections on this page depend on which control you are configuring. The following figure shows the Miscellaneous property page for the BatchList control.



Miscellaneous Property Page

The following table lists the items that are available on the Miscellaneous property page.

Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Show Toolbar check box	When selected, the toolbar is displayed in the control.	ToolBarEnabled
Top button	When selected, the toolbar appears at the top of the control.	ToolBarPosition
Bottom button	When selected, the toolbar appears at the bottom of the control.	ToolBarPosition
Show Status Toolbar	When selected, the status toolbar appears on the control.	StatusBarEnabled

Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Verify Command Actions check box (Available on the BatchList, BatchOperatorPromptsList, BatchBindingPromptsList, BatchManualPhase, BatchSFC (on the General tab), BatchCampaignClient and BatchActivePhaseList controls.)	When selected, operators are prompted to confirm each command before it is executed.	VerifyCommandActions
Enable Right Mouse Menu check box	When selected, the operator can right-click the running ActiveX control to view a right-click menu. If you remove the checkmark, the right mouse menu does not display when right clicking the running ActiveX control.	EnableRightMouseMenu
Enable Mouse Double-Click check box (Available on the BatchList, BatchAdd, BatchOperatorPromptsList, BatchBindingPromptsList, BatchCampaignClient and BatchActivePhaseList controls.)	When selected, operators can double click an item in the control to perform a function. For example, in the BatchAdd control, double-clicking on a recipe displays the Create Batch dialog box.	MouseDblClickedEnabled

Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Perform BatchAdd/SFC and Perform BatchAdd/DblClickListEx Event options (Available only on the BatchList ActiveX control)	If you select Perform BatchAdd/SFC, when the operator double-clicks an empty row in the BatchList ActiveX control the Batch Add dialog box appears. If a row that contains a batch is double-clicked, the SFC ActiveX control appears with information about the selected batch. Enabling the double-click actions is the default mode. If you select Perform BatchAdd/DblClickListEx Event and the operator double-clicks a row with data, the batch serial number is returned in the DblClickListEx event. For example, you can use this serial number to programmatically launch your own dialog/form/iFIX picture with the BatchSFC displaying the specified batch with that serial number. If the operator double-clicks an empty row of the BatchList ActiveX control, the Batch Add dialog box appears.	DoubleClickAction
Enable Multi-Row Selection (Available only on the BatchList ActiveX control)	If you select the Enable Multi-Row Selection check box, the operator can select multiple rows in the spreadsheet view. If you clear this check box, the operator can select only one row.	MultiRowSelectionEnabled

Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Decimal Places edit box (Available only on the BatchAdd ActiveX control) ***NOTE:*** *For information on changing the number of decimal places displayed after initial configuration, refer to Adjusting the Number of Decimal Places on a Control.*	Indicates the number of the decimal places used by the control.	DecimalValue
Show Recipes area (BatchAdd and BatchList only)	Allows a user to select a display option: show recipes, show formulations, show both.	ShowFormulationRecipeRadioButtons and SetFormulationRecipeRadioButtonValue (for BatchAdd) ShowAddBatchFormulationRadioButtons and AddBatchFormulationRadioButtonValue (for BatchList)

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Miscellaneous properties.*

Configuring Miscellaneous Properties

The steps that follow explain how to configure the miscellaneous properties for an ActiveX control.

To configure the miscellaneous properties for a control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Miscellaneous property page.
4. In the Control Bars area, if you want to display the toolbar select the Show Toolbar check box and select either top or bottom to indicate the location of the toolbar.
5. In the Control Bars area, select the Show Status Bar check box if you want to display the status bar. The status bar always appears at the bottom of the control.
6. Select the Verify Command Actions check box to prompt operators to verify commands before they are executed.
7. Select the Enable Mouse Double-Click check box to let operators double-click a row in the control to perform a function in the control.
8. Click OK to save your changes.

Adjusting the Number of Decimal Places on a Control

The DecimalValue precision property is set during the design time configuration of the BatchAdd control. The BatchAdd control is also displayed from within the BatchList control which does not allow design time configuration of the BatchAdd Control. However, you can use the following procedure to override the DecimalValue precision property setting and change the number of decimal places displayed in the BatchList control.

To change the number of decimal places displayed in a control after initial configuration:

1. Locate the autodisable.ini file. Typically on Windows Vista or Windows Server 2008, the autodisable.ini file resides in the C:\ProgramData\Proficy\Proficy Batch Execution\Configs folder. On Windows XP or Windows Server 2003, the file is located in the C:\Documents and Settings\All Users\Application Data\Proficy\Proficy Batch Execution\Configs folder.
2. In Notepad or another text editor, open the autodisable.ini file.
3. Add the following two lines to the file, but replace X with the number of decimal places you want to display on the control:

   ```
   [PRECISION_OVERRIDE]

   DecimalPlaces=X
   ```

4. Save the file.

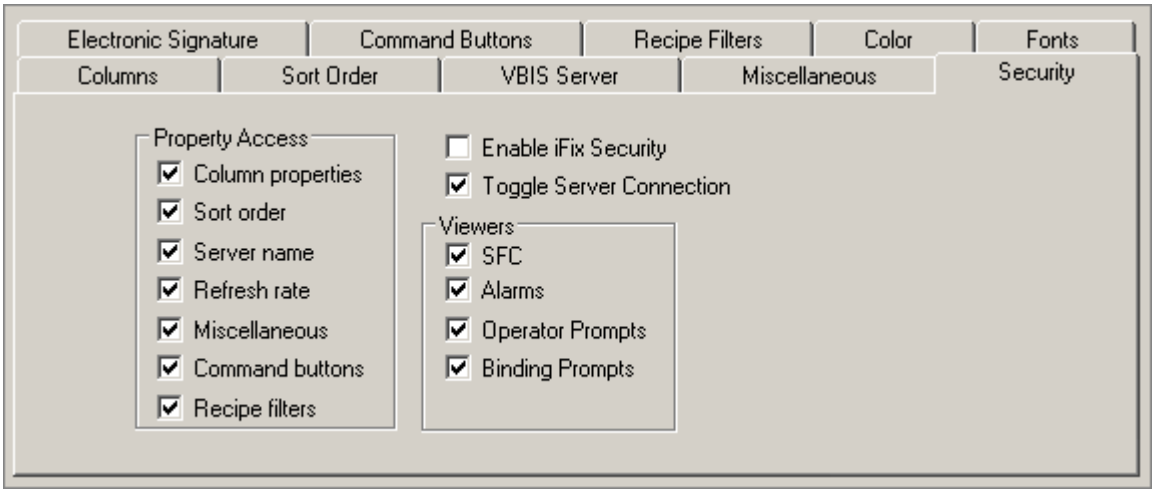
Security Property Page

Each ActiveX control contains a Security property page. The Security property page lets you configure the security settings for the control, such as which properties the operator can configure. The Security property page is available only at design time. The following containers are design-time containers.

- Visual Basic
- Visual C++
- Proficy iFIX WorkSpace

If the designer turns off the refresh rate in the security property page, it means the refresh rate will not be changeable at run time from the VBIS Server property page dialog. A programmer, however, can change the refresh rate through an automation function.

The selections on this page depend on which control you are configuring. The following figure shows the Security property page for the BatchList control.



Security Property Page

The following table lists the items that are available on the Security property page.

Security Property Page		
Item	**Description**	**Associated Property**
Column Properties check box	When selected, lets operators modify the settings on the Column property page.	ColumnEditEnabled
Sort Order check box	When selected, lets operators modify the settings on the Sort Order property page.	SortOrderEditEnabled
Server check box	When selected, lets operators modify the settings on the VBIS Server property page.	ServerEditEnabled
Refresh Rate check box	When selected, lets operators modify the refresh rate on the Miscellaneous property page. If de-selected, data is still refreshed at the set time, but the operator cannot change the refresh rate.	RefreshRateEditEnabled
Miscellaneous check box	When selected, lets operators modify the settings in the Miscellaneous property page.	MiscEditActions

Security Property Page		
Item	**Description**	**Associated Property**
Command Buttons check box (Not available on the BatchAlarmList, BatchRecipeList, and BatchAdd controls.)	When selected, lets operators modify the settings in the Command Buttons property page.	CommandBtnsEditEnabled
Recipe Filters check box (Only available from the BatchList control.)	When selected, lets operators modify the settings on the Recipe Filters property page.	RecipePageEditEnabled
Enable iFIX Security	When selected, the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. ***IMPORTANT:*** *If you enabled electronic signatures for a command on the Electronic Signature tab, then iFIX security is bypassed. Windows security is checked instead. Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	EnableIFIXSecurity
Toggle Server Connection check box	When selected, lets operators connect and disconnect from the VBIS Server by selecting the connection icon. The connection icon appears at the bottom-right corner of the control on the Status toolbar.	ToggleConnectionEnabled
Can Edit SFC Parameters check box (Only present on BatchActivePhaseList control.)	When selected, the operator can edit SFC parameters, if the user is allowed to access the SFC from the BatchActivePhaseList control.	ParamReportEditEnabled

Security Property Page		
Item	**Description**	**Associated Property**
SFC check box (Only present on BatchList, BatchCampaignClient and BatchActivePhaseList controls.)	When selected in the BatchList or BatchActivePhaseList control, the operator can click the SFC button (or, if configured on the Miscellaneous property page, double-click a row of data) to display the SFC ActiveX control with information about the selected batch.	ViewSFC
Alarms check box (Only present on BatchList, BatchCampaignClient and BatchActivePhaseList controls.)	When selected, the operator can click the View Alarms button on the BatchList or BatchActivePhaseList control to access the View Alarms dialog box.	ViewAlarms
Operator Prompts check box (Only present on BatchList, BatchCampaignClient and BatchActivePhaseList controls.)	When selected, the operator can click the Operator Prompts button on the BatchList or BatchActivePhaseList control to access the Operator Prompts dialog box.	ViewOperatorPrompts
Binding Prompts check box (Only present on BatchList, BatchCampaignClient and BatchActivePhaseList controls.)	When selected, the operator can click the Binding Prompts button on the BatchList or BatchActivePhaseList control to access the Binding Prompts dialog box.	ViewBindingPrompts

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Security properties.*

Configuring Security Properties

The steps that follow describe how to configure security properties for an ActiveX control.

To configure the security for a control:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.

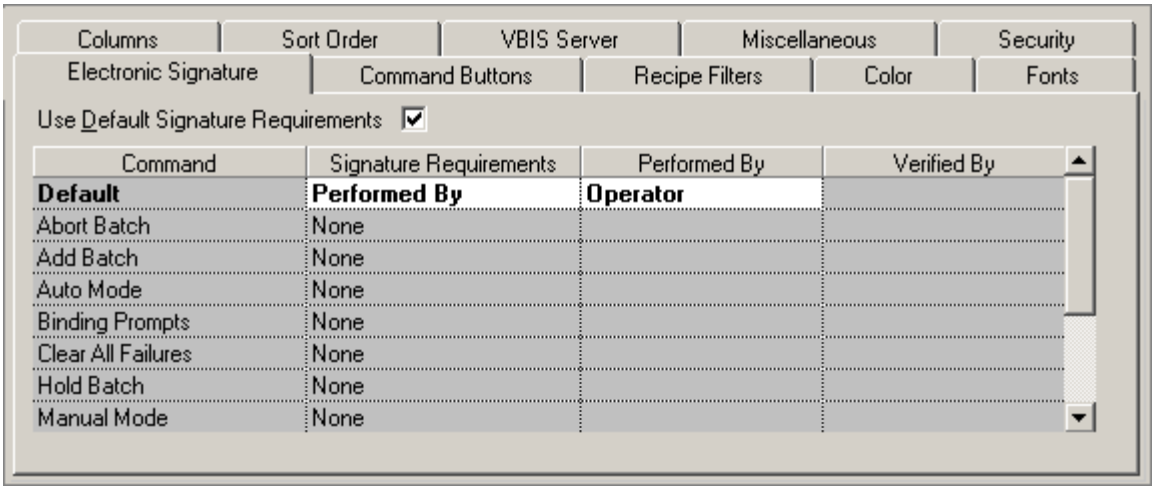
3. Select the Security property page.
4. In the Property Access area, select the properties you want operators to be able to modify at run time.
5. Select the Toggle Server Connections check box if you want operators to be able to disconnect and connect from the VBIS Server.
6. In the Viewers area, select which dialog boxes you want operators to be able to access during run time. The Viewers area appears only on the BatchList control's Security property page.
7. Click OK to save your changes.

Electronic Signature Property Page

The Electronic Signature property page specifies the electronic signature requirements associated with each of the commands available on the ActiveX control. The signature requirements define whether or not a user or users must "sign-off" on a command before it is performed. You can define the following signature types for each command:

- None (no signature required)
- Performed By (only "Performed By" signature required)
- Performed By/Verified By (both "Performed By" and "Verified By" signatures required)

The following figure shows the Electronic Signature property page for the BatchList control.



Electronic Signature Property Page

If you do not define a signature type, no signature is requested when the command is performed; NONE is the default signature type if you do not select one. If you do not want to define these signature types individually, you can specify one signature type for all commands by using the default signature row.

If you specified a Performed By or Performed By/Verified By signature type, Windows security groups are used to validate the user who performs the command. The user must be a member of the specified group to perform or verify the command. The same user *cannot* sign both the Performed By

and Verified By signature requirements, even if that user resides in both the Performed By and Verified By groups. What that means is that the user performing a command cannot be the same user who verifies that command. Likewise, the user verifying a command cannot be the same user who performed that command.

To enter the groups from which the user is authorized, you specify the Performed By and Verified By groups in the Electronic Signature property page. This property page is enabled at design time only. The number of times the user can attempt to enter a failed signature is defined by your administrator through Windows security. Refer to account policy information in your Microsoft Windows Help system for more details.

Batch Execution supports both Local and Domain groups. Upon receiving a request to validate a signature, Batch Execution checks the local computer for the specified group. If the group is not found, it then checks for a Domain level group to verify the user name and password. You can choose to configure your security groups and user names on the Batch Execution Server computer or as part of your plant's overall Domain Security Configuration.

When the ActiveX control captures an electronic signature, the signature is recorded in the BATCH_CMD_SIGNATURE_SUCCESS table, along with the computer name and time stamp of the signature, as well as other data. If a signature fails, the event is not logged, nor is the command performed. For more information on the fields in this table refer to the BATCH_CMD_SIGNATURE_SUCCESS Table section.

Using the Electronic Signatures Property Page

The Electronic Signatures Property Page is available in the following ActiveX controls:

- BatchList
- BatchAdd
- BatchOperatorPromptsList
- BatchBindingPromptsList
- BatchManualPhase
- BatchSFC

The following table lists the items that are available on the Electronic Signature property page.

Electronic Signature Property Page		
Item	**Description**	**Associated Property or Method**
Use Default Signature Requirements check box	When selected the user applies a single signature requirement for all methods on a control.	UseDefaultSignatureRequirements
Signature Requirements Grid Control	Contains the signature requirements for each command or the default.	GetSignatureRequirements SetSignatureRequirements

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Electronic Signature properties. The list of commands that appear in the grid depend on the control being configured.*

Configuring Electronic Signature Properties

The steps that follow describe how to configure the electronic signature properties for an ActiveX control.

NOTE: *You can only configure the electronic signature properties in design mode. You cannot configure these properties in run mode.*

▸To configure the electronic signature properties for a control:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Electronic Signature property page.
4. Select the Use Default Signature Requirements check box if you want to use a default signature.
5. Select the signature type for the default, or for each command listed (if default is not selected):
    - Performed By
    - Performed By / Verified By
    - None
6. Select the Windows security group for each Performed By and Verified By signature specified.

    NOTE: *Right-click on the edit box and select the Browse option to open the Windows Security Groups dialog box. Make sure that the cursor is not displaying in the text box when you right-click on it. Select a group from the Windows Security Groups dialog box and click OK. You can only browse local security groups, but you can manually enter a domain group.*

7. Click OK to save your changes.

Command Buttons Property Page

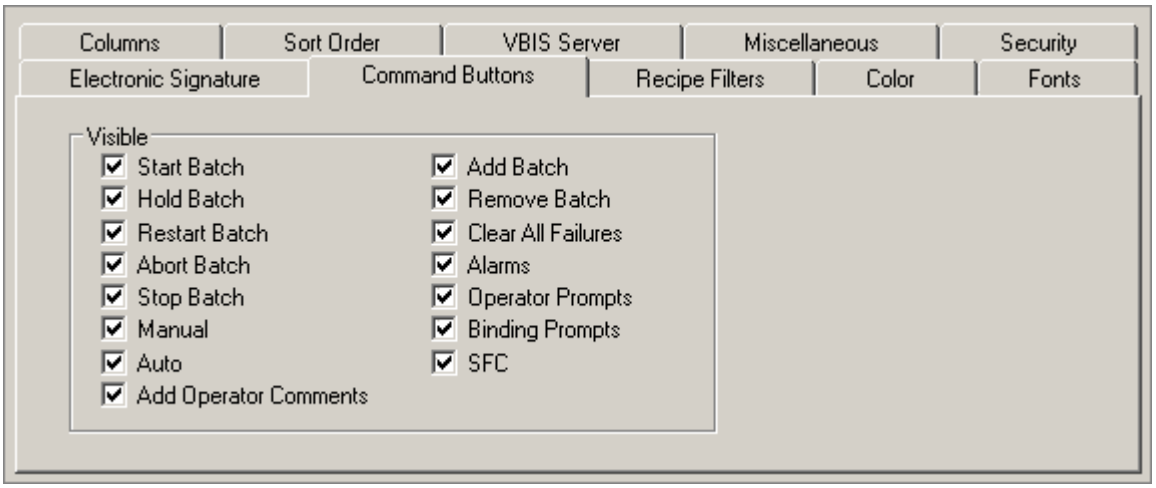
The Command Buttons (or Commands) property page is available for the following controls:

- BatchList control
- BatchOperatorPromptsList control
- BatchBindingPromptsList control
- BatchActivePhaseList control
- BatchCampaignClient control

This page lets you configure which command buttons are enabled for the control. The Command Buttons property page is available only at design time. To configure a control's command buttons, you need to edit the control in design-time mode. The following containers are design-time containers.

- Visual Basic
- Visual C++
- Proficy iFIX WorkSpace

The following figure shows the Command Buttons property page for the BatchList control.



Command Buttons Property Page

The following table lists the items that are available on the Command Buttons property page.

Command Buttons Property Page		
Item	**Description**	**Properties**
Check boxes on the BatchList control's property page: • Abort Batch • Add Batch • Alarms • Auto • Add Operator Comments • Binding Prompts • Clear All Failures • Hold Batch • Manual • Operator Prompts • Remove batch • Restart Batch • Start Batch • Stop Batch • SFC	Selecting a check box enables the associated button on the BatchList control. For example, selecting the Start Batch check box lets operators select the Start Batch button to start batches. Deselecting the check box removes the button from the toolbar.	AbortBatchButton AddBatchButton AlarmsButton AutomaticButton AddOperatorCommentsButton BindingPromptsButton ClearAllFailuresButton HoldBatchButton ManualButton OperatorPromptsButton RemoveBatchButton RestartBatchButton StartBatchButton StopBatchButton SFCButton
Check box on the BatchBindingPromptsList control's property page: • Acknowledge Prompts	When selected, the Acknowledge Prompts button is enabled.	AcknowledgePromptButton
Check box on the BatchOperatorPromptsList control's property page: • Acknowledge Prompts	When selected, the Acknowledge Prompts button is enabled.	AcknowledgePromptButton

Command Buttons Property Page		
Item	**Description**	**Properties**
Check boxes on the BatchActivePhaseList control's property page: • SFC • Alarms • Operator Prompts • Binding Prompts • Start • Hold • Abort • Add Comment • Clear All Failures • Stop • Restart • Auto • Manual	Selecting a check box in the Visible area, enables the associated button on the BatchActivePhaseList control. Clearing the check box removes the button from the toolbar. Selecting a check box in the SFC Dialog only area, enables the associated button on the SFC control, if the user is allowed to access it from the BatchActivePhaseList control. For example, selecting the Start check box lets operators select the Start Batch button to start batches.	SFCButton AlarmsButton OperatorPromptsButton BindingPromptsButton StartButton HoldButton AbortButton AddCommentButton ClearAllFailuresButton StopButton RestartButton AutoButton ManualButton

Command Buttons Property Page		
Item	**Description**	**Properties**
Check boxes on the BatchCampaignClient control's property page: Campaign • Start • Pause • Restart • Add • Remove • View • Modify • Duplicate BatchList • Start • Hold • Restart • Abort • Stop • Manual • Auto • Add Comment • Add Batch • Remove Batch • Clear All Failures • Alarms • Operator Prompts • Binding Prompts • SFC	Selecting a check box in the Visible area, enables the associated button on the BatchCampaignClient control. Clearing the check box removes the button from the toolbar.	StartCampaignButton PauseCampaignButton RestartCampaignButton AddCampaignButton RemoveCampaignButton ViewCampaignButton ModifyCampaignButton DuplicateCampaignButton AbortBatchButton AddBatchButton AlarmsButton AutomaticButton AddOperatorCommentsButton BindingPromptsButton ClearAllFailuresButton HoldBatchButton ManualButton OperatorPromptsButton RemoveBatchButton RestartBatchButton StartBatchButton StopBatchButton SFCButton

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Command Buttons properties.*

Configuring Command Buttons Properties

To configure the command buttons for the BatchList, BatchBindingPromptsList, and BatchOperatorPromptsList controls, follow the steps below.

To configure the command button properties:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Command Buttons property page.
4. In the Enabled area, select the buttons that you want enabled on the control.
5. Click OK to save your changes.

Recipe Filters Property Page

Only the BatchList ActiveX control contains a Recipe Filters property page. The Recipe Filters property page lets you set the filter attributes for the recipe list that appears when you add a batch through the BatchList control in run mode. The recipe list filters according to the filter conditions set on this property page.

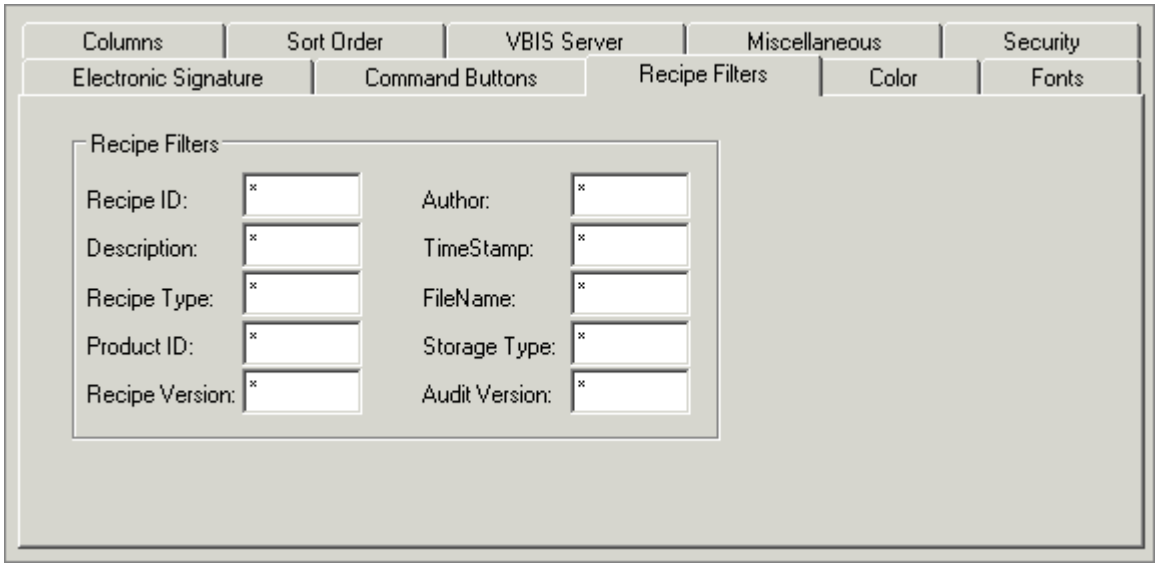
TIP: *If you want to allow an operator to change the recipe filter settings in run mode, on the Security Property Page, select the Recipe Filters check box.*

By default, all filters are enabled, and display the * symbol in the filter field (meaning that all recipes display in the list when you add a batch). However, you can define filters for this list using text and the * and ? symbols for wildcard matches. The * symbol can be used in expressions to search for multiple characters in a word or phrase. The ? symbol searches for a single character in a word. You can use the ! symbol to negate the condition. For example, if you use the filter condition !M* for the Recipe ID, then all recipes that do NOT begin with the letter M will display.

You can also use the | symbol to specify multiple OR conditions. For example, the expression CIP*|BATCH* returns all strings that begin with CIP or BATCH.

The \ escape character followed by the \, *, |, or ? symbol allows you to search with the actual `\, *, |, or ? symbol. Using the \! as the first part of your condition allows you to search for the actual ! symbol at the beginning of the condition. Be aware that pattern matching with recipe filters is case sensitive.

The Recipe Filters property page, shown in the following figure, can be available at both design time and run time.



Recipe Filters Property Page

The following table lists the items that are available on the Recipe Filters property page.

Recipe Filters Property Page		
Item	**Description**	**Associated Property**
Recipe ID edit box	Lists filter criteria for the recipe ID.	RecipeListRecipeIDFilter
Description edit box	Lists filter criteria for the recipe description.	RecipeListRecipeDescriptionFilter
Recipe Type edit box	Lists filter criteria for recipe type.	RecipeListRecipeTypeFilter
Product ID edit box	Lists filter criteria for product ID.	RecipeListProductIDFilter
Recipe Version edit box	Lists filter criteria for recipe version.	RecipeListRecipeVersionFilter

Recipe Filters Property Page		
Item	**Description**	**Associated Property**
Author edit box	Lists filter criteria for recipe author.	RecipeListAuthorFilter
TimeStamp edit box	Lists filter criteria for recipe timestamp.	RecipeListTimeStampFilter
FileName edit box	Lists filter criteria for recipe file name.	RecipeListRecipeFilenameFilter
Storage Type edit box	Lists filter criteria for recipe storage type.	RecipeListRecipeStorageTypeFilter
Audit Version edit box	Lists filter criteria for recipe audit version.	RecipeListRecipeAuditVersionFilter

Setting Recipe Filters

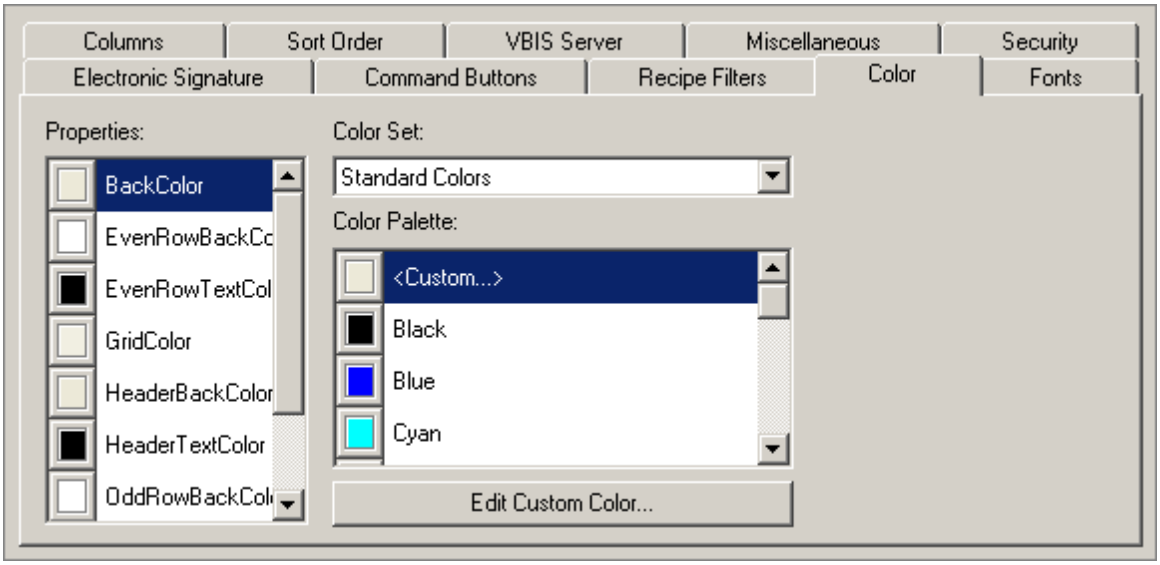
The following steps describe how to set the Recipe Filters for an ActiveX control.

To set the recipe filters for the BatchList control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Recipe Filters property page.
4. In the corresponding text box, enter the filter criteria for which you want to filter the recipe list. (This filtered recipe list appears when you add a batch in the BatchList ActiveX control.)

Colors Property Page

Each ActiveX control contains a Colors property page. The Colors property page lets you set the control's colors, such as the color of the grid lines and the color of column header text. You can choose a color from the displayed color chart or you can use the System Color drop-down list box to assign a pre-defined a system color to a property. The Colors property page, shown in the following figure, is available at design time and at run time.



Colors Property Page

The following table lists the items that are available on the Colors property page.

Colors Property Page		
Item	**Description**	**Corresponding Properties**
Properties list box	Lists all the available properties for which you can set the color.	BackColor EvenRowBackColor EvenRowTextColor GridColor HeaderBackColor HeaderTextColor OddRowBackColor OddRowTextColor
Color Set drop-down list	List the colors sets that you can choose from. For example: Windows Standard Colors, Windows System Colors.	None
Color Palette list box	Displays the available colors that you can assign to a property. Double-click the <Custom> option to display the Color dialog box and add a custom color to the Color Palette list.	None

Colors Property Page		
Item	**Description**	**Corresponding Properties**
Edit Custom Color button	Click this button to display the Color dialog box and add a custom color to the Color Palette list.	None

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Color properties.*

Setting Colors

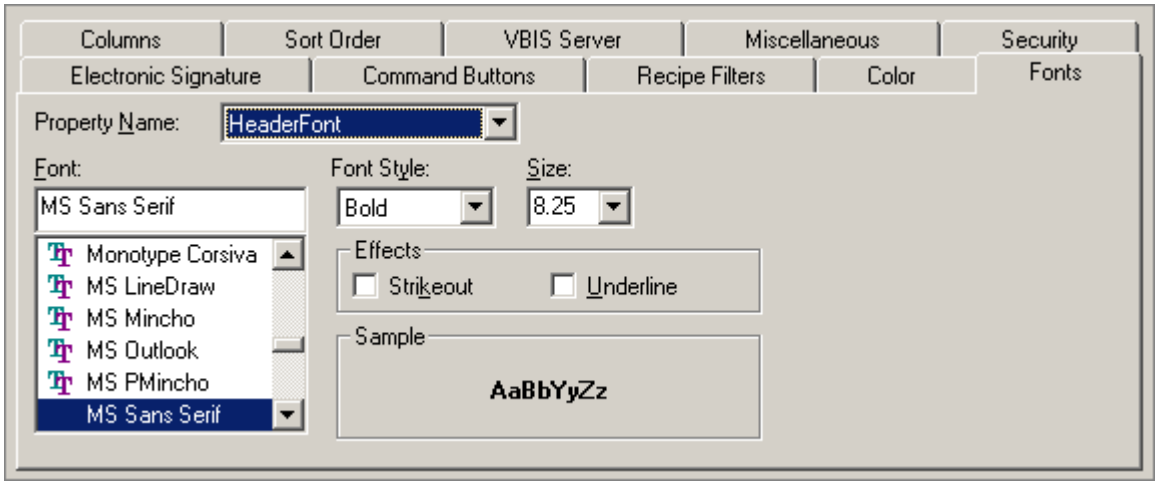
The steps that follow describe how to configure the colors used by the ActiveX control.

To set the colors for a control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Colors property page.
4. Select a property from the Property Name drop-down list box.
5. Click a color in the color palette or select a pre-defined system color from the System Color drop-down list box.

Fonts Property Page

Each ActiveX control contains a Fonts property page. The Fonts property page lets you set the font attributes for the text in the column headers and the data list. The Fonts property page, shown in the following figure, is available at design time and run time.



Fonts Property Page

The following table lists the items that are available on the Fonts property page.

Fonts Property Page		
Item	**Description**	**Associated Property**
Property Name drop-down list box	Lists all the properties for which you can set the font.	HeaderFont TextFont
Font drop-down list box	Lists the available font types that you can assign to the selected property.	None
Font Style drop-down list box	Lists the available font styles that you can assign to the selected property.	None
Font Size drop-down list box	Lists the available font sizes that you can assign to the selected property. ***NOTE:*** *Even though you select a whole number for the font size, a decimal number may be displayed, depending on the font selected, as depicted in the Size box in the preceding figure.*	None
Strikeout check box	When selected, applies strike out formatting to the selected property.	None
Underline check box	When selected, applies underline formatting to the selected property.	None

NOTE: *Refer to the specific section for each control for description and syntax information on the individual control's Font properties.*

Setting Fonts

The steps that follow describe how to set the fonts for an ActiveX control.

To set the fonts for a control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Fonts property page.
4. Select a property from the Property Name drop-down list box.
5. Select the font style, size, and font effects that you want to apply to the selected property.

Control Shortcut Keys

The following accelerator shortcut keys can be used for each control.

Control Shortcut Keys	
Use this key combination...	**To...**
Ctrl + P	Display the control's property pages.
Ctrl + R	Refresh the data within a control (used only at run time).
Ctrl + S	Display the VBIS Server property page.
Ctrl + T	Toggle the display of the toolbar.

Configuring the Tab key for the ActiveX Controls

If you want an ActiveX control in iFIX to respond to the Tab and Enter keys, there is a property (for each object that you draw in the WorkSpace) called "IsSelectable" which is False by default. To set this property, right-click on the ActiveX control in design mode, and then select the Property Window option to display the iFIX property window for the ActiveX control. Toggle the value of "IsSelectable" from False to True.

This can also be done with a VBA script when the picture loads at run time. An example is as follows:

```
Private Sub CFixPicture_Initialize()
EWIX1.IsSelectable = True
End Sub
```

These steps only apply to tabbing on the first level screen of the control. Pop-up dialog boxes do not require these steps.

Running a Control from a Web Page

Batch Execution supplies sample HTML (HyperText Markup Language) pages that show the HTML code that you can use to run Batch Execution ActiveX controls on a web page. The text that follows is the HTML code in the BatchList.htm file that is located in the Batch Execution Samples directory, typically C:\Program Files\Proficy\Proficy Batch Execution\samples\controls\HTML. Additional samples of HTML code reside in this directory, too. You can also use the Index.htm or Menu.htm to view a menu of all the samples.

NOTE: *To obtain all of the graphics for the HTML samples, run the setup.bat file located in the HTML folder. Select Run from the Start menu, browse to the setup.bat file, and click OK to run.*

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<base target="_top">
<title>Toothpaste Factory - BatchList</title>
</head>

<body background="Images/BackgroundLogo.gif" bgproperties="fixed">

<p align="center"><font face="Impact" size="6" color="#FF0000">Batch Add:</font></p>

<p align="center">
<object classid="clsid:F2B8E1C5-9D6C-11D1-80B0-006008A627D2" width="600" height="199">
<param name="_Version" value="65536">
<param name="_ExtentX" value="15875">
<param name="_ExtentY" value="5265">
<param name="_StockProps" value="1">
<param name="BackColor" value="8421440">
<param name="RefreshRate" value="0">
<param name="RecipeIDVisible" value="1">
<param name="RecipeIDHeaderText" value="Recipe ID">
<param name="RecipeIDWidth" value="23.875">
<param name="RecipeIDFilter" value="*">
<param name="RecipeDescriptionVisible" value="1">
<param name="RecipeDescriptionHeaderText" value="Recipe Description">
<param name="RecipeDescriptionWidth" value="34.625">
<param name="RecipeDescriptionFilter" value="*">
<param name="RecipeTypeVisible" value="1">
<param name="RecipeTypeHeaderText" value="Recipe Type">
<param name="RecipeTypeWidth" value="10">
<param name="RecipeTypeFilter" value="*">
<param name="ProductIDVisible" value="1">
<param name="ProductIDHeaderText" value="Product ID">
<param name="ProductIDWidth" value="10">
<param name="ProductIDFilter" value="*">
<param name="RecipeAuditVersionVisible" value="0">
<param name="RecipeAuditVersionHeaderText" value="Audit Version">
<param name="RecipeAuditVersionWidth" value="10">
<param name="RecipeAuditVersionFilter" value="*">
```

```
<param name="AuthorVisible" value="0">
<param name="AuthorHeaderText" value="Author">
<param name="AuthorWidth" value="15">
<param name="AuthorFilter" value="*">
<param name="RecipeVersionVisible" value="1">
<param name="RecipeVersionHeaderText" value="Ver">
<param name="RecipeVersionWidth" value="4.875">
<param name="RecipeVersionFilter" value="*">
<param name="TimeStampVisible" value="1">
<param name="TimeStampHeaderText" value="Time Stamp">
<param name="TimeStampWidth" value="20">
<param name="TimeStampFilter" value="*">
<param name="RecipeFilenameVisible" value="1">
<param name="RecipeFilenameHeaderText" value="Recipe File Name">
<param name="RecipeFilenameWidth" value="14">
<param name="RecipeFilenameFilter" value="*">
<param name="RecipeStorageTypeVisible" value="0">
<param name="RecipeStorageTypeHeaderText" value="Recipe Storage Type">
<param name="RecipeStorageTypeWidth" value="6">
<param name="RecipeStorageTypeFilter" value="*">
<param name="SortKey1" value="1">
<param name="SortKey2" value="2">
<param name="SortKey3" value="3">
<param name="SortKey1Order" value="1">
<param name="SortKey2Order" value="1">
<param name="SortKey3Order" value="2">
<param name="VBISServerName" value>
<param name="ToolBarEnabled" value="0">
<param name="ToolBarPosition" value="0">
<param name="MouseDblClickedEnabled" value="1">
<param name="StatusBarEnabled" value="0">
<param name="RowNumbersVisible" value="0">
<param name="OddTextColor" value="0">
<param name="EvenTextColor" value="0">
<param name="OddListBackColor" value="16777215">
<param name="EvenListBackColor" value="12632256">
<param name="HeaderTextColor" value="16777215">
<param name="HeaderBackColor" value="8421440">
<param name="GridColor" value="8421440">
<param name="ColumnOrderPPG" value="0;1;3;4;5;6;2;7;8;">
<param name="ColumnEditEnabled" value="1">
<param name="ServerEditEnabled" value="1">
<param name="RefreshRateEditEnabled" value="1">
<param name="SortOrderEditEnabled" value="1">
<param name="MiscEditEnabled" value="1">
<param name="ToggleConnectionEnabled" value="0">
<param name="ConnectAtStartup" value="1"><!-- set ServerEditEnabled to 0 if you do not
want the user to edit the server settings -->
<!-- set VBISServerName if you want to set the server name to a remote VBIS server(change
the "ComputerName" to the actual PC name) -->
<!-- <param name="VBISServerName" value="ComputerName"> -->
</object>
</p>

<p align="center"><font face="Impact" size="6" color="#FF0000">Batch List:</font></p>

<p align="center">
<object classid="clsid:1DD65396-9BD9-11D1-82BD-006008B04755" width="700" height="211">
```

```
<param name="_Version" value="65536">
<param name="_ExtentX" value="18521">
<param name="_ExtentY" value="5574">
<param name="_StockProps" value="1">
<param name="BackColor" value="8421440">
<param name="StatusBarEnabled" value="0">
<param name="RowNumbersVisible" value="1">
<param name="EvenListBackColor" value="12632256">
<param name="HeaderTextColor" value="16777215">
<param name="HeaderBackColor" value="8421440">
<param name="GridColor" value="8421440">
<!-- set ServerEditEnabled to 0 if you do not want the user to edit the server settings -
->
<param name="ServerEditEnabled" value="1">
<!-- set VBISServerName if you want to set the server name to a remote VBIS server(change
the "ComputerName" to the actual PC name) -->
<!-- <param name="VBISServerName" value="ComputerName"> -->
<param name="ToggleConnectionEnabled" value="0">
<param name="CommandBtnsEditEnabled" value="0">
<param name="AddBatchButton" value="0">
<param name="ViewAlarms" value="0">
<param name="ViewOperatorPrompts" value="0">
<param name="ViewBindingPrompts" value="0">
<param name="ConnectAtStartup" value="1">
</object>
</p>
</body>
</html>
```

Configuring IE to Run the Batch Execution ActiveX Controls

If you are running the ActiveX controls from Internet Explorer, the browser must run in a separate process and security must be configured for the ActiveX controls.

To set up the browser to run the ActiveX controls:

1. Start Internet Explorer. Do not connect to the web server.
2. Select Internet Options from the Tools menu.
3. Select the Advanced tab.
4. Select the Launch Browser Window in a New Process check box in the Browsing list, if it appears in the list.
5. Select the Security tab.
6. Select the Local Intranet icon.
7. Click the Sites button. The Local Intranet dialog box appears.
8. Click the Advanced button.
9. Enter the computer's path the Add This Web Site to the Zone field. For example, the path might be similar to this: \\computername\share.

   NOTE: *Make sure that the path is a shared folder.*

10. Click OK.

11. Click OK again to close the Local Intranet dialog box.
12. Click the Custom Level button on the Internet Options dialog box. The Security Settings dialog box appears.
13. Confirm that the following options are defined as detailed in the following table.

Setting	**Option**
Download signed ActiveX controls	Prompt
Download unsigned ActiveX controls	Disable
Initialize and script ActiveX controls not marked as safe	Enable
Run ActiveX controls and plug-ins	Enable
Script ActiveX controls marked safe for scripting	Enable

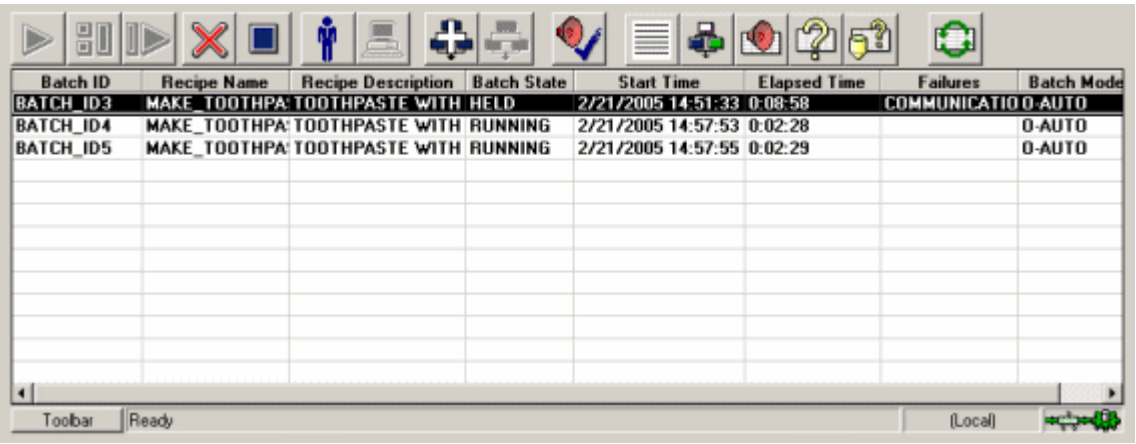
14. Make sure the security level for the zone used is *not* set to High. If it is set to High, reset it. The ActiveX controls will not load if it is set to High.
15. Click OK.
16. Click OK from the Internet Options dialog box.

BatchList ActiveX Control

The "Intellution BatchList Control" provides similar functionality to the Batch Execution Client's Batch List screen. Depending on which commands are enabled, operators can perform batch list functions such as adding a batch, starting a batch, and stopping a batch.

Designers can configure the BatchList control's GUI run-time appearance and functionality using the control's property pages. For example, the designer can disable command buttons that the operator should not have access to. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.

The following figure shows the BatchList control.



BatchList ActiveX Control

Developers can access the control programmatically through Visual Basic or Visual C++ using the control's properties and methods. The properties, methods, and events for the BatchList control are described in the following sections.

BatchList Control Properties

The following sections describe each property for the BatchList control. The properties are grouped by the following functions:

- Column properties
- VBIS Server properties
- Miscellaneous properties
- Security properties
- Electronic Signature properties
- Command Buttons properties
- Recipe Filters properties
- Color properties
- Font properties

BatchList Control Column Properties

The following table lists the properties that control the display of the columns in the BatchList control.

BatchList Control Column Properties	
Property	**Syntax**
RecipeAuditVersionFilter Sets the filter for the Audit Version column.	**C++ Syntax:** **CString** CBatchList::GetAuditVersionFilter(); **void** CBatchList::SetAuditVersionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionFilter[= *text$*]
RecipeAuditVersionHeaderText Specifies the column header text for the Audit Version column.	**C++ Syntax:** **CString** CBatchList::GetAuditVersionHeaderText(); **void** CBatchList::SetAuditVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionHeaderText[= *text$*]
RecipeAuditVersionVisible Sets whether or not to display the Audit Version column.	**C++ Syntax:** **BOOL** CBatchList::GetAuditVersionVisible (); **void** CBatchList::SetAuditVersionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionVisible [= *boolvalue*]
RecipeAuditVersionWidth Sets the width of the Audit Version column.	**C++ Syntax:** **double** CBatchList::GetAuditVersionWidth(); **void** CBatchList::SetAuditVersionWidth(double value); **Visual Basic Syntax:** [form.]*Control.*AuditVersionWidth[= *value!*]
BatchBoundFilter Sets the filter for the Batch Bound column.	**C++ Syntax:** **CString** CBatchList::GetBatchBoundFilter(); **void** CBatchList::SetBatchBoundFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchBoundFilter[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
BatchBoundHeaderText Specifies the column header text for the Batch Bound column.	**C++ Syntax:** **CString** CBatchList::GetBatchBoundHeaderText(); **void** CBatchList::SetBatchBoundHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchBoundHeaderText[= *text$*]
BatchBoundVisible Sets whether or not to display the Batch Bound column.	**C++ Syntax:** **BOOL** CBatchList::GetBatchBoundVisible (); **void** CBatchList::SetBatchBoundVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchBoundVisible [= *boolvalue*]
BatchBoundWidth Sets the width of the Batch Bound column.	**C++ Syntax:** **double** CBatchList::GetBatchBoundWidth(); **void** CBatchList::SetBatchBoundWidth(double value); **Visual Basic Syntax:** [form.]Control.BatchBoundWidth[= *value!*]
BatchIDFilter Sets the filter for the Batch ID column.	**C++ Syntax:** **CString** CBatchList::GetBatchIDFilter (); **void** CBatchList::SetBatchIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDFilter [= *text$*]
BatchIDHeaderText Specifies the column header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchList::GetBatchIDHeaderText (); **void** CBatchList::SetBatchIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDHeaderText [= *text$*]
BatchIDVisible Sets whether or not to display the Batch ID column.	**C++ Syntax:** **BOOL** CBatchList::GetBatchIDVisible (); **void** CBatchList::SetBatchIDVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDVisible [= *boolvalue*]

BatchList Control Column Properties	
Property	**Syntax**
BatchIDWidth Sets the width of the Batch ID column.	**C++ Syntax:** **double** CBatchList::GetBatchIDWidth(); **void** CBatchList::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDWidth[= *value!*]
CommandMaskFilter Sets the filter for the Command Mask column.	**C++ Syntax:** **CString** CBatchList::GetCommandMaskFilter(); **void** CBatchList::SetCommandMaskFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.CommandMaskFilter[= *text$*]
CommandMaskHeaderText Specifies the column header text for the Command Mask column.	**C++ Syntax:** **CString** CBatchList::GetCommandMaskHeaderText(); **void** CBatchList::SetCommandMaskHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.CommandMaskHeaderText[= *text$*]
CommandMaskVisible Sets whether or not to display the Command Mask column.	**C++ Syntax:** **BOOL** CBatchList::GetCommandMaskVisible (); **void** CBatchList::SetCommandMaskVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.CommandMaskVisible [= *boolvalue*]
CommandMaskWidth Sets the width of the Command Mask column.	**C++ Syntax:** **double** CBatchList::GetCommandMaskWidth(); **void** CBatchList::SetCommandMaskWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.CommandMaskWidth[= *value!*]

BatchList Control Column Properties	
Property	**Syntax**
DefaultBindingFilter Sets the filter for the Default Binding column.	**C++ Syntax:** **CString** CBatchList::GetDefaultBindingFilter(); **void** CBatchList::SetDefaultBindingFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultBindingFilter[= *text$*]
DefaultBindingHeaderText Specifies the column header text for the Default Binding column.	**C++ Syntax:** **CString** CBatchList::GetDefaultBindingHeaderText(); **void** CBatchList::SetDefaultBindingHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultBindingHeaderText[= *text$*]
DefaultBindingVisible Sets whether or not to display the Default Binding column.	**C++ Syntax:** **BOOL** CBatchList::GetDefaultBindingVisible (); **void** CBatchList::SetDefaultBindingVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultBindingVisible [= *boolvalue*]
DefaultBindingWidth Sets the width of the Default Binding column.	**C++ Syntax:** **double** CBatchList::GetDefaultBindingWidth(); **void** CBatchList::SetDefaultBindingWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultBindingWidth[= *value!*]
DescriptionFilter Sets the filter for the Recipe Description column.	**C++ Syntax:** **CString** CBatchList::GetDescriptionFilter(); **void** CBatchList::SetDescriptionFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionFilter[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
DescriptionHeaderText Specifies the column header text for the Recipe Description column.	**C++ Syntax:** **CString** CBatchList::GetDescriptionHeaderText(); **void** CBatchList::SetDescriptionHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionHeaderText[= *text$*]
DescriptionVisible Sets whether or not to display the Recipe Description column.	**C++ Syntax:** **BOOL** CBatchList::GetDescriptionVisible (); **void** CBatchList::SetDescriptionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionVisible [= *boolvalue*]
DescriptionWidth Sets the width of the Recipe Description column.	**C++ Syntax:** **double** CBatchList::GetDescriptionWidth(); **void** CBatchList::SetDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionWidth[= *value!*]
ElapsedTimeFilter Sets the filter for the Elapsed Time column.	**C++ Syntax:** **CString** CBatchList::GetElapsedTimeFilter(); **void** CBatchList::SetElapsedTimeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ElapsedTimeFilter[= *text$*]
ElapsedTimeHeaderText Specifies the column header text for the Elapsed Time column.	**C++ Syntax:** **CString** CBatchList::GetElapsedTimeHeaderText(); **void** CBatchList::SetElapsedTimeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ElapsedTimeHeaderText[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
ElapsedTimeVisible Sets whether or not to display the Elapsed Time column.	**C++ Syntax:** **BOOL** CBatchList::GetElapsedTimeVisible (); **void** CBatchList::SetElapsedTimeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ElapsedTimeVisible [= *boolvalue*]
ElapsedTimeWidth Sets the width of the Elapsed Time column.	**C++ Syntax:** **double** CBatchList::GetElapsedTimeWidth(); **void** CBatchList::SetElapsedTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ElapsedTimeWidth[= *value!*]
FailureFilter Sets the filter for the Failure column.	**C++ Syntax:** **CString** CBatchList::GetFailureFilter(); **void** CBatchList::SetFailureFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureFilter[= *text$*]
FailureHeaderText Specifies the column header text for the Failure column.	**C++ Syntax:** **CString** CBatchList::GetFailureHeaderText(); **void** CBatchList::SetFailureHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureHeaderText[= *text$*]
FailureVisible Sets whether or not to display the Failure column.	**C++ Syntax:** **BOOL** CBatchList::GetFailureVisible (); **void** CBatchList::SetFailureVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureVisible [= *boolvalue*]
FailureWidth Sets the width of the Failure column.	**C++ Syntax:** **double** CBatchList::GetFailureWidth(); **void** CBatchList::SetFailureWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureWidth[= *value!*]

BatchList Control Column Properties	
Property	**Syntax**
InternalIDFilter Sets the filter for the Internal Batch ID column.	**C++ Syntax:** **CString** CBatchList::GetInternalIDFilter(); **void** CBatchList::SetInternalIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*InternalIDFilter[= *text$*]
InternalIDHeaderText Specifies the column header text for the Internal Batch ID column.	**C++ Syntax:** **CString** CBatchList::GetInternalIDHeaderText(); **void** CBatchList::SetInternalIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*InternalIDHeaderText[= *text$*]
InternalIDVisible Sets whether or not to display the Internal Batch ID column.	**C++ Syntax:** **BOOL** CBatchList::GetInternalIDVisible (); **void** CBatchList::SetInternalIDVisible (**BOOL** value); **Visual Basic Syntax:** VB: [*form.*]*Control.*UnitVisible [= *boolvalue*]
InternalIDWidth Sets the width of the Internal Batch ID column.	**C++ Syntax:** **double** CBatchList::GetInternalIDWidth(); **void** CBatchList::SetInternalIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*InternalIDWidth[= *value!*]
ModeFilter Sets the filter for the Batch Mode column.	**C++ Syntax:** **CString** CBatchList::GetModeFilter(); **void** CBatchList::SetModeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeFilter[= *text$*]
ModeHeaderText Specifies the column header text for the Batch Mode column.	**C++ Syntax:** **CString** CBatchList::GetModeHeaderText(); **void** CBatchList::SetModeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeHeaderText[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
ModeVisible Sets whether or not to display the Batch Mode column.	**C++ Syntax:** **BOOL** CBatchList::GetModeVisible (); **void** CBatchList::SetModeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeVisible [= *boolvalue*]
ModeWidth Sets the width of the Batch Mode column.	**C++ Syntax:** **double** CBatchList::GetModeWidth(); **void** CBatchList::SetModeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeWidth[= *value!*]
OpBindParametersFilter Sets the filter for the Operator Bind Parameters column.	**C++ Syntax:** **CString** CBatchList::GetOpBindParametersFilter(); **void** CBatchList::SetOpBindParametersFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OpBindParametersFilter[= *text$*]
OpBindParametersHeaderText Specifies the column header text for the Operator Bind parameters column.	**C++ Syntax:** **CString** CBatchList::GetOpBindParametersHeaderText(); **void** CBatchList::SetOpBindParametersHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OpBindParametersHeaderText[= *text$*]
OpBindParametersVisible Sets whether or not to display the Operator Bind Parameters column.	**C++ Syntax:** **BOOL** CBatchList::GetOpBindParametersVisible (); **void** CBatchList::SetOpBindParametersVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OpBindParametersVisible [= *boolvalue*]

BatchList Control Column Properties	
Property	**Syntax**
OpBindParametersWidth Sets the width of the Operator Bind Parameters column.	**C++ Syntax:** **double** CBatchList::GetOpBindParametersWidth(); **void** CBatchList::SetOpBindParametersWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.OpBindParametersWidth[= *value!*]
OpBindUnitsFilter Sets the filter for the Operator Bind Units column.	**C++ Syntax:** **CString** CBatchList::GetOpBindUnitsFilter(); **void** CBatchList::SetOpBindUnitsFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.OpBindUnitsFilter[= *text$*]
OpBindUnitsHeaderText Specifies the column header text for the Operator Bind Units column.	**C++ Syntax:** **CString** CBatchList::GetOpBindUnitsHeaderText(); **void** CBatchList::SetOpBindUnitsHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.OpBindUnitsHeaderText[= *text$*]
OpBindUnitsVisible Sets whether or not to display the Operator Bind Units column.	**C++ Syntax:** **BOOL** CBatchList::GetOpBindUnitsVisible (); **void** CBatchList::SetOpBindUnitsVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.OpBindUnitsVisible [= *boolvalue*]
OpBindUnitsWidth Sets the width of the Operator Bind Units column.	**C++ Syntax:** **double** CBatchList::GetOpBindUnitsWidth(); **void** CBatchList::SetOpBindUnitsWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.OpBindUnitsWidth[= *value!*]

BatchList Control Column Properties	
Property	**Syntax**
OpInteractionFilter Sets the filter for the Operator Interaction column.	**C++ Syntax:** **CString** CBatchList::GetOpInteractionFilter(); **void** CBatchList::SetOpInteractionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.OpInteractionFilter[= *text$*]
OpInteractionHeaderText Specifies the column header text for the Operator Interaction column.	**C++ Syntax:** **CString** CBatchList::GetOpInteractionHeaderText(); **void** CBatchList::SetOpInteractionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.OpInteractionHeaderText[= *text$*]
OpInteractionVisible Sets whether or not to display the Operator Interaction column.	**C++ Syntax:** **BOOL** CBatchList::GetOpInteractionVisible (); **void** CBatchList::SetOpInteractionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.OpInteractionVisible [= *boolvalue*]
OpInteractionWidth Sets the width of the Operator Interaction column.	**C++ Syntax:** **double** CBatchList::GetOpInteractionWidth(); **void** CBatchList::SetOpInteractionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.OpInteractionWidth[= *value!*]
ParametersRequiredFilter Sets the filter for the Parameters Required column.	**C++ Syntax:** **CString** CBatchList::GetParametersRequiredFilter(); **void** CBatchList::SetParametersRequiredFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ParametersRequiredFilter[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
ParametersRequiredHeaderText Specifies the column header text for the Parameters Required column.	**C++ Syntax:** **CString** CBatchList::GetParametersRequiredHeaderText(); **void** CBatchList::SetParametersRequiredHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersRequiredHeaderText[= *text$*]
ParametersRequiredVisible Sets whether or not to display the Parameters Required column.	**C++ Syntax:** **BOOL** CBatchList::Get ParametersRequiredVisible (); **void** CBatchList::Set ParametersRequiredVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersRequiredVisible [= *boolvalue*]
ParametersRequiredWidth Sets the width of the Parameters Required column.	**C++ Syntax:** **double** CBatchList::GetParametersRequiredWidth(); **void** CBatchList::SetParametersRequiredWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersRequiredWidth[= *value!*]
ParametersSupportedFilter Sets the filter for the Parameters Supported column.	**C++ Syntax:** **CString** CBatchList::GetParametersSupportedFilter(); **void** CBatchList::SetParametersSupportedFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersSupportedFilter[= *text$*]
ParametersSupportedHeaderText Specifies the column header text for the Parameters Supported column.	**C++ Syntax:** **CString** CBatchList::GetParametersSupportedHeaderText(); **void** CBatchList::SetParametersSupportedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersSupportedHeaderText[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
ParametersSupportedVisible Sets whether or not to display the Parameters Supported column.	**C++ Syntax:** **BOOL** CBatchList::GetParametersSupportedVisible (); **void** CBatchList::SetParametersSupportedVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersSupportedVisible [= *boolvalue*]
ParametersSupportedWidth Sets the width of the Parameters Supported column.	**C++ Syntax:** **double** CBatchList::GetParametersSupportedWidth(); **void** CBatchList::SetParametersSupportedWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ParametersSupportedWidth[= *value!*]
PhaseFilter Sets the filter for the Phase column.	**C++ Syntax:** **CString** CBatchList::GetPhaseFilter(); **void** CBatchList::SetPhaseFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseFilter[= *text$*]
PhaseHeaderText Specifies the column header text for the Phase column.	**C++ Syntax:** **CString** CBatchList::GetPhaseHeaderText(); **void** CBatchList::SetPhaseHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseHeaderText[= *text$*]
PhaseVisible Sets whether or not to display the Phase column.	**C++ Syntax:** **BOOL** CBatchList::GetPhaseVisible (); **void** CBatchList::SetPhaseVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseVisible [= *boolvalue*]
PhaseWidth Sets the width of the Phase column.	**C++ Syntax:** **double** CBatchList::GetPhaseWidth(); **void** CBatchList::SetPhaseWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseWidth[= *value!*]

BatchList Control Column Properties	
Property	**Syntax**
ProcessCellFilter Sets the filter for the Process Cell column.	**C++ Syntax:** **CString** CBatchList::GetProcessCellFilter(); **void** CBatchList::SetProcessCellFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcessCellFilter[= *text$*]
ProcessCellHeaderText Specifies the header text for Process Cell column.	**C++ Syntax:** **CString** CBatchList::GetProcessCellHeaderText(); **void** CBatchList::SetProcessCellHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcessCellHeaderText[= *text$*]
ProcessCellVisible Sets whether or not to display the Process Cell column.	**C++ Syntax:** **BOOL** CBatchList::GetProcessCellVisible (); **void** CBatchList::SetProcessCellVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcessCellVisible [= *boolvalue*]
ProcessCellWidth Sets the width of the Process Cell column.	**C++ Syntax:** **double** CBatchList::GetProcessCellWidth(); **void** CBatchList::SetProcessCellWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcessCellWidth[= *value!*]
RecipeFilter Sets the filter for the Recipe Name column.	**C++ Syntax:** **CString** CBatchList::GetRecipeFilter (); **void** CBatchList::SetRecipeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilter [= *text$*]
RecipeHeaderText Specifies the column header text for the Recipe Name column.	**C++ Syntax:** **CString** CBatchList::GetRecipeHeaderText (); **void** CBatchList::SetRecipeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeHeaderText [= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
RecipeVerFilter Sets the filter for the Recipe Version column.	**C++ Syntax:** **CString** CBatchList::GetRecipeVerFilter (); **void** CBatchList::SetRecipeVerFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVerFilter [= *text$*]
RecipeVerHeaderText Specifies the column header text for the Recipe Version column.	**C++ Syntax:** **CString** CBatchList::GetRecipeVerHeaderText (); **void** CBatchList::SetRecipeVerHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVerHeaderText[= *text$*]
RecipeVerVisible Sets whether or not to display the Recipe Version column.	**C++ Syntax:** **BOOL** CBatchList::GetRecipeVerVisible (); **void** CBatchList::SetRecipeVerVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVerVisible [= *boolvalue*]
RecipeVerWidth Sets the width of the Recipe Version column.	**C++ Syntax:** **double** CBatchList::GetRecipeVerWidth(); **void** CBatchList::SetRecipeVerWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVerWidth[= *value!*]
RecipeVisible Sets whether or not to display the Recipe Name column.	**C++ Syntax** **BOOL** CBatchList::GetRecipeVisible (); **void** CBatchList::SetRecipeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVisible [= *boolvalue*]
RecipeWidth Sets the width of the Recipe Name column.	**C++ Syntax:** **double** CBatchList::GetRecipeWidth(); **void** CBatchList::SetRecipeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeWidth[= *value!*]

BatchList Control Column Properties	
Property	**Syntax**
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchList::GetRowNumbersVisible(); **void** CBatchList::SetRowNumbersVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RowNumbersVisible[= *boolvalue*]
ScaleFilter Sets the filter for the Batch Scale column.	**C++ Syntax:** **CString** CBatchList::GetScaleFilter(); **void** CBatchList::SetScaleFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StartTimeFilter[= *text$*]
ScaleHeaderText Specifies the column header text for the Batch Scale column.	**C++ Syntax:** **CString** CBatchList::GetScaleHeaderText(); **void** CBatchList::SetScaleHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ScaleHeaderText[= *text$*]
ScaleVisible Sets whether or not to display the Batch Scale column.	**C++ Syntax:** **BOOL** CBatchList::GetScaleVisible (); **void** CBatchList::SetScaleVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ScaleVisible [= *boolvalue*]
ScaleWidth Sets the width of the Batch Scale column.	**C++ Syntax:** **double** CBatchList::GetScaleWidth(); **void** CBatchList::SetScaleWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ScaleWidth[= *value!*]
StartTimeFilter Sets the filter for the Start Time column.	**C++ Syntax:** **CString** CBatchList::GetStartTimeFilter(); **void** CBatchList::SetStartTimeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StartTimeFilter[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
StartTimeHeaderText Specifies the column header text for the Start Time column.	**C++ Syntax:** **CString** CBatchList::GetStartTimeHeaderText(); **void** CBatchList::SetStartTimeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StartTimeHeaderText[= *text$*]
StartTimeVisible Sets whether or not to display the Start Time column.	**C++ Syntax:** **BOOL** CBatchList::GetStartTimeVisible (); **void** CBatchList::SetStartTimeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StartTimeVisible [= *boolvalue*]
StartTimeWidth Sets the width of the Start Time column.	**C++ Syntax:** **double** CBatchList::GetStartTimeWidth(); **void** CBatchList::SetStartTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StartTimeWidth[= *value!*]
StateFilter Sets the filter for the State column.	**C++ Syntax:** **CString** CBatchList::GetStateFilter(); **void** CBatchList::SetStateFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StateFilter[= *text$*]
StateHeaderText Specifies the header text for the State column.	**C++ Syntax:** **CString** CBatchList::GetStateHeaderText(); **void** CBatchList::SetStateHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StateHeaderText[= *text$*]
StateVisible Sets whether or not to display the State column.	**C++ Syntax:** **BOOL** CBatchList::GetStateVisible (); **void** CBatchList::SetStateVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StateVisible [= *boolvalue*]

BatchList Control Column Properties	
Property	**Syntax**
StateWidth Sets the width of the State column.	**C++ Syntax:** **double** CBatchList::GetStateWidth(); **void** CBatchList::SetStateWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StateWidth[= *value!*]
TypeFilter Sets the filter for the Type column.	**C++ Syntax:** **CString** CBatchList::GetTypeFilter(); **void** CBatchList::SetTypeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TypeFilter[= *text$*]
TypeHeaderText Specifies the header text for the Type column.	**C++ Syntax:** **CString** CBatchList::GetTypeHeaderText(); **void** CBatchList::SetTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TypeHeaderText[= *text$*]
TypeVisible Sets whether or not to display the Type column.	**C++ Syntax:** **BOOL** CBatchList::GetTypeVisible (); **void** CBatchList::SetTypeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*TypeVisible [= *boolvalue*]
TypeWidth Sets the width of the Type column.	**C++ Syntax:** **double** CBatchList::GetTypeWidth(); **void** CBatchList::SetTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*TypeWidth[= *value!*]
UnitFilter Sets the filter for the Unit column.	**C++ Syntax:** **CString** CBatchList::GetUnitFilter(); **void** CBatchList::SetUnitFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitFilter[= *text$*]

BatchList Control Column Properties	
Property	**Syntax**
UnitVisible Sets whether or not to display the Unit column.	**C++ Syntax:** **BOOL** CBatchList::GetUnitVisible (); **void** CBatchList::SetUnitVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitVisible [= *boolvalue*]
UnitWidth Sets the width of the Unit column.	**C++ Syntax:** **double** CBatchList::GetUnitWidth(); **void** CBatchList::SetUnitWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitWidth[= *value!*]
UnitHeaderText Specifies the header text for the Unit column.	**C++ Syntax:** **CString** CBatchList::GetUnitHeaderText(); **void** CBatchList::SetUnitHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitHeaderText[= *text$*]
UnitsRequiredFilter Sets the filter for the Units Required column.	**C++ Syntax:** **CString** CBatchList::GetUnitsRequiredFilter(); **void** CBatchList::SetUnitsRequiredFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsRequiredFilter[= *text$*]
UnitsRequiredHeaderText Specifies the column header text for the Units Required column.	**C++ Syntax:** **CString** CBatchList::GetUnitsRequiredHeaderText(); **void** CBatchList::SetUnitsRequiredHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsRequiredHeaderText[= *text$*]
UnitsRequiredVisible Sets whether or not to display the Units Required column.	**C++ Syntax:** **BOOL** CBatchList::GetUnitsRequiredVisible (); **void** CBatchList::SetUnitsRequiredVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsRequiredVisible [= *boolvalue*]

BatchList Control Column Properties	
Property	**Syntax**
UnitsRequiredWidth Sets the width of the Units Required column.	**C++ Syntax:** **double** CBatchList::GetUnitsRequiredWidth(); **void** CBatchList::SetUnitsRequiredWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsRequiredWidth[= *value!*]
UnitsSupportedFilter Sets the filter for the Units Supported column.	**C++ Syntax:** **CString** CBatchList::GetUnitsSupportedFilter(); **void** CBatchList::SetUnitsSupportedFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsSupportedFilter[= *text$*]
UnitsSupportedHeaderText Specifies the column header text for the Units Supported column.	**C++ Syntax:** **CString** CBatchList::GetUnitsSupportedHeaderText(); **void** CBatchList::SetUnitsSupportedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsSupportedHeaderText[= *text$*]
UnitsSupportedVisible Sets whether or not to display the Units Supported column.	**C++ Syntax:** **BOOL** CBatchList::GetUnitsSupportedVisible (); **void** CBatchList::SetUnitsSupportedVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitsSupportedVisible [= *boolvalue*]

UnitsSupportedWidth
Sets the width of the Units Supported column.
	C++ Syntax:
	double CBatchList::GetUnitsSupportedWidth();
	void CBatchList::SetUnitsSupportedWidth(double value);
	Visual Basic Syntax:
	[*form.*]*Control.*UnitsSupportedWidth[= *value!*]

BatchList Control VBIS Server Properties

The following table lists the properties that control the VBIS Server settings for the BatchList control.

BatchList Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup If TRUE, the control will connect when instantiated.	**C++ Syntax:** **BOOL** CBatchList::GetConnectAtStartup(); **void** CBatchList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchList ActiveX control is 5. ***NOTE:*** *For the BatchAdd and BatchRecipeList controls, the default Refresh Rate is 0, since the information in these lists changes less frequently. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually.*	**C++ Syntax:** **short** CBatchList::GetRefreshRate(); **void** CBatchList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RefreshRate[= *value%*]
VBISServerName The name of the VBIS Server to connect to. If the string is empty, the local VBIS Server is used.	**C++ Syntax:** **CString** CBatchList::GetVBISServerName(); **void** CBatchList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.VBISServerName[= *text$*]

BatchList Control Miscellaneous Properties

The following table lists the properties that control the miscellaneous settings for the BatchList control.

BatchList Control Miscellaneous Properties	
Property	**Syntax**
DoubleClickAction Sets the double-click action: • A value of 0 (default) indicates that when the operator double-clicks an empty row in the BatchList ActiveX control, the Batch Add dialog box appears. When double-clicking a row that contains a batch, the SFC control appears with information about the selected batch. ***NOTE:*** *Be aware that the ViewSFC property on the Security tab also must be enabled if you want the SFC ActiveX control to display when double-clicked. For more information, refer to the BatchList Control Miscellaneous Properties section.* • A value of 1 indicates that when the operator double-clicks a row with data, the batch serial number is returned in the DblClickListEx event. You can use this number, for instance, to programmatically launch the BatchSFC displaying the specified batch with that serial number. If the operator double-clicks an empty row of the BatchList ActiveX control, the Batch Add dialog box appears.	**C++ Syntax:** **short** CBatchList::GetDoubleClickAction(); **void** CBatchList::SetDoubleClickAction(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.DoubleClickAction[= *value%*]
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchList::GetEnableRightMouseMenu(); **void** CBatchList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]

BatchList Control Miscellaneous Properties	
Property	**Syntax**
MouseDblClickedEnabled Sets whether or not to enable operators double-click access to commands. When enabled, double-clicking displays the Create Batch dialog box.	**C++ Syntax:** **BOOL** CBatchList::GetMouseDblClickedEnabled(); **void** CBatchList::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MouseDblClickedEnabled[= *boolvalue*]
MultiRowSelectionEnabled Sets whether or not an operator can select more than one row in a spreadsheet. When set to True, the default, multiple rows can be selected. When set to False, an operator can only select one row.	**C++ Syntax:** **BOOL** CBatchList::GetMultiRowSelectionEnabled(); **void** CBatchList::SetMultiRowSelectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MultiRowSelectionEnabled[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchList::GetStatusBarEnabled(); **void** CBatchList::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StatusBarEnabled[= *boolvalue*]
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchList::GetToolBarEnabled(); **void** CBatchList::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarEnabled[= *boolvalue*]
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchList::GetToolBarPosition(); **void** CBatchList::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarPosition[= *value%*]
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when the operator executes a command.	**C++ Syntax:** **BOOL** CBatchList::GetVerifyCommandActions(); **void** CBatchList::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VerifyCommandActions[= *boolvalue*]

BatchList Control Miscellaneous Properties	
Property	**Syntax**
ShowAddBatchFormulationRadioButtons Sets whether the display options (to show master recipes, formulations, or both) appear on the BatchList control when you create a batch. 0 does not display the option buttons. 1 (default) displays the option buttons.	**C++ Syntax:** **BOOL** CBatchList::GetShowAddBatchFormulationRadioButtons(); **void** CBatchList::SetShowAddBatchFormulationRadioButtons(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ShowAddBatchFormulationRadioButtons[= *boolvalue*]
AddBatchFormulationRadioButtonValue If the ShowAddBatchFormulationRadioButtons property is enabled, recipe options appear on the BatchList control when you create a batch: 0 displays all master recipes and formulations in the list. 1 displays only formulations in the list. 2 displays only master recipes in the list.	**C++ Syntax:** **short** CBatchList::GetAddBatchFormulationRadioButtonValue(); **void** CBatchList::SetAddBatchFormulationRadioButtonValue(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AddBatchFormulationRadioButtonValue[= *value%*]

BatchList Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchList control.

BatchList Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements If this property is True, it indicates that the default signature settings will be used for all commands for the ActiveX control. If this property is False, then the signature setting of each command is used. The default value of UseDefaultSignatureRequirements is True. The default signature requirements are None. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchList::GetUseDefaultSignatureRequirements();**void** CBatchList::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.UseDefaultSignatureRequirements[= *boolvalue*]

BatchList Control Security Properties

The following table lists the properties that control the security settings for elements on the BatchList control.

BatchList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchList::GetColumnEditEnabled(); **void** CBatchList::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ColumnEditEnabled[= booleanvalue]

BatchList Control Security Properties	
Property	**Syntax**
CommandBtnsEditEnabled If TRUE, the operator can edit the command buttons that are visible.	**C++ Syntax:** **BOOL** CBatchList::Get CommandBtnsEditEnabled(); **void** CBatchList::Set CommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CommandBtnsEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchList::Get EnableIFIXSecurity (); **void** CBatchList::Set EnableIFIXSecurity (**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue*]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchList::GetMiscEditEnabled(); **void** CBatchList::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchList::GetRefreshRateEditEnabled(); **void** CBatchList::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= booleanvalue]

BatchList Control Security Properties	
Property	**Syntax**
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchList::GetServerEditEnabled(); **void** CBatchList::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchList::GetSortOrderEditEnabled(); **void** CBatchList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*SortOrderEditEnabled[= booleanvalue]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchList::GetToggleConnectionEnabled(); **void** CBatchList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= booleanvalue]
ViewSFC Sets whether or not the operator can access the SFC ActiveX control by double-clicking a batch in the list. The SFC control displays information about the selected batch. ***NOTE:*** *To view the SFC, you must also allow for double-clicking in the BatchList control by enabling the DoubleClickAction property on the Miscellaneous tab. For more information, refer to the BatchList Control Miscellaneous Properties section.*	**C++ Syntax:** **BOOL** CBatchList::GetViewSFC(); **void** CBatchList::SetViewSFC(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewSFC[= *boolvalue*]

BatchList Control Security Properties	
Property	**Syntax**
ViewAlarms Sets whether or not the operator can access the View Alarms dialog box by clicking the View Alarms button.	**C++ Syntax:** **BOOL** CBatchList::GetViewAlarms(); **void** CBatchList::SetViewAlarms(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewAlarms[= *boolvalue*]
ViewOperatorPrompts Sets whether or not the operator can access the Operator Prompts dialog box by clicking on the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchList::GetViewOperatorPrompts(); **void** CBatchList::SetViewOperatorPrompts(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewOperatorPrompts[= *boolvalue*]
ViewBindingPrompts Sets whether or not the operator can access the Binding Prompts dialog by clicking the Binding Prompts button.	**C++ Syntax:** **BOOL** CBatchList::GetViewBindingPrompts(); **void** CBatchList::SetViewBindingPrompts(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewBindingPrompts[= *boolvalue*]
RecipePageEditEnabled Sets whether or not the operator can access the options on the Recipe Filters tab.	**C++ Syntax:** **BOOL** CBatchList::GetRecipePageEditEnabled(); **void** CBatchList::SetRecipePageEditEnabled(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipePageEditEnabled[= *boolvalue*]

BatchList Control Command Buttons Properties

The following table lists the properties that control the display of command buttons in the BatchList control.

BatchList Control Command Buttons Properties	
Property	**Syntax**
AbortBatchButton Sets whether or not to display the Abort Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetAbortBatchButton (); **void** CBatchList::SetAbortBatchButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AbortBatchButton[= *boolvalue*]
AddBatchButton Sets whether or not to display the Add Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetAddBatchButton(); **void** CBatchList::SetAddBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddBatchButton[= *boolvalue*]
AddOperatorCommentsButton Sets whether or not to display the Add Comment button.	**C++ Syntax:** **BOOL** CBatchList::GetAddOperatorCommentsButton(); **void** CBatchList::SetAddOperatorCommentsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddOperatorCommentsButton[= *boolvalue*]
AlarmsButton Sets whether or not to display the Alarms button.	**C++ Syntax:** **BOOL** CBatchList::GetAlarmsButton(); **void** CBatchList::SetAlarmsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AlarmsButton[= *boolvalue*]
AutomaticButton Sets whether or not to display the Automatic Mode button.	**C++ Syntax:** **BOOL** CBatchList::GetAutomaticButton(); **void** CBatchList::SetAutomaticButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AutomaticButton[= *boolvalue*]

BatchList Control Command Buttons Properties	
Property	**Syntax**
BindingPromptsButton Sets whether or not to display the Binding Prompts button.	**C++ Syntax:** **BOOL** CBatchList::GetBindingPromptsButton(); **void** CBatchList::SetBindingPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BindingPromptsButton[= *boolvalue*]
ClearAllFailuresButton Sets whether or not to display the Clear All Failures button.	**C++ Syntax:** **BOOL** CBatchList::GetClearAllFailuresButton(); **void** CBatchList::SetClearAllFailuresButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ClearAllFailuresButton[= *boolvalue*]
HoldBatchButton Sets whether or not to display the Hold Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetHoldBatchButton(); **void** CBatchList::SetHoldBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*HoldBatchButton[= *boolvalue*]
ManualButton Sets whether or not to display the Manual Mode button.	**C++ Syntax:** **BOOL** CBatchList::GetManualButton(); **void** CBatchList::SetManualButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ManualButton[= *boolvalue*]
OperatorPromptsButton Sets whether or not to display the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchList::GetOperatorPromptsButton(); **void** CBatchList::SetOperatorPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OperatorPromptsButton[= *boolvalue*]
RemoveBatchButton Sets whether or not to display the Remove Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetRemoveBatchButton(); **void** CBatchList::SetRemoveBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RemoveBatchButton[= *boolvalue*]

BatchList Control Command Buttons Properties	
Property	**Syntax**
RestartBatchButton Sets whether or not to display the Restart Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetRestartBatchButton(); **void** CBatchList::SetRestartBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RestartBatchButton[= *boolvalue*]
SFCButton Sets whether or not to display the SFC button.	**C++ Syntax:** **BOOL** CBatchList::GetSFCButton(); **void** CBatchList::SetSFCButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*SFCButton[= *boolvalue*]
StartBatchButton Sets whether or not to display the Start Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetStartBatchButton(); **void** CBatchList::SetStartBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StartBatchButton[= *boolvalue*]
StopBatchButton Sets whether or not to display the Stop Batch button.	**C++ Syntax:** **BOOL** CBatchList::GetStopBatchButton(); **void** CBatchList::SetStopBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StopBatchButton[= *boolvalue*]

BatchList Control Recipe Filters Properties

The following table lists the properties that control the Recipe Filter settings for the BatchList control.

BatchList Control Recipe Filter Properties	
Property	**Syntax**
RecipeListRecipeIDFilter The filter specified for the recipe ID.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeIDFilter(); **void** CBatchList::SetRecipeListRecipeIDFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeIDFilter[= *text$*]
RecipeListRecipeDescriptionFilter The filter specified for the recipe description.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeDescriptionFilter(); **void** CBatchList::SetRecipeListRecipeDescriptionFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeDescriptionFilter[= *text$*]
RecipeListRecipeTypeFilter The filter specified for the recipe type.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeTypeFilter(); **void** CBatchList::SetRecipeListRecipeTypeFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeTypeFilter[= *text$*]
RecipeListProductIDFilter The filter specified for the product ID.	**C++ Syntax:** **CString** CBatchList::GetRecipeListProductIDFilter(); **void** CBatchList::SetRecipeListProductIDFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListProductIDFilter[= *text$*]

BatchList Control Recipe Filter Properties	
Property	**Syntax**
RecipeListRecipeVersionFilter The filter specified for the recipe version.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeVersionFilter(); **void** CBatchList::SetRecipeListRecipeVersionFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeVersionFilter[= *text$*]
RecipeListAuthorFilter The filter specified for the recipe author.	**C++ Syntax:** **CString** CBatchList::GetRecipeListAuthorFilter(); **void** CBatchList::SetRecipeListAuthorFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListAuthorFilter[= *text$*]
RecipeListTimeStampFilter The filter specified for the recipe timestamp.	**C++ Syntax:** **CString** CBatchList::GetRecipeListTimeStampFilter(); **void** CBatchList::SetRecipeListTimeStampFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListTimeStampFilter[= *text$*]
RecipeListRecipeFilenameFilter The filter specified for the recipe file name.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeFilenameFilter(); **void** CBatchList::SetRecipeListRecipeFilenameFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeFilenameFilter[= *text$*]
RecipeListRecipeStorageTypeFilter The filter specified for the recipe storage type.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeStorageTypeFilter(); **void** CBatchList::SetRecipeListRecipeStorageTypeFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RecipeListRecipeStorageTypeFilter[= *text$*]

BatchList Control Recipe Filter Properties	
Property	**Syntax**
RecipeListRecipeAuditVersionFilter The filter specified for the recipe audit version.	**C++ Syntax:** **CString** CBatchList::GetRecipeListRecipeAuditVersionFilter(); **void** CBatchList::SetRecipeListRecipeAuditVersionFilter(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RecipeListRecipeAuditVersionFilter[= *text$*]

BatchList Control Color Properties

The following table lists the properties that control the colors in the BatchList control.

BatchList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetBackColor(); **void** CBatchList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color%*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetEvenRowBackColor(); **void** CBatchList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]
EvenRowTextColor Sets the color of text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetEvenRowTextColor(); **void** CBatchList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowTextColor[= *color%*]

BatchList Control Color Properties	
Property	**Syntax**
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetGridColor(); **void** CBatchList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetHeaderBackColor(); **void** CBatchList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderBackColor[= *color%*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetHeaderTextColor(); **void** CBatchList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetOddRowBackColor(); **void** CBatchList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowBackColor[= *color%*]
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchList::GetOddRowTextColor(); **void** CBatchList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

Color Property Examples

The following show examples for setting color properties.

C++ Example

```
OLE_COLOR gridColor = pBatchList->GetGridColor();
OLE_COLOR newGridColor = 0x0; // black
pBatchList->SetGridColor(newGridColor);
```

Visual Basic Example

```
Static BL_OriginalGridColor As Long
     BL_OriginalGridColor = BatchList1.GridColor
     ' get the current color
BatchList1.GridColor = &H808080     ' set to dark gray
```

BatchList Control Font Properties

The following table lists the properties that control the fonts in the BatchList control.

BatchList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchList::GetHeaderFont(); **void** CBatchList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderFont[= *stdfontvariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchList::GetTextFont(); **void** CBatchList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control*.TextFont[= *stdfontvariable*]

Font Property Examples

The following show examples for setting font properties.

C++ Example

```
// this is inserted into your project when you insert the BatchList control
#include "font.h"

// get the header font info
```

```
COleFont headerFont = pBatchList->GetHeaderFont();
CString name = headerFont.GetName();
CY size = headerFont.GetSize();
BOOL bold = headerFont.GetBold();
BOOL italic = headerFont.GetItalic();
BOOL underline = headerFont.GetUnderline();
BOOL strikeThrough = headerFont.GetStrikethrough();

// set the header font info to a new font
// Note: I am re-using the COleFont object from above. This is the easiest way to
// create, initialize, and use a COleFont object.

headerFont.SetName("Arial");
size.Lo = 120000; // 12 point (multiply 12 by 10,000)
size.Hi=0;
headerFont.SetSize(size);
headerFont.SetBold(FALSE);
headerFont.SetItalic(TRUE);
headerFont.SetUnderline(TRUE);
headerFont.SetStrikethrough(FALSE);

pBatchList->SetHeaderFont(headerFont);
```

Visual Basic Example

```
Dim NewHeaderFont As New StdFont
Dim OriginalHeaderFont As New StdFont
With NewHeaderFont
    .Name = "Arial"
    .Size = 12
    .Bold = True
    .Italic = True
    .Underline = False
End With

    ' get the current font
    OriginalHeaderFont = BatchList1.HeaderFont

    ' set to the new font
    BatchList1.HeaderFont = NewHeaderFont
```

BatchList Control Methods

The BatchList control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- SelectRowByID Method
- SelectRowByNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method

- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method
- SetIVBISPointer Method
- GetIVBIS Method
- Refresh Method
- RunCommandSelectedRows Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

These methods are also used by the following:

- BatchAdd ActiveX Control
- BatchRecipeList ActiveX Control
- BatchOperatorPromptsList ActiveX Control
- BatchBindingPromptsList ActiveX Control
- BatchAlarmList ActiveX Control

NOTE: *The BatchRecipeList and the BatchAlarmList ActiveX controls do not use the RunCommandSelectedRows, GetCommandSignatureRequirement, or SetCommand SignatureRequirements methods.*

ConnectToServer Method

Description

Establishes the connection to the VBIS Server.

C++ Syntax

```
BOOL CBatchList:: ConnectToServer();
```

Visual Basic Syntax

```
[form.]Control.ConnectToServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if a connection is made.

- 0 if a connection is not made.

C++ Example

```
BOOL result = pBatchList->ConnectToServer();
```

Visual Basic Example

```
BatchList1.ConnectToServer
```

DisconnectFromServer Method

Description

Disconnects from the currently connected VBIS Server.

C++ Syntax

```
BOOL CBatchList::DisconnectFromServer();
```

Visual Basic Syntax

```
[form.]Control.DisconnectFromServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the VBIS Server is disconnected.
- 0 if the VBIS Server is not connected.

C++ Example

```
BOOL result = pBatchList->DisconnectFromServer();
```

Visual Basic Example

```
BatchList1.DisconnectFromServer
```

SelectRowByID Method

Description

Removes the cursor selection from any currently selected row, and then selects a row indicated by a unique ID.

C++ Syntax

```
BOOL CBatchList::SelectRowByID(string strUniqueID);
```

Visual Basic Syntax

```
[form.]Control.SelectRowByID(strUniqueID As String) As Boolean
```

Parameters

Parameter	Description
strUniqueID	A unique ID that indicates: • The batch serial number for the BatchList control. • The recipe ID for the BatchAdd and BatchRecipeList controls. • The event ID for the BatchOperatorPromptsList and BatchBindingPromptsList controls. • The phase ID for the BatchAlarmList control. The strUniqueID parameter is not case sensitive.

Return Type

Boolean.

- 0 = unsuccessful in moving the cursor to the indicated row
- 1 = successful in moving the cursor to the indicated row

C++ Example

```
BOOL result = pBatchList->SelectRowByID("61")  // where 61 is a batch serial number
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchList1.SelectRowByID ("61")   ' where 61 is a batch serial number
```

SelectRowByRowNumber Method

Description

Moves the cursor selection to a specified row number.

C++ Syntax

```
BOOL CBatchList:: SelectRowByRowNumber(long lRowNumber);
```

Visual Basic Syntax

```
[form.]Control.SelectRowByRowNumber(lRowNumber As Long) As Boolean
```

Parameters

Parameter	Description
lRowNumber	The row number that you want to select. Valid values for lRowNumber range from 1, to the total number of data rows plus one.

Return Type

Boolean.

- 0 = unsuccessful in moving the cursor to the indicated row
- 1 = successful in moving the cursor to the indicated row

C++ Example

```
BOOL result = pBatchList->SelectRowByRowNumber(2)  // where 2 is the row number
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchList1.SelectRowByRowNumber(2)  ' where 2 is the row number
```

GetNumberOfDataRows Method

Description

Returns the number of data rows displayed in the spreadsheet.

C++ Syntax

```
LONG CBatchList:: GetNumberOfDataRows();
```

Visual Basic Syntax

```
[form.]Control.GetNumberOfDataRows() As Long
```

Return Type

Long.

C++ Example

```
LONG RowNum = pBatchList->GetNumberOfDataRows();
```

Visual Basic Example

```
Dim RowNum as Long
RowNum = BatchList1.GetNumberOfDataRows()
```

GetSelectedRowData Method

Description

Returns the data for one row in a variant. If more than one row is selected, the method returns the last row that was selected.

C++ Syntax

```
BOOL CBatchList::GetSelectedRowData(VARIANT* pvDataRecord);
```

Visual Basic Syntax

```
[form.]Control.GetSelectedRowData(pvDataRecord) As Boolean
```

Parameters

Parameter	Description
PvDataRecord	The column information.

Return Type

Boolean.

- 1 if rows are selected.
- 0 if rows are not selected.

C++ Example

```
CBatchList* pBatchList = (CBatchList*) GetDlgItem(IDC_BATCHLISTCTRL1);

VARIANT vDataRecord;
VariantInit(&vDataRecord);
vDataRecord.vt = VT_VARIANT | VT_ARRAY;
vDataRecord.parray = NULL;
pBatchList->GetSelectedRowData(&vDataRecord);

SAFEARRAY* psaData;
psaData = vDataRecord.parray;
CString strColumnInfo;
```

```
VARIANT varData;
VariantInit (&varData);

// loop through the 26 columns in batchlist
for (long i=0;i<26;i++)
{
      SafeArrayGetElement (psaData, &i, &varData);
      strColumnInfo = varData.bstrVal;
      VariantClear (&varData);
      AfxMessageBox(strColumnInfo);
}
```

Visual Basic Example

```
Dim vRecord As Variant
Dim vArray(25) As Variant '26 columns in the batch list control
Dim bRet As Boolean
vRecord = vArray

bRet = BatchList1.GetSelectedRowData(vRecord)

For i = 0 To 25
      MsgBox (vRecord(i))
Next i
```

SetColumnOrder Method

Description

Sets the order of the columns using an array of longs.

C++ Syntax

```
BOOL CBatchList::SetColumnOrder(const VARIANT& vColumnOrderArray);
```

Visual Basic Syntax

```
[form.]Control.SetColumnOrder(vColumnOrderArray) As Boolean
```

Parameters

Parameter	Description
vColumnOrderArray	An array of columns indexed by absolute order and stored as relative.

Return Type

Boolean.

- 0 if an array is returned.

- 1 if an array is not returned.

  NOTE: *The function will not be successful if all twenty-six array values are not set or if any of the values are repeated.*

C++ Example

```
CBatchList* pBatchList = (CBatchList*) GetDlgItem(IDC_BATCHLISTCTRL1);

VARIANT vDataRecord;
VariantInit(&vDataRecord);
vDataRecord.vt = VT_VARIANT | VT_ARRAY;
vDataRecord.parray = NULL;

SAFEARRAY* psaRecord;
SAFEARRAYBOUND rgsabound[1];

// set the array bounds (for the 26 columns in batchlist)
rgsabound[0].lLbound = 0;
rgsabound[0].cElements = 26;

// create the safe array descriptor
psaRecord = SafeArrayCreate(VT_I4, 1, rgsabound);

if (psaRecord == NULL)
return;

long columnOrder[26]; // 26 columns in batch list

// init to 26,25,24,...1 so that the order is reversed.
for (long i=0;i<26;i++)
{
      columnOrder[i] = 26-i;
      SafeArrayPutElement(psaRecord, &i, &columnOrder[i]);
}

// copy the array
vDataRecord.parray = psaRecord;

pBatchList->SetColumnOrder(vDataRecord);
VariantClear(&vDataRecord); //clear the memory
```

Visual Basic Example

```
Dim bRet As Boolean
Static BatchList1_Reversed As Boolean

Dim vRecord As Variant
Dim vArray(24) As Long '26 columns in the batch list control

      'reverse the column order
      For i = 0 To 25
           vArray(i) = 26 - i ' make it 26, 25, 24, 23, ...,1
      Next i

vRecord = vArray
```

```
bRet = BatchList1.SetColumnOrder(vRecord)
```

Remarks

The following list provides the twenty-six columns, in relative order, available for the Batch List.

- BatchID
- Recipe
- RecipeVersion
- Description
- Scale
- StartTime
- ElapsedTime
- Failure
- State
- Mode
- Type
- ParametersRequired
- UnitsRequired
- ParametersSupported
- UnitsSupported
- BatchBound
- DefaultBinding
- OpBindParameters
- OpBindUnits
- OpInteraction
- ProcessCell
- Phase
- Unit
- InternalID
- CommandMask
- RecipeAuditVersion

Column order is based upon two situations: relative and absolute. Absolute column order is based upon the order of the columns at that instance. Relative order is based upon the default order of the columns. Each column is numbered sequentially. For example, the columns listed in the following table show the default order for the BatchList control's columns.

Column	Absolute Order	Relative Order
Batch ID	1	1
Description	2	2
Start Time	3	3
Elapsed Time	4	4
Scale	5	5
Failure	6	6

In this case, the columns appear as follows on the control.

Batch ID	Description	Start Time	Elapsed Time	Scale	Failure

If the user switches the order of columns 1 and 2, and also switches the order of columns 3 and 6, the new order assignments are as follows:

Column	Absolute Order	Relative Order
Description	1	2
Batch ID	2	1
Failure	3	6
Elapsed Time	4	4
Scale	5	5
Start Time	6	3

Note that the absolute column order does not change. In this case, specifying the absolute column 1 refers to the Description column, while referring to the relative column 1 refers to the Batch ID column.

In this case, the columns would appear as follows on the control:

Description	Batch ID	Failure	Elapsed Time	Scale	Start Time

SetSortKeys Method

Description

Sets the sort keys and their order. Each key is the absolute column. The keys correspond to column numbers.

C++ Syntax

```
BOOL CBatchList:: SetSortKeys(long lSortKey1, long lSortKey2, long lSortKey3, short sSortOrder1, short sSortOrder2, short sSortOrder3);
```

Visual Basic Syntax

```
[form.]Control.SetSortKeys(lSortKey1 As Long, lSortKey2 As Long, lSortKey3 As Long, sSortOrder1 As Long, sSortOrder2 As Long, sSortOrder3 As Long) As Boolean
```

Parameters

Parameter	**Description**
lSortKey1	The absolute column number of the sort key. 1. 0 = not used 2. 1 = ascending order 3. 2 = descending order
lSortKey2	The absolute column number of the sort key. 4. 0 = not used 5. 1 = ascending order 6. 2 = descending order
lSortKey3	The absolute column number of the sort key. 7. 0 = not used 8. 1 = ascending order 9. 2 = descending order
sSortOrder1	The absolute column number of the sort key.
sSortOrder2	The absolute column number of the sort key.

Parameter	Description
sSortOrder3	The absolute column number of the sort key.

Return Type

Boolean.

- 0 = sort order was not set
- 1 = successful

C++ Example

```
BOOL result = pBatchList->SetSortKeys(1, 2, 3, 1, 1, 1);
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchList1.SetSortKeys(1, 2, 3, 1, 1, 1)
```

Remarks

Specify 0 (zero) as the sort key if you do not want to use a sort key. A key cannot be 0 unless its successors are also 0.

Each sort key is the absolute column that the sort will be based upon. The order corresponds to the key of the same number. It can be ascending or descending.

SwapColumns Method

Description

Swaps the display order of two columns in the display. Column numbers are absolute.

C++ Syntax

```
BOOL CBatchList::SwapColumns(long lCol1, long lCol2);
```

Visual Basic Syntax

```
[form.]Control.SwapColumns(lCol1 As Long, lCol2 As Long) As Boolean
```

Parameters

Parameter	Description
lCol1	The absolute order of the first column to swap.
LCol2	The absolute order of the second column to swap.

C++ Example

```
BOOL result = pBatchList->SwapColumns(1, 2);
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchList1.SwapColumns(1, 2)
```

SetIVBISPointer Method

Description

Sets the internal VBIS pointer of the control to the given VBIS pointer. VBIS Server name is also passed, but only for display information. Make sure that the VBIS Server name is correct.

C++ Syntax

```
BOOL CBatchList:: SetIVBISPointer(IDispatch* pIVBIS, LPCTSTR lpstrServerName);
```

Visual Basic Syntax

```
[form.]Control.SetIVBISPointer(pIVBIS As Object, lpstrServerName As String) As Boolean
```

Parameters

Parameter	Description
pIVBIS	The pointer to the VBIS Server.
lpstrServerName	The name of the VBIS Server.

Return Type

Boolean.

- 1 if successful.
- 0 if not successful.

C++ Example

```
CString VBISServerName = pBatchList->GetVBISServerName();

// get pIVBIS
IVBIS8* pIVBIS=pBatchList1->GetIVBIS();
pBatchList2->SetIVBISPointer(pIVBISP, VBISServerName);
pIVBIS->Release();
```

Visual Basic Example

```
Dim VBISServerName as String

VBISServerName = BatchList1.VBISServerName
Set VBISP=BatchList1.GetIVBIS

' get VBISP
BatchList2.SetIVBISPointer VBISP, VBISServerName
Set VBISP = Nothing
```

NOTE: *These examples set the IVBIS pointer from one control and use it to set the IVBIS pointer of the second control.*

GetIVBIS Method

Description

Returns the IVBIS pointer to which the control is connected. If the control is not connected, it returns NULL. Call Release () on the pointer when done with it.

C++ Syntax

```
LPDISPATCH CBatchList::GetIVBIS();
```

Visual Basic Syntax

```
[form.]Control.GetIVBIS() As Objects
```

C++ Example

```
IVBIS8* pIVBIS = pBatchList->GetIVBIS();
...
///Release it when done.

pIVBIS->Release();
```

Visual Basic Example

```
Set VBISP = BatchList1.GetIVBIS
...
Set VBISP = Nothing
```

Refresh Method

Description

Refreshes the data display list. Returns TRUE if the data is refreshed.

C++ Syntax

```
BOOL CBatchList:: Refresh();
```

Visual Basic Syntax

```
[form.]Control.Refresh() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the refresh was successful.
- 0 if the refresh was not successful.

C++ Example

```
BOOL result = pBatchList->Refresh();
```

Visual Basic Example

```
BatchList1.Refresh
```

RunCommandSelectedRows Method

Description

Executes a command against a selected row of the spreadsheet.

C++ Syntax

```
LONG RunCommandSelectedRows(long command);
```

Visual Basic Syntax

```
Object.RunCommandSelectedRows(command As Long) As Long
```

Parameters

Parameter	Description
command	One of the following commands: 0 = START 1 = HOLD 2 = RESTART 3 = ABORT 4 = STOP 5 = MANUAL 6 = AUTO 7 = ADD 8 = REMOVE 9 = CLEAR_ALL_FAILURES

Return Type

The function returns the following error codes:

- 0 = SUCCESS
- 1 = UNKNOWNERROR
- 2 = COMMANDNOTSUPPORTED
- 3 = DATAERROR
- 4 = COMMANDNOTVALID

C++ Example

```
LONG status = pBatchList->RunCommandSelectedRows(0);
```

Visual Basic Example

```
Dim status as Long
Status = BatchList1.RunCommandSelectedRows(0)
```

GetCommandSignatureRequirements Method

Description

Gets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchList::GetCommandSignatureRequirements(long Command, long*
SignatureRequirements, BSTR* PerformedBy, BSTR* VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.GetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchList the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortBatch = 1 • bcAddBatch = 2 • bcAutoMode = 3 • bcBindingPrompts = 4 • bcClearAllFailures = 5 • bcHoldBatch = 6 • bcManualMode = 7 • bcOperatorPrompts = 8 • bcRemoveBatch = 9 • bcRestartBatch = 10 • bcStartBatch = 11 • bcStopBatch = 12 For BatchAdd the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAddBatch = 1 For BatchOperatorPromptsList and BatchBindingPromptsList the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAcknowledgePrompt = 1

Parameter	Description
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Remarks

The bcAddBatch constant exists in two places, Batchlistlib and BatchAddLib. Each constant has a different value. It is recommended that you fully indicate which constant that you intend to use. For example, in Visual Basic, use Command = BatchAddLib.bcAddBatch or Command = BatchListLib.bcAddBatch.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

     typedef enum _tagCommandID{
          bcDefault = 0,
          bcAbortBatch = 1,
          bcAddBatch = 2,
          bcAutoMode = 3,
          bcBindingPrompts = 4,
          bcClearAllFailures = 5,
          bcHoldBatch = 6,
          bcManualMode = 7,
```

```
          bcOperatorPrompts = 8,
          bcRemoveBatch = 9,
          bcRestartBatch = 10,
          bcStartBatch = 11,
          bcStopBatch = 12,
     } COMMANDID;


     BSTR bstrPerformedBy;
     BSTR bstrVerifiedBy;
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     // Example of reading the signature requirements for the
     // Start Batch Command.
     // Note, that if the UseDefaultSignatureRequirements is TRUE
     // then the setting for the Default command is used,
     // else the setting for the start batch command is used.

     if (m_pBatchList->GetUseDefaultSignatureRequirements ())
     {
          lCommand = bcDefault;
     }
     else
     {
          lCommand = bcStartBatch;
     }

     m_pBatchList->GetCommandSignatureRequirements ( lCommand, &lSignatureType,
&bstrPerformedBy, &bstrVerifiedBy);

     strPerformedBy = bstrPerformedBy;
     strVerifiedBy = bstrVerifiedBy;

     ::SysFreeString(bstrPerformedBy);
     ::SysFreeString(bstrVerifiedBy);

     if (lSignatureType == stNone)
     {
          AfxMessageBox ("No Signature requirements for the Start Batch Command");
     }
     else if (lSignatureType == stPerformedBy)
     {
          AfxMessageBox ("Signature requirements for the Start Batch Command are Perform
By: " + strPerformedBy);
     }
     else if (lSignatureType == stPerformedByVerifiedBy)
     {
          AfxMessageBox ("Signature requirements for the Start Batch Command are Perform
By: " + strPerformedBy + " Verify By: " + strVerifiedBy);
     }
```

Visual Basic Example

```
Private Sub Command1_Click()
Dim Command As BATCHLISTLib.COMMANDID
```

```
Dim SignatureType As BATCHLISTLib.SignatureType

Dim strPerformedBy As String
Dim strVerifiedBy As String

' example of reading the signature requirements for the
' Start Batch Command note that if the
' UseDefaultSignatureRequirements is TRUE then
' the setting for the Default command is used,
' else the setting for the start batch command is used.

If BatchList1.UseDefaultSignatureRequirements Then
    Command = bcDefault
Else
    Command = bcStartBatch
End If

BatchList1.GetCommandSignatureRequirements Command, SignatureType, strPerformedBy,
strVerifiedBy

If SignatureType = stNone Then
    MsgBox ("No Signature requirements for the Start Batch Command")
ElseIf SignatureType = stPerformedBy Then
    MsgBox ("Signature requirements for the Start Batch Command are Perform By: " +
strPerformedBy)
ElseIf SignatureType = stVerifiedBy Then
    MsgBox ("Signature requirements for the Start Batch Command are Perform By: " +
strPerformedBy + " Verified By: " + strVerifiedBy)
End If
```

SetCommandSignatureRequirements Method

Description

Sets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchList::SetCommandSignatureRequirements(Long Command, Long
SignatureRequirements, CString PerformedBy, CString VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.SetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchList the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortBatch = 1 • bcAddBatch = 2 • bcAutoMode = 3 • bcBindingPrompts = 4 • bcClearAllFailures = 5 • bcHoldBatch = 6 • bcManualMode = 7 • bcOperatorPrompts = 8 • bcRemoveBatch = 9 • bcRestartBatch = 10 • bcStartBatch = 11 • bcStopBatch = 12 For BatchAdd the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAddBatch = 1 For BatchOperatorPromptsList and BatchBindingPromptsList the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAcknowledgePrompt = 1
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.

Parameter	Description
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

     typedef enum _tagCommandID{
          bcDefault = 0,
          bcAbortBatch = 1,
          bcAddBatch = 2,
          bcAutoMode = 3,
          bcBindingPrompts = 4,
          bcClearAllFailures = 5,
          bcHoldBatch = 6,
          bcManualMode = 7,
          bcOperatorPrompts = 8,
          bcRemoveBatch = 9,
          bcRestartBatch = 10,
          bcStartBatch = 11,
          bcStopBatch = 12,
     } COMMANDID;


     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     strPerformedBy = _T("Operator");
     strVerifiedBy = _T("Supervisor");

     lCommand = bcStartBatch;
     lSignatureType = stPerformedByVerifiedBy;
     m_pBatchList->SetCommandSignatureRequirements( lCommand, lSignatureType,
strPerformedBy, strVerifiedBy );
```

Visual Basic Example

```
' example to set a specific command signature requirement

Dim Command As BATCHLISTLib.COMMANDID
Dim SignatureType As BATCHLISTLib.SignatureType
Dim strPerformedBy As String
Dim strVerifiedBy As String

Command = bcStopBatch
strPerformedBy = "Operator"
strVerifiedBy = "Supervisor"
SFC1.SetCommandSignatureRequirements Command, stPerformedByVerifiedBy, strPerformedBy,
strVerifiedBy

End Sub
```

BatchList Control Events

The BatchList control generates the following events:

- BatchAdded Event
- CommandExecuted Event
- ConnectedToServer Event
- DblClickList Event
- DblClickListEx Event
- DisconnectedFromServer Event
- Refresh Event
- RefreshEx Event
- RowActivated Event
- ServerChanged Event

All of the above events, except the DblClickListEx and CommandExecuted events, are also generated by these ActiveX controls:

- BatchAdd ActiveX Control
- BatchRecipeList ActiveX Control
- BatchOperatorPromptsList ActiveX Control
- BatchBindingPromptsList ActiveX Control
- BatchAlarmList ActiveX Control

NOTE: *The BatchRecipeList and the BatchAlarmList are the only controls in the above list that do not generate the CommandExecuted event. Only the BatchList control generates DblClickListEx events.*

The sections that follow describe each event in detail.

BatchAdded Event

Description

Occurs you add a batch using the Create Batch dialog box in the BatchList ActiveX control.

Event ID

607

C++ Syntax

```
Event BatchAdded(LPCTSTR RecipeID, LPCTSTR BatchID, long BatchSerialNumber);
```

Visual Basic Syntax

```
void BatchAdded(RecipeID As String, BatchID As String, BatchSerialNumber As Long)
```

Parameters

Parameter	Description
RecipeID	Recipe ID associated with the added batch.
BatchID	User defined batch ID associated with the added batch.
BatchSerialNumber	Batch serial number associated with this batch, generated internally by the server.

CommandExecuted Event

Description

Occurs when a command was successfully executed.

Event ID

606

C++ Syntax

```
void CommandExecuted(long Command);
```

Visual Basic Syntax

```
Event CommandExecuted(Command As Long)
```

Parameters

Parameter	Description
Command	Returns one of the values: 0 = START 1 = HOLD 2 = RESTART 3 = ABORT 4 = STOP 5 = MANUAL 6 = AUTO 7 = ADD 8 = REMOVE 9 = CLEAR_ALL_FAILURES

ConnectedToServer Event

Description

Occurs when the control has connected to the VBIS Server.

Event ID

602

C++ Syntax

```
void ConnectedToServer();
```

Visual Basic Syntax

```
Event ConnectedToSever()
```

DblClickList Event

Description

Occurs when the operator double-clicks within the data list.

NOTE: *This event occurs even if the MouseDblClickEnabled property is disabled.*

Event ID

605

C++ Syntax

```
void DblClickList(long Col, long Row);
```

Visual Basic Syntax

```
Event DblClickList(Col as Long, Row as Long)
```

Parameters

Parameter	Description
Col	The column that was double-clicked.
Row	The row that was double-clicked.

DblClickListEx Event

Description

This event is only available when the Enable Mouse Double Click check box is selected and the Perform BatchAdd/DblClickListEx Event option is enabled in the Miscellaneous properties.

Occurs when operator double-clicks within the data list. Unlike the DblClickList event, the DblClickListEx event also returns the batch serial number. For example, with this information you could programmatically launch the BatchSFC control displaying the specified batch, as shown in this example:

```
SFC1.SetCurrentBatch(BatchSerialNumber)
```

Event ID

609

C++ Syntax

```
Event DblClickListEx(long Col, long Row, long BatchSerialNumber);
```

Visual Basic Syntax

```
void DblClickListEx(Col As Long, Row As Long, BatchSerialNumber As Long)
```

Parameters

Parameter	Description
Col	The column that was double-clicked.
Row	The row that was double-clicked.
BatchSerialNumber	Batch serial number associated with this batch, generated internally by the server.

DisconnectedFromServer Event

Description

Occurs when the control has disconnected from the VBIS Server.

Event ID

603

C++ Syntax

```
void DisconnectedFromServer();
```

Visual Basic Syntax

```
Event DisconnectedFromServer()
```

Refresh Event

Description

Occurs when the control has refreshed its data.

Event ID

601

C++ Syntax

```
void Refresh();
```

Visual Basic Syntax

```
Event Refresh()
```

RefreshEx Event

Description

Occurs each time the control is refreshed. Returns the number of rows appearing in the list after any filters are applied. The default refresh rate is 5 seconds.

NOTE: *The BatchAdd and BatchRecipeList have a default refresh rate of 0 seconds, which means that the data is not automatically refreshed.*

Event ID

608

C++ Syntax

```
Event RefreshEx(long NumberOfRows);
```

Visual Basic Syntax

```
void RefreshEx(NumberOfRows As Long)
```

Parameters

Parameter	Description
NumberOfRows	Number of rows appearing in the list, after filtering has been applied.

RowActivated Event

Description

Occurs when the operator clicks on a row or scrolls to a row using the keyboard's up and down arrow keys. The activated row is the row containing the cursor.

Event ID

604

C++ Syntax

```
void RowActivated(long Row);
```

Visual Basic Syntax

```
Event RowActivated(Row as Long)
```

Parameters

Parameter	Description
Row	The row that was selected by the operator.

ServerChanged Event

Description

Occurs when the control has switched VBIS Servers.

Event ID

600

C++ Syntax

```
void ServerChanged();
```

Visual Basic Syntax

```
Event ServerChanged()
```

C++ Event Sink Map

The following shows an example of an event sink map.

```
BEGIN_EVENTSINK_MAP(CTestDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CTestDlg)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 607 /* BatchAdded */,
OnBatchAddedBatchlistctrl1, VTS_BSTR VTS_BSTR VTS_I4)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 602 /* ConnectedToServer */,
OnConnectedToServerBatchlistctrl1, VTS_NONE)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 606 /* CommandExecuted */,
OnCommandExecutedBatchlistctrl1, VTS_I4)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 605 /* DblClickList */,
OnDblClickListBatchlistctrl1, VTS_I4 VTS_I4)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 603 /* DisconnectedFromServer */,
OnDisconnectedFromServerBatchlistctrl1, VTS_NONE)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 604 /* RowActivated */,
OnRowActivatedBatchlistctrl1, VTS_I4)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 601 /* Refresh */, OnRefreshBatchlistctrl1,
VTS_NONE)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 600 /* ServerChanged */,
OnServerChangedBatchlistctrl1, VTS_NONE)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 608 /* RefreshEx */,
OnRefreshExBatchlistctrl1, VTS_I4)
     ON_EVENT(CTestDlg, IDC_BATCHLISTCTRL1, 609 /* DblClickListEx */,
OnDblClickListExBatchlistctrl1, VTS_I4 VTS_I4 VTS_I4)
     //}}AFX_EVENTSINK_MAP
```

```
END_EVENTSINK_MAP()
```

Visual Basic Event Procedures

These examples show procedures that Visual Basic creates for the BatchList events that you include in your project.

```
Private Sub BatchList1_CommandExecuted(ByVal command As Long)
Dim msgString As String
If (command = 0) Then
    MsgBox ("Start Executed")
ElseIf (command = 1) Then
    MsgBox ("Hold Executed")
ElseIf (command = 2) Then
    MsgBox ("Restart Executed")
ElseIf (command = 3) Then
    MsgBox ("Abort Executed")
ElseIf (command = 4) Then
    MsgBox ("Stop Executed")
ElseIf (command = 5) Then
    MsgBox ("Manual Mode Executed")
ElseIf (command = 6) Then
    MsgBox ("Automatic Mode Executed")
ElseIf (command = 7) Then
    ' There is now a better event with more information -
    ' BatchAdded event. A MsgBox is displayed from there.
    ' MsgBox ("Add Executed")
ElseIf (command = 8) Then
    MsgBox ("Remove Executed")
ElseIf (command = 9) Then
    MsgBox ("Clear All Failures Executed")
Else msgString = "Unknown Command Executed:" & command
    ' batch list commands are 0-9
    MsgBox (msgString)
End If
End Sub
Private Sub BatchList1_ConnectedToServer()
MsgBox ("Connected To Server")
BatchList1_Connected = True
End Sub
Private Sub BatchList1_DblClickList(ByVal Col As Long, ByVal Row As Long)
MsgBox ("DoubleClickList")
End Sub
Private Sub BatchList1_DisconnectedFromServer()
MsgBox ("Disconnected From Server")
BatchList1_Connected = False
End Sub
Private Sub BatchList1_Refresh()
End Sub
Private Sub BatchList1_RefreshEx(ByVal NumberOfRows As Long)
End Sub
Private Sub BatchList1_ServerChanged()
MsgBox ("Server Changed")
End Sub
Private Sub BatchList1_RowActivated(ByVal Row As Long)
End Sub
Private Sub BatchList1_BatchAdded(ByVal RecipeID As String, ByVal BatchID As String,
```

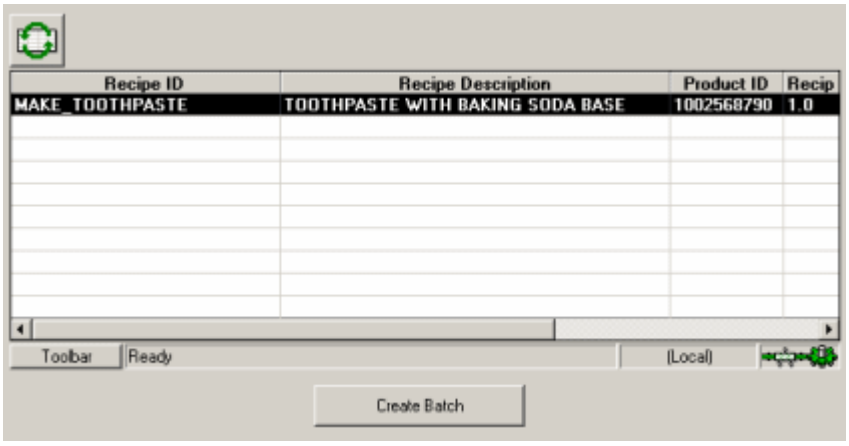
```
ByVal BatchSerialNumber As Long)
MsgBox "Batch Added: " + BatchID + Chr(13) + "Recipe: " + RecipeID + Chr(13) + "Batch
Serial Number: " + Str(BatchSerialNumber)
End Sub
Private Sub BatchList1_DblClickListEx(Col As Long, Row As Long, BatchSerialNumber As
Long)
SFC1.SetCurrentBatch BatchSerialNumber
End Sub
```

BatchAdd ActiveX Control

The "Intellution BatchAdd Control" lets operators add a batch to the BatchList. Operators can access this control by selecting the Add Batch button on the BatchList control's toolbar. The following figure shows the BatchAdd control.

From this control, operators select the recipe for which they want to create a batch and click the Create Batch button. Then, a wizard guides the operator through the process of adding the batch to the BatchList.

Using the control's property pages, designers can configure the control's GUI appearance and functionality. For example, the designer can configure the sort order of the recipe list. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.



Intellution BatchAdd ActiveX Control

Developers can access the BatchAdd control programmatically through Visual Basic or Visual C++. The properties, methods, and events for the BatchAdd control are described in the following sections.

BatchAdd Control Properties

The following sections describe each property for the BatchAdd control. The properties are grouped by the following functions:

- Column properties
- VBIS Server properties
- Miscellaneous
- Security properties
- Electronic Signature
- Color properties
- Font properties

BatchAdd Control Column Properties

The following table lists the properties that control the display of the columns in the BatchAdd control.

Column Properties	
Property	**Syntax**
RecipeAuditVersionFilter Sets the filter for the Audit Version column.	**C++ Syntax:** **CString** CBatchAdd::GetAuditVersionFilter(); **void** CBatchAdd::SetAuditVersionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionFilter[= *text$*]
RecipeAuditVersionHeaderText Specifies the column header text for the Audit Version column.	**C++ Syntax:** **CString** CBatchAdd::GetAuditVersionHeaderText(); **void** CBatchAdd::SetAuditVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionHeaderText[= *text$*]
RecipeAuditVersionVisible Sets whether or not to display the Audit Version column.	**C++ Syntax:** **BOOL** CBatchAdd::GetAuditVersionVisible (); **void** CBatchAdd::SetAuditVersionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AuditVersionVisible [= *boolvalue*]

Column Properties	
Property	**Syntax**
RecipeAuditVersionWidth Sets the width of the Audit Version column.	**C++ Syntax:** **double** CBatchAdd::GetAuditVersionWidth(); **void** CBatchAdd::SetAuditVersionWidth(double value); **Visual Basic Syntax:** [form.]*Control*.AuditVersionWidth[= *value!*]
RecipeReleasedFilter Sets the filter for the Recipe Released column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeReleasedFilter(); **void** CBatchAdd::SetRecipeReleasedFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeReleasedFilter[= *text$*]
RecipeReleasedHeaderText Specifies the column header text for the Recipe Released column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeReleasedHeaderText(); **void** CBatchAdd::SetRecipeReleasedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeReleasedHeaderText[= *text$*]
RecipeReleasedVisible Sets whether or not to display the Recipe Released column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeReleasedVisible (); **void** CBatchAdd::SetRecipeReleasedVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeReleasedVisible [= *boolvalue*]
RecipeReleasedWidth Sets the width of the Recipe Released column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeReleasedWidth(); **void** CBatchAdd::SetRecipeReleasedWidth(double value); **Visual Basic Syntax:** [form.]*Control*.RecipeReleasedWidth[= *value!*]

Column Properties	
Property	**Syntax**
RecipeProductNameFilter Sets the filter for the Recipe Product Name column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeProductNameFilter(); **void** CBatchAdd::SetRecipeProductNameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeProductNameFilter[= *text$*]
RecipeProductNameHeaderText Specifies the column header text for the Recipe Product Name column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeProductNameHeaderText(); **void** CBatchAdd::SetRecipeProductNameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeProductNameHeaderText[= *text$*]
RecipeProductNameVisible Sets whether or not to display the Recipe Product Name column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeProductNameVisible (); **void** CBatchAdd::SetRecipeProductNameVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeProductNameVisible [= *boolvalue*]
RecipeProductNameWidth Sets the width of the Recipe Product Name column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeProductNameWidth(); **void** CBatchAdd::SetRecipeProductNameWidth(double value); **Visual Basic Syntax:** [form.]*Control.*RecipeProductNameWidth[= *value!*]
AuthorFilter The filter for the Author column.	**C++ Syntax:** **CString** CBatchAdd::GetAuthorFilter(); **void** CBatchAdd::SetAuthorFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuthorFilter[= *text$*]

Column Properties	
Property	**Syntax**
AuthorHeaderText Specifies the column header text for the Author column.	**C++ Syntax:** **CString** CBatchAdd::GetAuthorHeaderText(); **void** CBatchAdd::SetAuthorHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AuthorHeaderText[= *text$*]
AuthorVisible Sets whether or not to display the Author column.	**C++ Syntax:** **BOOL** CBatchAdd::GetAuthorVisible(); **void** CBatchAdd::SetAuthorVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ProductIDVisible[= *boolvalue*]
AuthorWidth Sets the width of the Author column.	**C++ Syntax:** **double** CBatchAdd::GetAuthorWidth(); **void** CBatchAdd::SetAuthorWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*AuthorWidth[= *value!*]
ProductIDFilter The filter for the Product ID column.	**C++ Syntax:** **CString** CBatchAdd::GetProductIDFilter(); **void** CBatchAdd::SetProductIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProductIDFilter[= *text$*]
ProductIDHeaderText Specifies the column header text for the Product ID column.	**C++ Syntax:** **CString** CBatchAdd::GetProductIDHeaderText(); **void** CBatchAdd::SetProductIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProductIDHeaderText[= *text$*]
ProductIDVisible Sets whether or not to display the Product ID column.	**C++ Syntax:** **BOOL** CBatchAdd::GetProductIDVisible(); **void** CBatchAdd::SetProductIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ProductIDVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
RecipeDescriptionFilter The filter for the Recipe Description column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeDescriptionFilter(); **void** CBatchAdd::SetRecipeDescriptionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeDescriptionFilter[= *text$*]
RecipeDescriptionHeaderText Specifies the column header text for the Recipe Description column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeDescriptionHeaderText(); **void** CBatchAdd::SetRecipeDescriptionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeDescriptionHeaderText[= *text$*]
RecipeDescriptionVisible Sets whether or not to display the Recipe Description column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeDescriptionVisible(); **void** CBatchAdd::SetRecipeDescriptionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeDescriptionVisible[= *boolvalue*]
RecipeDescriptionWidth Sets the width of the Recipe Description column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeDescriptionWidth(); **void** CBatchAdd::SetRecipeDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeDescriptionWidth[= *value!*]
RecipeFilenameFilter The filter for the Recipe Filename column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeFilenameFilter(); **void** CBatchAdd::SetRecipeFilenameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilenameFilter[= *text$*]

Column Properties	
Property	**Syntax**
RecipeFilenameHeaderText Specifies the column header text for the Recipe Filename column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeFilenameHeaderText(); **void** CBatchAdd::SetRecipeFilenameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilenameHeaderText[= *text$*]
RecipeFilenameVisible Sets whether or not to display the Recipe Filename column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeFilenameVisible(); **void** CBatchAdd::SetRecipeFilenameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilenameVisible[= *boolvalue*]
RecipeFilenameWidth Sets the width of the Recipe Filename column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeFilenameWidth(); **void** CBatchAdd::SetRecipeFilenameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilenameWidth[= *value!*]
RecipeIDFilter The filter for the Recipe ID column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeIDFilter(); **void** CBatchAdd::SetRecipeIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeIDFilter[= *text$*]
RecipeIDHeaderText Specifies the column header text for the Recipe ID column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeIDHeaderText(); **void** CBatchAdd::SetRecipeIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeIDHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
RecipeIDVisible Sets whether or not to display the Recipe ID column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeIDVisible(); **void** CBatchAdd::SetRecipeIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeIDVisible[= *boolvalue*]
RecipeIDWidth Sets the width of the Recipe ID column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeIDWidth(); **void** CBatchAdd::SetRecipeIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeIDWidth[= *value!*]
RecipeStorageTypeFilter The filter for the Recipe Storage Type column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeStorageTypeFilter(); **void** CBatchAdd::SetRecipeStorageTypeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeStorageTypeFilter[= *text$*]
RecipeStorageTypeHeaderText Specifies the column header text for the Recipe Storage Type column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeStorageTypeHeaderText(); **void** CBatchAdd::SetRecipeStorageTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeStorageTypeHeaderText[= *text$*]
RecipeStorageTypeVisible Sets whether or not to display the Recipe Storage Type column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeStorageTypeVisible(); **void** CBatchAdd::SetRecipeStorageTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeStorageTypeVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
RecipeStorageTypeWidth Sets the width of the Recipe Storage Type column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeStorageTypeWidth(); **void** CBatchAdd::SetRecipeStorageTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeStorageTypeWidth[= *value!*]
RecipeTypeFilter The filter for the Recipe Type column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeTypeFilter(); **void** CBatchAdd::SetRecipeTypeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeTypeFilter[= *text$*]
RecipeTypeHeaderText Specifies the column header text for the Recipe Type column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeTypeHeaderText(); v**oid** CBatchAdd::SetRecipeTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeTypeHeaderText[= *text$*]
RecipeTypeVisible Sets whether or not to display the Recipe Type column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeTypeVisible(); **void** CBatchAdd::SetRecipeTypeVisible(**BOOL** value); **Visual Basic Syntax:** [form.]Control.RecipeTypeVisible[= *boolvalue*]
RecipeTypeWidth Sets the width of the Recipe Type column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeTypeWidth(); **void** CBatchAdd::SetRecipeTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeTypeWidth[= *value!*]

Column Properties	
Property	**Syntax**
RecipeVersionFilter The filter for the Recipe Version column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeVersionFilter(); **void** CBatchAdd::SetRecipeVersionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVersionFilter[= *text$*]
RecipeVersionHeaderText Specifies the column header text for the Recipe Version column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeVersionHeaderText(); **void** CBatchAdd::SetRecipeVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVersionHeaderText[= *text$*]
RecipeVersionVisible Sets whether or not to display the Recipe Version column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeVersionVisible(); **void** CBatchAdd::SetRecipeVersionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVersionVisible[= *boolvalue*]
RecipeVersionWidth Sets the width of the Recipe Version column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeVersionWidth(); **void** CBatchAdd::SetRecipeVersionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVersionWidth[= *value!*]
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchAdd::GetRowNumbersVisible(); **void** CBatchAdd::SetRowNumbersVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RowNumbersVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
TimeStampFilter The filter for the Time Stamp column.	**C++ Syntax:** **CString** CBatchAdd::GetTimeStampFilter(); **void** CBatchAdd::SetTimeStampFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeStampFilter[= *text$*]
TimeStampHeaderText Specifies the column header text for the Time Stamp column.	**C++ Syntax:** **CString** CBatchAdd::GetTimeStampHeaderText(); **void** CBatchAdd::SetTimeStampHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeStampHeaderText[= *text$*]
TimeStampVisible Sets whether or not to display the Time Stamp column.	**C++ Syntax:** **BOOL** CBatchAdd::GetTimeStampVisible(); **void** CBatchAdd::SetTimeStampVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeStampVisible[= *boolvalue*]
TimeStampWidth Sets the width of the Time Stamp column.	**C++ Syntax:** **double** CBatchAdd::GetTimeStampWidth(); **void** CBatchAdd::SetTimeStampWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeStampWidth[= *value!*]

BatchAdd Control VBIS Server Properties

The following table lists the VBIS Server properties for the BatchAdd control.

BatchAdd Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchAdd::GetConnectAtStartup(); **void** CBatchAdd::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchAdd ActiveX control is 0. ***NOTE:*** *The BatchRecipeList is the only other ActiveX control with a default Refresh Rate of 0. This is because the information in these lists does not change often. The other ActiveX controls have a default Refresh Rate of 5 seconds.*	**C++ Syntax:** **short** CBatchAdd::GetRefreshRate(); **void** CBatchAdd::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **Cstring** CBatchAdd::GetVBISServerName(); **void** CBatchAdd::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

BatchAdd Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchAdd control.

BatchAdd Control Miscellaneous Properties	
Property	**Syntax**
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchAdd::GetEnableRightMouseMenu(); **void** CBatchAdd::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]
MouseDblClickedEnabled Sets whether or not to enable operators to double-click a recipe to create a batch.	**C++ Syntax:** **BOOL** CBatchAdd::GetMouseDblClickedEnabled(); **void** CBatchAdd::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.MouseDblClickedEnabled[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchAdd::GetStatusBarEnabled(); **void** CBatchAdd::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StatusBarEnabled[= *boolvalue*]
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchAdd::GetToolBarEnabled(); **void** CBatchAdd::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarEnabled[= *boolvalue*]
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control. 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchAdd::GetToolBarPosition(); **void** CBatchAdd::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarPosition[= *value%*]

BatchAdd Control Miscellaneous Properties	
Property	**Syntax**
DecimalValue Sets the precision of the data. The default value is 2 places after the decimal point.	**C++ Syntax:** **short** CBatchAdd::GetDecimalValue(); **void** CBatchAdd::SetDecimalValue(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.DecimalValue[= *value%*]
VerifyCommandActions The BatchAdd control does not currently use this property.	N/A
ShowFormulationRecipeRadioButtons Sets whether the display options (to show master recipes, formulations, or both) appear on the BatchAdd control. 0 does not display the option buttons. 1 (default) displays the option buttons.	**C++ Syntax:** **BOOL** CBatchAdd::GetShowFormulationRecipeRadioButtons(); **void** CBatchAdd::SetShowFormulationRecipeRadioButtons(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ShowFormulationRecipeRadioButtons[= *boolvalue*]
SetFormulationRecipeRadioButtonValue If the ShowFormulationRecipeRadioButtons property is enabled, recipe options appear on the BatchAdd control: 0 displays all master recipes and formulations in the list. 1 displays only formulations in the list. 2 displays only master recipes in the list.	**C++ Syntax:** **short** CBatchAdd::GetSetFormulationRecipeRadioButtonValue(); **void** CBatchAdd::SetSetFormulationRecipeRadioButtonValue(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SetFormulationRecipeRadioButtonValue[= *value%*]

BatchAdd Control Security Properties

The following table lists the security properties for the BatchAdd control

BatchAdd Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchAdd::GetColumnEditEnabled(); **void** CBatchAdd::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ColumnEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchAdd::GetEnableIFIXSecurity(); **void** CBatchAdd::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue*]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchAdd::GetMiscEditEnabled(); **void** CBatchAdd::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchAdd::GetRefreshRateEditEnabled(); **void** CBatchAdd::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= *boolvalue*]

BatchAdd Control Security Properties	
Property	**Syntax**
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchAdd::GetServerEditEnabled(); **void** CBatchAdd::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ServerEditEnabled [= *boolvalue*]
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchAdd::GetSortOrderEditEnabled(); **void** CBatchAdd::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SortOrderEditEnabled[= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchAdd::GetToggleConnectionEnabled(); **void** CBatchAdd::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToggleConnectionEnabled[= *boolvalue*]

BatchAdd Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchAdd control.

BatchAdd Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements Sets a default signature type (None, Performed By, Performed By/Verified By) to all of the commands for this ActiveX control. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchAdd::GetUseDefaultSignatureRequirements();**void** CBatchAdd::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

BatchAdd Control Color Properties

The following table lists the color properties for the BatchAdd control.

BatchAdd Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetBackColor(); **void** CBatchAdd::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetEvenRowBackColor(); **void** CBatchAdd::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]

BatchAdd Control Color Properties	
Property	**Syntax**
EvenRowTextColor Sets the color of text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetEvenRowTextColor(); **void** CBatchAdd::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetGridColor(); **void** CBatchAdd::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetHeaderBackColor(); **void** CBatchAdd::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetHeaderTextColor(); **void** CBatchAdd::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetOddRowBackColor(); **void** CBatchAdd::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]

BatchAdd Control Color Properties	
Property	**Syntax**
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAdd::GetOddRowTextColor(); **void** CBatchAdd::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

NOTE: *For examples on setting color properties, refer to the BatchList ActiveX Control section.*

BatchAdd Control Font Properties

The following table lists the font properties for the BatchAdd control.

BatchAdd Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchAdd::GetHeaderFont(); **void** CBatchAdd::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderFont[= *StdFontVariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchAdd::GetTextFont(); **void** CBatchAdd::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *StdFontVariable*]

NOTE: *For examples on setting font properties, refer to the BatchList ActiveX Control section.*

BatchAdd Control Methods

The BatchAdd control supports the following methods:

- SetDoneButton Method
- ConnectToServer Method

- DisconnectFromServer Method
- GetIVBIS Method
- SetIVBISPointer Method
- SelectRowByID Method
- SelectRowByRowNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method
- RunCommandSelectedRows Method
- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

The SetDoneButton method is described in the following section. The remaining methods are the same for each control that uses them. Click the SetDoneButton method to link to the following section describing that method. Click on a common method to link to a section in the BatchList ActiveX Control section that describes the method using the BatchList ActiveX control as an example.

SetDoneButton Method

Description

Sets whether the Done button is displayed on the BatchAdd control. When the Done button is clicked, it fires the ExitBatchAddControl event. Programmers can capture this event and then close the control.

C++ Syntax

```
BOOL CBatchAdd::SetDoneButton(long ButtonState);
```

Visual Basic Syntax

```
[form.]Control.SetDoneButton[=value% ]
```

Parameters

Parameter	Description
ButtonState	The state of the button: 1 = display button 0 = do not display button

Return Type

Boolean.

C++ Example

```
pBatchAddList->SetDoneButton(1);
```

Visual Basic Example

```
BatchAddList.SetDoneButton(1);
```

BatchAdd Control Events

The BatchAdd control generates the following events:

- ExitBatchAddControl Event
- CommandExecuted Event
- CommandExecutedEx Event
- ConnectedToServer Event
- DblClickList Event
- DisconnectedFromServer Event
- Refresh Event
- RowActivated Event
- ServerChanged Event
- RefreshEx Event

The ExitBatchAddControl event is described in the following section. The remaining events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

ExitBatchAddControl Event

Description

Occurs when the operator clicks the Done button to exit the BatchAdd control. This event is useful in situations where the dialog or application containing the control needs to know when the operator has exited the BatchAdd control.

Event ID

675

C++ Syntax

```
afx_msg void ExitBatchAddControl();
```

Visual Basic Syntax

```
Event ExitBatchAddControl()
```

CommandExecutedEx Event

Description

Occurs when a batch add is successfully executed. This event returns more details than the CommandExecuted event.

Event ID

676

C++ Syntax

```
afx_msg void CommandExecutedEx(long command, CString RecipeID, CString BatchId, long
BatchSerialNumber);
```

Visual Basic Syntax

```
Event CommandExecutedEx(ByVal Command as long, ByVal RecipeId as string, ByVal BatchId as
string, ByVal BatchSerialNumber as long)
```

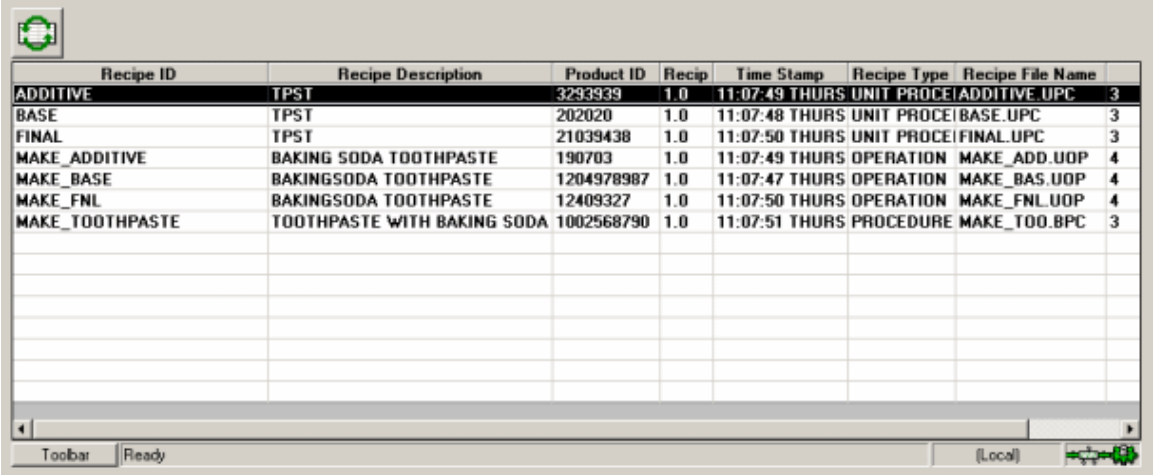
Parameters

Parameter	Description
Command	7 = Add
RecipeId	Recipe ID associated with this batch.
BatchId	Batch ID associated with this batch.
BatchSerialNumber	Batch serial number associated with this batch.

BatchRecipeList ActiveX Control

The "Intellution BatchRecipeList Control" displays a list of recipes. The following figure shows the BatchRecipeList control.

Using the control's property pages, designers can configure the control's GUI appearance and functionality. For example, the designer can configure the colors, fonts, and column widths for the recipe list. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.



Intellution BatchRecipeList ActiveX Control

Developers can access the BatchRecipeList control programmatically through Visual Basic or Visual C++. The properties, methods, and events for the BatchRecipeList control are described in the following sections.

BatchRecipeList Control Properties

The following sections describe each property for the BatchRecipeList control. The properties are grouped into the following categories:

- Column properties
- Security properties
- VBIS Server properties
- Miscellaneous
- Color properties
- Font properties

BatchRecipeList Control Column Properties

The following table lists the properties that control the display of the columns in the BatchRecipeList control.

Column Properties	
Property	**Syntax**
RecipeAuditVersionFilter Sets the filter for the Audit Version column.	**C++ Syntax:** **CString** CBatchRecipeList::GetAuditVersionFilter(); **void** CBatchRecipeList::SetAuditVersionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AuditVersionFilter[= *text$*]
RecipeAuditVersionHeaderText Specifies the column header text for the Audit Version column.	**C++ Syntax:** **CString** CBatchRecipeList::GetAuditVersionHeaderText(); **void** CBatchRecipeList::SetAuditVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AuditVersionHeaderText[= *text$*]
RecipeAuditVersionVisible Sets whether or not to display the Audit Version column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetAuditVersionVisible (); **void** CBatchRecipeList::SetAuditVersionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.AuditVersionVisible [= *boolvalue*]
RecipeAuditVersionWidth Sets the width of the Audit Version column.	**C++ Syntax:** **double** CBatchRecipeList::GetAuditVersionWidth(); **void** CBatchRecipeList::SetAuditVersionWidth(double value); **Visual Basic Syntax:** [form.]*Control*.AuditVersionWidth[= *value!*]
RecipeProductNameFilter Sets the filter for the Recipe Product Name column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeProductNameFilter(); **void** CBatchAdd::SetRecipeProductNameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeProductNameFilter[= *text$*]

Column Properties	
Property	**Syntax**
RecipeProductNameHeaderText Specifies the column header text for the Recipe Product Name column.	**C++ Syntax:** **CString** CBatchAdd::GetRecipeProductNameHeaderText(); **void** CBatchAdd::SetRecipeProductNameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeProductNameHeaderText[= *text$*]
RecipeProductNameVisible Sets whether or not to display the Recipe Product Name column.	**C++ Syntax:** **BOOL** CBatchAdd::GetRecipeProductNameVisible (); **void** CBatchAdd::SetRecipeProductNameVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeProductNameVisible [= *boolvalue*]
RecipeProductNameWidth Sets the width of the Recipe Product Name column.	**C++ Syntax:** **double** CBatchAdd::GetRecipeProductNameWidth(); **void** CBatchAdd::SetRecipeProductNameWidth(double value); **Visual Basic Syntax:** [form.]*Control*.RecipeProductNameWidth[= *value!*]
AuthorFilter The filter for the Author column.	**C++ Syntax:** **CString** CBatchRecipeList::GetAuthorFilter(); **void** CBatchRecipeList::SetAuthorFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AuthorFilter[= *text$*]
AuthorHeaderText Specifies the column header text for the Author column.	**C++ Syntax:** **CString** CBatchRecipeList::GetAuthorHeaderText(); **void** CBatchRecipeList::SetAuthorHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AuthorHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
AuthorVisible Sets whether or not to display the Author column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetAuthorVisible(); **void** CBatchRecipeList::SetAuthorVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ProductIDVisible[= *boolvalue*]
AuthorWidth Sets the width of the Author column.	**C++ Syntax:** **double** CBatchRecipeList::GetAuthorWidth(); **void** CBatchRecipeList::SetAuthorWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.AuthorWidth[= *value!*]
ProductIDFilter The filter for the Product ID column.	**C++ Syntax:** **CString** CBatchRecipeList::GetProductIDFilter(); **void** CBatchRecipeList::SetProductIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ProductIDFilter[= *text$*]
ProductIDHeaderText Specifies the column header text for the Product ID column.	**C++ Syntax:** **CString** CBatchRecipeList::GetProductIDHeaderText(); **void** CBatchRecipeList::SetProductIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ProductIDHeaderText[= *text$*]
ProductIDVisible Sets whether or not to display the Product ID column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetProductIDVisible(); **void** CBatchRecipeList::SetProductIDVisible(**BOOL** value); **Visual Basic Syntax:** [form.]Control.ProductIDVisible[= boolvalue]
RecipeDescriptionFilter The filter for the Recipe Description column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeDescriptionFilter(); **void** CBatchRecipeList::SetRecipeDescriptionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeDescriptionFilter[= *text$*]

Column Properties	
Property	**Syntax**
RecipeDescriptionHeaderText Specifies the column header text for the Recipe Description column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeDescriptionHeaderText(); **void** CBatchRecipeList::SetRecipeDescriptionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeDescriptionHeaderText[= *text$*]
RecipeDescriptionVisible Sets whether or not to display the Recipe Description column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeDescriptionVisible(); **void** CBatchRecipeList::SetRecipeDescriptionVisible(**BOOL** value); **Visual Basic Syntax:** [form.]Control.RecipeDescriptionVisible[= boolvalue]
RecipeDescriptionWidth Sets the width of the Recipe Description column.	**C++ Syntax:** **double** CBatchRecipeList::GetRecipeDescriptionWidth(); **void** CBatchRecipeList::SetRecipeDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeDescriptionWidth[= *value!*]
RecipeFilenameFilter The filter for the Recipe Filename column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeFilenameFilter(); **void** CBatchRecipeList::SetRecipeFilenameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeFilenameFilter[= *text$*]
RecipeFilenameHeaderText Specifies the column header text for the Recipe Filename column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeFilenameHeaderText(); **void** CBatchRecipeList::SetRecipeFilenameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeFilenameHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
RecipeFilenameVisible Sets whether or not to display the Recipe Filename column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeFilenameVisible(); **void** CBatchRecipeList::SetRecipeFilenameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeFilenameVisible[= *boolvalue*]
RecipeFilenameWidth Sets the width of the Recipe Filename column.	**C++ Syntax:** **double** CBatchRecipeList::GetRecipeFilenameWidth(); **void** CBatchRecipeList::SetRecipeFilenameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeFilenameWidth[= *value!*]
RecipeIDFilter The filter for the Recipe ID column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeIDFilter(); **void** CBatchRecipeList::SetRecipeIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeIDFilter[= *text$*]
RecipeIDHeaderText Specifies the column header text for the Recipe ID column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeIDHeaderText(); **void** CBatchRecipeList::SetRecipeIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeIDHeaderText[= *text$*]
RecipeIDVisible Sets whether or not to display the Recipe ID column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeIDVisible(); **void** CBatchRecipeList::SetRecipeIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeIDVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
RecipeIDWidth Sets the width of the Recipe ID column.	**C++ Syntax:** **double** CBatchRecipeList::GetRecipeIDWidth(); **void** CBatchRecipeList::SetRecipeIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeIDWidth[= *value!*]
RecipeStorageTypeFilter The filter for the Recipe Storage Type column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeStorageTypeFilter(); **void** CBatchRecipeList::SetRecipeStorageTypeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeStorageTypeFilter[= *text$*]
RecipeStorageTypeHeaderText Specifies the column header text for the Recipe Storage Type column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeStorageTypeHeaderText(); **void** CBatchRecipeList::SetRecipeStorageTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeStorageTypeHeaderText[= *text$*]
RecipeStorageTypeVisible Sets whether or not to display the Recipe Storage Type column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeStorageTypeVisible(); **void** CBatchRecipeList::SetRecipeStorageTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeStorageTypeVisible[= *boolvalue*]

RecipeStorageTypeWidth
Sets the width of the Recipe Storage Type column.
	C++ Syntax:
	double CBatchRecipeList::GetRecipeStorageTypeWidth();
	void CBatchRecipeList::SetRecipeStorageTypeWidth(double value);
	Visual Basic Syntax:
	[*form.*]*Control*.RecipeStorageTypeWidth[= *value!*]

Column Properties	
Property	**Syntax**
RecipeTypeFilter The filter for the Recipe Type column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeTypeFilter(); **void** CBatchRecipeList::SetRecipeTypeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeTypeFilter[= *text$*]
RecipeTypeHeaderText Specifies the column header text for the Recipe Type column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeTypeHeaderText(); **void** CBatchRecipeList::SetRecipeTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeTypeHeaderText[= *text$*]
RecipeTypeVisible Sets whether or not to display the Recipe Type column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeTypeVisible(); **void** CBatchRecipeList::SetRecipeTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeTypeVisible[= *boolvalue*]
RecipeTypeWidth Sets the width of the Recipe Type column.	**C++ Syntax:** **double** CBatchRecipeList::GetRecipeTypeWidth(); **void** CBatchRecipeList::SetRecipeTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeTypeWidth[= *value!*]
RecipeVersionFilter The filter for the Recipe Version column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeVersionFilter(); **void** CBatchRecipeList::SetRecipeVersionFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVersionFilter[= *text$*]

Column Properties	
Property	**Syntax**
RecipeVersionHeaderText Specifies the column header text for the Recipe Version column.	**C++ Syntax:** **CString** CBatchRecipeList::GetRecipeVersionHeaderText(); **void** CBatchRecipeList::SetRecipeVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVersionHeaderText[= *text$*]
RecipeVersionVisible Sets whether or not to display the Recipe Version column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRecipeVersionVisible(); **void** CBatchRecipeList::SetRecipeVersionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVersionVisible[= *boolvalue*]
RecipeVersionWidth Sets the width of the Recipe Version column.	**C++ Syntax:** **double** CBatchRecipeList::GetRecipeVersionWidth(); **void** CBatchRecipeList::SetRecipeVersionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVersionWidth[= *value!*]
ReleasedFilter The filter for the Released column.	**C++ Syntax:** **CString** CBatchRecipeList::GetReleasedFilter(); **void** CBatchRecipeList::SetReleasedFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ReleasedFilter[= *text$*]
ReleasedHeaderText Specifies the column header text for the Released column.	**C++ Syntax:** **CString** CBatchRecipeList::GetReleasedHeaderText(); **void** CBatchRecipeList::SetReleasedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ReleasedHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
ReleasedVisible Sets whether or not to display the Released column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetReleasedVisible(); **void** CBatchRecipeList::SetReleasedVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ReleasedVisible [= *boolvalue*]
ReleasedWidth Sets the width of the Released column.	**C++ Syntax:** **double** CBatchRecipeList::GetReleasedWidth(); **void** CBatchRecipeList::SetReleasedWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.ReleasedWidth [= *value!*]
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRowNumbersVisible(); **void** CBatchRecipeList::SetRowNumbersVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RowNumbersVisible[= *boolvalue*]
TimeStampFilter The filter for the Time Stamp column.	**C++ Syntax:** **CString** CBatchRecipeList::GetTimeStampFilter(); **void** CBatchRecipeList::SetTimeStampFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeStampFilter[= *text$*]
TimeStampHeaderText Specifies the column header text for the Time Stamp column.	**C++ Syntax:** **CString** CBatchRecipeList::GetTimeStampHeaderText(); **void** CBatchRecipeList::SetTimeStampHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeStampHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
TimeStampVisible Sets whether or not to display the Time Stamp column.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetTimeStampVisible(); **void** CBatchRecipeList::SetTimeStampVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeStampVisible[= *boolvalue*]
TimeStampWidth Sets the width of the Time Stamp column.	**C++ Syntax:** **double** CBatchRecipeList::GetTimeStampWidth(); **void** CBatchRecipeList::SetTimeStampWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVersionWidth[= *value!*]

BatchRecipeList Control Security Properties

The following table lists the properties that control the security settings for elements on the BatchRecipeList control

BatchRecipeList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetColumnEditEnabled(); **void** CBatchRecipeList::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ColumnEditEnabled[= booleanvalue]

BatchRecipeList Control Security Properties	
Property	**Syntax**
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchRecipeList::GetEnableIFIXSecurity(); **void** CBatchRecipeList::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= booleanvalue]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetMiscEditEnabled(); **void** CBatchRecipeList::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetRefreshRateEditEnabled(); **void** CBatchRecipeList::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= booleanvalue]
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetServerEditEnabled(); **void** CBatchRecipeList::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]

BatchRecipeList Control Security Properties	
Property	**Syntax**
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetSortOrderEditEnabled(); **void** CBatchRecipeList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*SortOrderEditEnabled[= booleanvalue]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetToggleConnectionEnabled(); **void** CBatchRecipeList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= booleanvalue]

BatchRecipeList Control VBIS Server Properties

The following table lists the VBIS Server properties for the BatchRecipeList control.

BatchRecipeList Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetConnectAtStartup(); **void** CBatchRecipeList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]

RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchRecipeList ActiveX control is 0. ***NOTE:*** *The BatchAdd control is the only other ActiveX control with a default Refresh Rate of 0. This is because the information in these lists does not change often. The other ActiveX controls have a default Refresh Rate of 5 seconds.*	**C++ Syntax:** **short** CBatchRecipeList::GetRefreshRate(); **void** CBatchRecipeList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchRecipeList::GetVBISServerName(); **void** CBatchRecipeList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

BatchRecipeList Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchRecipeList control.

BatchRecipeList Control Miscellaneous Properties	
Property	**Syntax**
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetEnableRightMouseMenu(); **void** CBatchRecipeList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableRightMouseMenu[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetStatusBarEnabled(); **void** CBatchRecipeList::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StatusBarEnabled[= *boolvalue*]

BatchRecipeList Control Miscellaneous Properties	
Property	**Syntax**
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchRecipeList::GetToolBarEnabled(); **void** CBatchRecipeList::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarEnabled[= *boolvalue*]
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control. 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchRecipeList::GetToolBarPosition(); **void** CBatchRecipeList::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarPosition[= *value%*]

BatchRecipeList Control Color Properties

The following table lists the color properties for the BatchRecipeList control.

BatchRecipeList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetBackColor(); **void** CBatchRecipeList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color%*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetEvenRowBackColor(); **void** CBatchRecipeList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]

BatchRecipeList Control Color Properties	
Property	**Syntax**
EvenRowTextColor Sets the color of text in the even rows on the data list.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetEvenRowTextColor(); **void** CBatchRecipeList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetGridColor(); **void** CBatchRecipeList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetHeaderBackColor(); **void** CBatchRecipeList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color%*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetHeaderTextColor(); **void** CBatchRecipeList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows on the data list.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetOddRowBackColor(); **void** CBatchRecipeList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]

BatchRecipeList Control Color Properties	
Property	**Syntax**
OddRowTextColor Sets the text color for odd rows on the data list.	**C++ Syntax:** **OLE_COLOR** CBatchRecipeList::GetOddRowTextColor(); **void** CBatchRecipeList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowTextColor[= *color%*]

NOTE: *For examples on setting Color properties, refer to the BatchList ActiveX Control section.*

BatchRecipeList Control Font Properties

The following table lists the font properties for the BatchRecipeList control.

BatchRecipeList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchRecipeList::GetHeaderFont(); **void** CBatchRecipeList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderFont[= *StdFontVariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchRecipeList::GetTextFont(); **void** CBatchRecipeList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control*.TextFont[= *StdFontVariable*]

NOTE: *For examples on setting Font properties, refer to the BatchList ActiveX Control section.*

BatchRecipeList Control Methods

The BatchRecipeList control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- GetIVBIS Method
- SetIVBISPointer Method
- SelectRowByID Method
- SelectRowByRowNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method
- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method

These methods are the same for each control. Click on a method above to link to a section in the BatchList ActiveX Control section that describes the common method using the BatchList ActiveX control as an example.

BatchRecipeList Control Events

The BatchRecipeList control generates the following events:

- ConnectedToServer Event
- DblClickList Event
- DisconnectedFromServer Event
- Refresh Event
- RowActivated Event
- ServerChanged Event
- RefreshEx Event

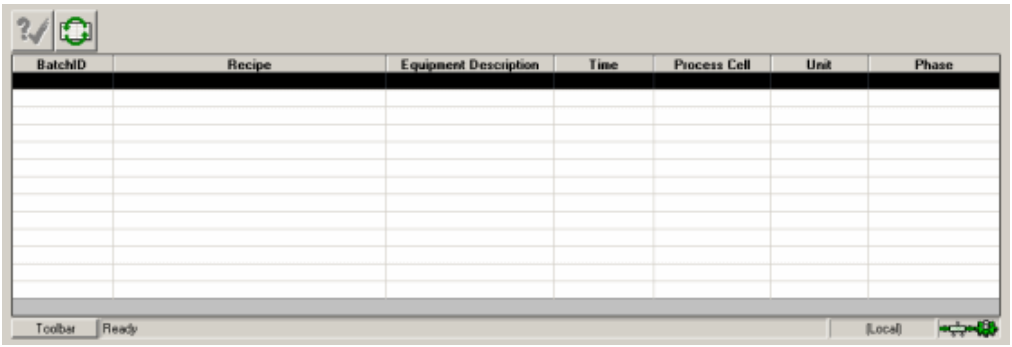
These events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

BatchOperatorPromptsList ActiveX Control

The "Intellution BatchOperatorPromptsList Control" displays a list of operator prompts. Operators can acknowledge prompts using this control. The following figure shows the BatchOperatorPromptsList control.

When operator prompts are ready to be acknowledged, the Display Prompts button on the BatchList control blinks. To acknowledge a prompt, operators click the Display Prompts button to display the BatchOperatorPromptsList control. From this control, the operator can then acknowledge awaiting prompts and the batch can continue executing.

Using the control's property pages, designers can configure the control's GUI appearance and functionality for operators. For example, the designer can configure the colors, fonts, and column widths for the operator prompts list. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.



Intellution BatchOperatorPromptsList ActiveX Control

Developers can access the BatchOperatorPromptsList control programmatically through Visual Basic or Visual C++. The properties, methods, and events for the BatchOperatorPromptsList control are described in the following sections.

BatchOperatorPromptsList Control Properties

The following sections describe each property for the BatchOperatorPromptsList control. The properties are grouped into the following categories:

- Column properties
- VBIS Server properties
- Miscellaneous properties
- Security properties
- Electronic Signature properties
- Command properties
- Color properties
- Font properties

BatchOperatorPromptsList Control Column Properties

The following table lists the properties that control the display of the columns in the BatchOperatorPromptsList control.

Column Properties	
Property	**Syntax**
AreaFilter Sets the filter for the Area column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetAreaFilter (); **void** CBatchOperatorPromptsList::SetAreaFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AreaFilter [= *text$*]
AreaHeaderText Specifies the header text for the Area column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetAreaHeaderText (); **void** CBatchOperatorPromptsList::SetAreaHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.AreaHeaderText [= *text$*]
AreaVisible Sets whether or not to display the Area column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetAreaVisible(); **void** CBatchOperatorPromptsList::SetAreaVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.AreaVisible[= *boolvalue*]
AreaWidth Sets the width of the Area column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetAreaWidth(); **void** CBatchOperatorPromptsList::SetAreaWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.AreaWidth[= value!]
BatchIDVisible Sets whether or not to display the Batch ID column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetBatchIDVisible(); **void** CBatchOperatorPromptsList::SetBatchIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
BatchIDFilter Sets the filter for the Batch ID column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetBatchIDFilter (); **void** CBatchOperatorPromptsList::SetBatchIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDFilter [= *text$*]
BatchIDHeaderText Specifies the header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetBatchIDHeaderText (); **void** CBatchOperatorPromptsList::SetBatchIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDHeaderText [= *text$*]
BatchIDWidth Sets the width of the Batch ID column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetBatchIDWidth(); **void** CBatchOperatorPromptsList::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDWidth[= value!]
DefaultFilter Sets the filter for the Default Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetDefaultFilter (); **void** CBatchOperatorPromptsList::SetDefaultFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultFilter [= *text$*]
DefaultHeaderText Specifies the header text for the Default Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetDefaultHeaderText (); **void** CBatchOperatorPromptsList::SetDefaultHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
DefaultVisible Sets whether or not to display the Default Value column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetDefaultVisible(); **void** CBatchOperatorPromptsList::SetDefaultVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultVisible[= *boolvalue*]
DefaultWidth Sets the width of the Default Value column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetDefaultWidth(); **void** CBatchOperatorPromptsList::SetDefaultWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultWidth[= value!]
EquipmentDescriptionFilter Sets the filter for the Equipment Description column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEquipmentDescriptionFilter (); **void** CBatchOperatorPromptsList::SetEquipmentDescriptionFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EquipmentDescriptionFilter [= *text$*]
EquipmentDescriptionHeaderText Specifies the header text for the Equipment Description column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEquipmentDescriptionHeaderText (); **void** CBatchOperatorPromptsList::SetEquipmentDescriptionHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EquipmentDescriptionHeaderText [= *text$*]
EquipmentDescriptionVisible Sets whether or not to display the Equipment Description column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEquipmentDescriptionVisible(); **void** CBatchOperatorPromptsList::SetEquipmentDescriptionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EquipmentDescriptionVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
EquipmentDescriptionWidth Sets the width of the Equipment Description column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetEquipmentDescriptionWidth(); **void** CBatchOperatorPromptsList::SetEquipmentDescriptionWidth(**double** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EquipmentDescriptionWidth[= value!]
EUFilter Sets the filter for the EU (Engineering Units) column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEUFilter (); **void** CBatchOperatorPromptsList::SetEUFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EUFilter [= *text$*]
EUHeaderText Specifies the header text for the EU (Engineering Units) column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEUHeaderText (); **void** CBatchOperatorPromptsList::SetEUHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EUHeaderText [= *text$*]
EUVisible Sets whether or not to display the EU (Engineering Units) column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEUVisible(); **void** CBatchOperatorPromptsList::SetEUVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EUVisible[= *boolvalue*]
EUWidth Sets the width of the EU (Engineering Units) column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetEUWidth(); **void** CBatchOperatorPromptsList::SetEUWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EUWidth[= value!]

Column Properties	
Property	**Syntax**
EventIDFilter Sets the filter for the Event ID column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEventIDFilter (); **void** CBatchOperatorPromptsList::SetEventIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDFilter [= *text$*]
EventIDHeaderText Specifies the header text for the Event ID column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEventIDHeaderText (); **void** CBatchOperatorPromptsList::SetEventIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDHeaderText [= *text$*]
EventIDVisible Sets whether or not to display the Event ID column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEventIDVisible(); **void** CBatchOperatorPromptsList::SetEventIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDVisible[= *boolvalue*]
EventIDWidth Sets the width of the Event ID column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetEventIDWidth(); **void** CBatchOperatorPromptsList::SetEventIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDWidth[= value!]
EventTypeFilter Sets the filter for the Event Type column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEventTypeFilter (); **void** CBatchOperatorPromptsList::SetEventTypeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeFilter [= *text$*]

Column Properties	
Property	**Syntax**
EventTypeHeaderText Specifies the header text for the Event Type column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetEventTypeHeaderText (); **void** CBatchOperatorPromptsList::SetEventTypeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeHeaderText [= *text$*]
EventTypeVisible Sets whether or not to display Event Type column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEventTypeVisible(); **void** CBatchOperatorPromptsList::SetEventTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeVisible[= *boolvalue*]
EventTypeWidth Sets the width of the Event Type column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetEventTypeWidth(); **void** CBatchOperatorPromptsList::SetEventTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeWidth[= value!]
HighFilter Sets the filter for the High Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetHighFilter (); **void** CBatchOperatorPromptsList::SetHighFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.HighFilter [= *text$*]
HighHeaderText Specifies the header text for the High Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetHighHeaderText (); **void** CBatchOperatorPromptsList::SetHighHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.HighHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
HighVisible Sets whether or not to display the High Value column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetHighVisible(); **void** CBatchOperatorPromptsList::SetHighVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.HighVisible[= *boolvalue*]
HighWidth Sets the width of the High Value column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetHighWidth(); **void** CBatchOperatorPromptsList::SetHighWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.HighWidth[= value!]
LowFilter Sets the filter for the Low Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetLowFilter (); **void** CBatchOperatorPromptsList::SetLowFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.LowFilter [= *text$*]
LowHeaderText Specifies the header text for the Low Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetLowHeaderText (); **void** CBatchOperatorPromptsList::SetLowHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.LowHeaderText [= *text$*]
LowVisible Sets whether or not to display the Low Value column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetLowVisible(); **void** CBatchOperatorPromptsList::SetLowVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.LowVisible[= *boolvalue*]
LowWidth Sets the width of the Low Value column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetLowWidth(); **void** CBatchOperatorPromptsList::SetLowWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.LowWidth[= value!]

Column Properties	
Property	**Syntax**
PhaseFilter Sets the filter for the Phase column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetPhaseFilter (); **void** CBatchOperatorPromptsList::SetPhaseFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.PhaseFilter [= *text$*]
PhaseHeaderText Specifies the header text for the Phase column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetPhaseHeaderText (); **void** CBatchOperatorPromptsList::SetPhaseHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.PhaseHeaderText [= *text$*]
PhaseVisible Sets whether or not to display the Phase column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetPhaseVisible(); **void** CBatchOperatorPromptsList::SetPhaseVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.PhaseVisible[= *boolvalue*]
PhaseWidth Sets the width of the Phase column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetPhaseWidth(); **void** CBatchOperatorPromptsList::SetPhaseWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.PhaseWidth[= value!]
ProcCellFilter Sets the filter for the Process Cell column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetProcCellFilter (); **void** CBatchOperatorPromptsList::SetProcCellFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ProcCellFilter [= *text$*]

Column Properties	
Property	**Syntax**
ProcCellHeaderText Specifies the header text for the Process Cell column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetProcCellHeaderText (); **void** CBatchOperatorPromptsList::SetProcCellHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ProcCellHeaderText [= *text$*]
ProcCellVisible Sets whether or not to display the Process Cell column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetProcCellVisible(); **void** CBatchOperatorPromptsList::SetProcCellVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ProcCellVisible[= *boolvalue*]
ProcCellWidth Sets the width of the Process Cell column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetProcCellWidth(); **void** CBatchOperatorPromptsList::SetProcCellWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.ProcCellWidth[= value!]
RecipeFilter Sets the filter for the Recipe column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetRecipeFilter (); **void** CBatchOperatorPromptsList::SetRecipeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeFilter [= *text$*]
RecipeHeaderText Specifies the header text for the Recipe Header column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetRecipeHeaderText (); **void** CBatchOperatorPromptsList::SetRecipeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
RecipeVisible Sets whether or not to display the Recipe column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetRecipeVisible(); **void** CBatchOperatorPromptsList::SetRecipeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeVisible[= *boolvalue*]
RecipeWidth Sets the width of the Recipe column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetRecipeWidth(); **void** CBatchOperatorPromptsList::SetRecipeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RecipeWidth[= value!]
RespTypeFilter Sets the filter for the Response Type column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetRespTypeFilter (); **void** CBatchOperatorPromptsList::SetRespTypeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RespTypeFilter [= *text$*]
RespTypeHeaderText Specifies the header text for the Response Type column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetRespTypeHeaderText (); **void** CBatchOperatorPromptsList::SetRespTypeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.RespTypeHeaderText [= *text$*]
RespTypeVisible Sets whether or not to display the Response Type column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetRespTypeVisible(); **void** CBatchOperatorPromptsList::SetRespTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RespTypeVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
RespTypeWidth Sets the width of the Response Type column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetRespTypeWidth(); **void** CBatchOperatorPromptsList::SetRespTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.RespTypeWidth[= value!]
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetRowNumbersVisible(); **void** CBatchOperatorPromptsList::SetRowNumbersVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RowNumbersVisible[= *boolvalue*]
TimeFilter Sets the filter for the Time column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetTimeFilter (); **void** CBatchOperatorPromptsList::SetTimeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeFilter [= *text$*]
TimeHeaderText Specifies the header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetTimeHeaderText (); **void** CBatchOperatorPromptsList::SetTimeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeHeaderText [= *text$*]
TimeVisible Sets whether or not to display the Time column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetTimeVisible(); **void** CBatchOperatorPromptsList::SetTimeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeVisible[= *boolvalue*]
TimeWidth Sets the width of the Time column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetTimeWidth(); **void** CBatchOperatorPromptsList::SetTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.TimeWidth[= value!]

Column Properties	
Property	**Syntax**
UnitFilter Sets the filter for the Unit column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetUnitFilter (); **void** CBatchOperatorPromptsList::SetUnitFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.UnitFilter [= *text$*]
UnitHeaderText Specifies the header text for the Unit column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetUnitHeaderText (); **void** CBatchOperatorPromptsList::SetUnitHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.UnitHeaderText [= *text$*]
UnitVisible Sets whether or not to display the Unit column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetUnitVisible(); **void** CBatchOperatorPromptsList::SetUnitVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.UnitVisible[= *boolvalue*]
UnitWidth Sets the width of the Unit column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetUnitWidth(); **void** CBatchOperatorPromptsList::SetUnitWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.UnitWidth[= value!]
ValueFilter Sets the filter for the Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetValueFilter (); **void** CBatchOperatorPromptsList::SetValueFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ValueFilter [= *text$*]
ValueHeaderText Specifies the header text for the Value column.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetValueHeaderText (); **void** CBatchOperatorPromptsList::SetValueHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ValueHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
ValueVisible Sets whether or not to display the Value column.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetValueVisible(); **void** CBatchOperatorPromptsList::SetValueVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueVisible[= *boolvalue*]
ValueWidth Sets the width of the Value column.	**C++ Syntax:** **double** CBatchOperatorPromptsList::GetValueWidth(); **void** CBatchOperatorPromptsList::SetValueWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueWidth[= value!]

BatchOperatorPromptsList Control VBIS Server Properties

The following table lists the VBIS Server properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetConnectAtStartup(); **void** CBatchOperatorPromptsList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]

BatchOperatorPromptsList Control VBIS Server Properties	
Property	**Syntax**
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchOperatorPromptsList ActiveX control is 5. ***NOTE:*** *For the BatchAdd and BatchRecipeList controls, the default Refresh Rate is 0, since the information in these lists changes less frequently. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually.*	**C++ Syntax:** **short** CBatchOperatorPromptsList::GetRefreshRate(); **void** CBatchOperatorPromptsList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchOperatorPromptsList::GetVBISServerName(); **void** CBatchOperatorPromptsList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

BatchOperatorPromptsList Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Miscellaneous Properties

Property	Syntax
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEnableRightMouseMenu(); **void** CBatchOperatorPromptsList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]
MouseDblClickedEnabled Sets whether or not to enable double-click access to commands. When enabled, double-clicking displays a list of valid commands.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetMouseDblClickedEnabled(); **void** CBatchOperatorPromptsList::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.MouseDblClickedEnabled[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetStatusBarEnabled(); **void** CBatchOperatorPromptsList::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StatusBarEnabled[= *boolvalue]*
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetToolBarEnabled(); **void** CBatchOperatorPromptsList::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarEnabled[= *boolvalue]*
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control. 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchOperatorPromptsList::GetToolBarPosition(); **void** CBatchOperatorPromptsList::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarPosition[= *value%*]

BatchOperatorPromptsList Control Miscellaneous Properties	
Property	**Syntax**
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when the operator executes a command.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetVerifyCommandActions(); **void** CBatchOperatorPromptsList::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VerifyCommandActions[= *boolvalue*]

BatchOperatorPromptsList Control Security Properties

The following table lists the security properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetColumnEditEnabled(); **void** CBatchOperatorPromptsList::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ColumnEditEnabled[= *boolvalue]*
CommandBtnsEditEnabled Sets whether or not the operator can edit the command buttons that are visible.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetCommandBtnsEditEnabled(); **void** CBatchOperatorPromptsList::SetCommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CommandBtnsEditEnabled[= *boolvalue]*

BatchOperatorPromptsList Control Security Properties	
Property	**Syntax**
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetEnableIFIXSecurity(); **void** CBatchOperatorPromptsList::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue]*
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetMiscEditEnabled(); **void** CBatchOperatorPromptsList::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetRefreshRateEditEnabled(); **void** CBatchOperatorPromptsList::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= *boolvalue*]
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetServerEditEnabled(); **void** CBatchOperatorPromptsList::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]

BatchOperatorPromptsList Control Security Properties	
Property	**Syntax**
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetSortOrderEditEnabled(); **void** CBatchOperatorPromptsList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SortOrderEditEnabled[= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetToggleConnectionEnabled(); **void** CBatchOperatorPromptsList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToggleConnectionEnabled[= *boolvalue*]

BatchOperatorPromptsList Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements Sets a default signature type (None, Performed By, Performed By/Verified By) to all of the commands for this ActiveX control. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetUseDefaultSignatureRequirements();**void** CBatchOperatorPromptsList::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.UseDefaultSignatureRequirements[= *boolvalue*]

BatchOperatorPromptsList Control Command Buttons Properties

The following table lists the security properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Command Buttons Properties	
Property	**Syntax**
AcknowledgePromptButton Sets whether or not to display the Acknowledge Prompts button.	**C++ Syntax:** **BOOL** CBatchOperatorPromptsList::GetAcknowledgePromptButton (); **void** CBatchOperatorPromptsList::SetAcknowledgePromptButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AcknowledgePromptButton[= *boolvalue*]

BatchOperatorPromptsList Control Color Properties

The following table lists the color properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetBackColor(); **void** CBatchOperatorPromptsList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color%*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetEvenRowBackColor(); **void** CBatchOperatorPromptsList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]

BatchOperatorPromptsList Control Color Properties	
Property	**Syntax**
EvenRowTextColor Sets the color of text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetEvenRowTextColor(); **void** CBatchOperatorPromptsList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetGridColor(); **void** CBatchOperatorPromptsList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetHeaderBackColor(); **void** CBatchOperatorPromptsList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color%*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetHeaderTextColor(); **void** CBatchOperatorPromptsList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetOddRowBackColor(); **void** CBatchOperatorPromptsList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]

BatchOperatorPromptsList Control Color Properties	
Property	**Syntax**
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchOperatorPromptsList::GetOddRowTextColor(); **void** CBatchOperatorPromptsList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

NOTE: *For examples on setting Color properties, refer to the BatchList ActiveX Control section.*

BatchOperatorPromptsList Control Font Properties

The following table lists the font properties for the BatchOperatorPromptsList control.

BatchOperatorPromptsList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchOperatorPromptsList::GetHeaderFont(); **void** CBatchOperatorPromptsList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderFont[= *StdFontVariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchOperatorPromptsList::GetTextFont(); **void** CBatchOperatorPromptsList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *StdFontVariable*]

NOTE: *For examples on setting Font properties, refer to the BatchList ActiveX Control section.*

BatchOperatorPromptsList Control Methods

The BatchOperatorPromptsList control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- GetIVBIS Method

- SetIVBISPointer Method
- SelectRowByID Method
- SelectRowByRowNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method
- RunCommandSelectedRows Method
- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

These methods are the same for each control that contains them. Click on a method above to link to the BatchList ActiveX Control section that describes the common method using the BatchList ActiveX control as an example.

BatchOperatorPromptsList Control Events

The BatchOperatorPromptsList control generates the following events:

- CommandExecuted Event
- ConnectedToServer Event
- DblClickList Event
- DisconnectedFromServer Event
- Refresh Event
- RowActivated Event
- ServerChanged Event
- RefreshEx Event

These events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

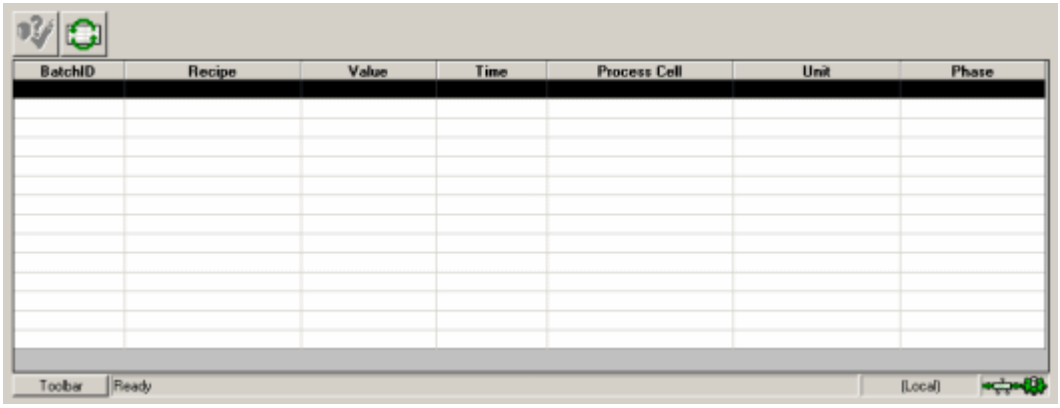
BatchBindingPromptsList ActiveX Control

The "Intellution BatchBindingPromptsList Control" displays a list of binding prompts. Operators can acknowledge these prompts using this control. The following figure shows the BatchBindingPromptsList control.

When binding prompts occur, the Display Binding Prompts button on the BatchList control blinks. To acknowledge a prompt, operators click the Display Prompts button to display the

BatchBindingPromptsList control. From this control, the operator can then acknowledge awaiting prompts and the batch can continue executing.

Using the control's property pages, designers can configure the control's GUI appearance and functionality. For example, the designer can configure the colors, fonts, and column widths. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.



Intellution BatchBindingPromptsList ActiveX Control

Developers can access the BatchBindingPromptsList control programmatically through Visual Basic or Visual C++. The properties, methods, and events for the BatchBindingPromptsList control are described in the following sections.

BatchBindingPromptsList Control Properties

The following sections describe each property for the BatchBindingPromptsList control. The properties are grouped into the following categories:

- Column properties
- VBIS Server properties
- Miscellaneous properties
- Security properties
- Electronic Signature properties
- Command Properties
- Color properties
- Font properties

BatchBindingPromptsList Control Column Properties

The following table lists the properties that control the display of the columns in the BatchBindingPromptsList control.

Column Properties	
Property	**Syntax**
AreaFilter Sets the filter for the Area column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetAreaFilter (); **void** CBatchBindingPromptsList::SetAreaFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AreaFilter [= *text$*]
AreaHeaderText Specifies the header text for the Area column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetAreaHeaderText (); **void** CBatchBindingPromptsList::SetAreaHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*AreaHeaderText [= *text$*]
AreaVisible Sets whether or not to display the Area column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetAreaVisible(); **void** CBatchBindingPromptsList::SetAreaVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AreaVisible[= *boolvalue*]
AreaWidth Sets the width of the Area column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetAreaWidth(); **void** CBatchBindingPromptsList::SetAreaWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*AreaWidth[= *value!*]
BatchIDVisible Sets whether or not to display Batch ID column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetBatch IDVisible(); **void** CBatchBindingPromptsList::SetBatch IDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*Batch IDVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
BatchIDFilter Sets the filter for the Batch ID column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetBatchIDFilter (); **void** CBatchBindingPromptsList::SetBatchIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDFilter [= *text$*]
BatchIDHeaderText Specifies the header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetBatchIDHeaderText (); **void** CBatchBindingPromptsList::SetBatchIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDHeaderText [= *text$*]
BatchIDWidth Sets the width of the Batch ID column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetBatchIDWidth(); **void** CBatchBindingPromptsList::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDWidth[= *value!*]
DefaultFilter Sets the filter for the Default Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetDefaultFilter (); **void** CBatchBindingPromptsList::SetDefaultFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultFilter [= *text$*]
DefaultHeaderText Specifies the header text for the Default Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetDefaultHeaderText (); **void** CBatchBindingPromptsList::SetDefaultHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.DefaultHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
DefaultVisible Sets whether or not to display the Default Value column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetDefaultVisible(); **void** CBatchBindingPromptsList::SetDefaultVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultVisible[= *boolvalue*]
DefaultWidth Sets the width of the Default Value column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetDefaultWidth(); **void** CBatchBindingPromptsList::SetDefaultWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*DefaultWidth[= *value!*]
EquipmentDescriptionFilter Sets the filter for the Equipment Description column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEquipmentDescriptionFilter (); **void** CBatchBindingPromptsList::SetEquipmentDescriptionFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*EquipmentDescriptionFilter [= *text$*]
EquipmentDescriptionHeaderText Specifies the header text for the Equipment Description column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEquipmentDescriptionHeaderText (); **void** CBatchBindingPromptsList::SetEquipmentDescriptionHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*EquipmentDescriptionHeaderText [= *text$*]
EquipmentDescriptionVisible Sets whether or not to display the Equipment Description column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEquipmentDescriptionVisible(); **void** CBatchBindingPromptsList::SetEquipmentDescriptionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*EquipmentDescriptionVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
EquipmentDescriptionWidth Sets the width of the Equipment Description column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetEquipmentDescriptionWidth(); **void** CBatchBindingPromptsList::SetEquipmentDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EquipmentDescriptionWidth[= *value!*]
EUFilter Sets the filter for the EU (Engineering Units) column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEUFilter (); **void** CBatchBindingPromptsList::SetEUFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EUFilter [= *text$*]
EUHeaderText Specifies the header text for the EU (Engineering Units) column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEUHeaderText (); **void** CBatchBindingPromptsList::SetEUHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EUHeaderText [= *text$*]
EUVisible Sets whether or not to display the EU (Engineering Units) column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEUVisible(); **void** CBatchBindingPromptsList::SetEUVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EUVisible[= *boolvalue*]
EUWidth Sets the width of the EU (Engineering Units) column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetEUWidth(); **void** CBatchBindingPromptsList::SetEUWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EUWidth[= *value!*]

Column Properties	
Property	**Syntax**
EventIDFilter Sets the filter for the Event ID column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEventIDFilter (); **void** CBatchBindingPromptsList::SetEventIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDFilter [= *text$*]
EventIDHeaderText Specifies the header text for the Event ID column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEventIDHeaderText (); **void** CBatchBindingPromptsList::SetEventIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDHeaderText [= *text$*]
EventIDVisible Sets whether or not to display the Event ID column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEventIDVisible(); **void** CBatchBindingPromptsList::SetEventIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDVisible[= *boolvalue*]
EventIDWidth Sets the width of the Event ID column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetEventIDWidth(); **void** CBatchBindingPromptsList::SetEventIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EventIDWidth[= *value!*]
EventTypeFilter Sets the filter for the Event Type column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEventTypeFilter (); **void** CBatchBindingPromptsList::SetEventTypeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeFilter [= *text$*]

Column Properties	
Property	**Syntax**
EventTypeHeaderText Specifies the header text for the Event Type column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetEventTypeHeaderText (); **void** CBatchBindingPromptsList::SetEventTypeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeHeaderText [= *text$*]
EventTypeVisible Sets whether or not to display the Event Type column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEventTypeVisible(); **void** CBatchBindingPromptsList::SetEventTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeVisible[= *boolvalue*]
EventTypeWidth Sets the width of the Event Type column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetEventTypeWidth(); **void** CBatchBindingPromptsList::SetEventTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.EventTypeWidth[= *value!*]
HighFilter Sets the filter for the High Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetHighFilter (); **void** CBatchBindingPromptsList::SetHighFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.HighFilter [= *text$*]
HighHeaderText Specifies the header text for the High Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetHighHeaderText (); **void** CBatchBindingPromptsList::SetHighHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.HighHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
HighVisible Sets whether or not to display the High Value column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetHighVisible(); **void** CBatchBindingPromptsList::SetHighVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*HighVisible[= *boolvalue*]
HighWidth Sets the width of the High Value column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetHighWidth(); **void** CBatchBindingPromptsList::SetHighWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*HighWidth[= *value!*]
LowFilter Sets the filter for the Low Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetLowFilter (); **void** CBatchBindingPromptsList::SetLowFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*LowFilter [= *text$*]
LowHeaderText Specifies the header text for the Low Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetLowHeaderText (); **void** CBatchBindingPromptsList::SetLowHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*LowHeaderText [= *text$*]
LowVisible Sets whether or not to display the Low Value column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetLowVisible(); **void** CBatchBindingPromptsList::SetLowVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*LowVisible[= *boolvalue*]
LowWidth Sets the width of the Low Value column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetLowWidth(); **void** CBatchBindingPromptsList::SetLowWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*LowWidth[= *value!*]

Column Properties	
Property	**Syntax**
PhaseFilter Sets the filter for the Phase column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetPhaseFilter (); **void** CBatchBindingPromptsList::SetPhaseFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseFilter [= *text$*]
PhaseHeaderText Specifies the header text for the Phase column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetPhaseHeaderText (); **void** CBatchBindingPromptsList::SetPhaseHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseHeaderText [= *text$*]
PhaseVisible Sets whether or not to display the Phase column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetPhaseVisible(); **void** CBatchBindingPromptsList::SetPhaseVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseVisible[= *boolvalue*]
PhaseWidth Sets the width of the Phase column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetPhaseWidth(); **void** CBatchBindingPromptsList::SetPhaseWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseWidth[= *value!*]
ProcCellFilter Sets the filter for the Process Cell column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetProcCellFilter (); **void** CBatchBindingPromptsList::SetProcCellFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcCellFilter [= *text$*]

Column Properties	
Property	**Syntax**
ProcCellHeaderText Specifies the header text for the Process Cell column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetProcCellHeaderText (); **void** CBatchBindingPromptsList::SetProcCellHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcCellHeaderText [= *text$*]
ProcCellVisible Sets whether or not to display the Process Cell column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetProcCellVisible(); **void** CBatchBindingPromptsList::SetProcCellVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcCellVisible[= *boolvalue*]
ProcCellWidth Sets the width of the Process Cell column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetProcCellWidth(); **void** CBatchBindingPromptsList::SetProcCellWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ProcCellWidth[= *value!*]
RecipeFilter Sets the filter for the Recipe column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetRecipeFilter (); **void** CBatchBindingPromptsList::SetRecipeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilter [= *text$*]
RecipeHeaderText Specifies the columns header text for the Recipe Header column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetRecipeHeaderText (); **void** CBatchBindingPromptsList::SetRecipeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
RecipeVisible Sets whether or not to display the Recipe column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetRecipeVisible(); **void** CBatchBindingPromptsList::SetRecipeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVisible[= *boolvalue*]
RecipeWidth Sets the width of the Recipe column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetRecipeWidth(); **void** CBatchBindingPromptsList::SetRecipeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeWidth[= *value!*]
RespTypeFilter Sets the filter for the Response Type column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetRespTypeFilter (); **void** CBatchBindingPromptsList::SetRespTypeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RespTypeFilter [= *text$*]
RespTypeHeaderText Specifies the header text for the Response Type column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetRespTypeHeaderText (); **void** CBatchBindingPromptsList::SetRespTypeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RespTypeHeaderText [= *text$*]
RespTypeVisible Sets whether or not to display the Response Type column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetRespTypeVisible(); **void** CBatchBindingPromptsList::SetRespTypeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RespTypeVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
RespTypeWidth Sets the width of the Response Type column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetRespTypeWidth(); **void** CBatchBindingPromptsList::SetRespTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RespTypeWidth[= *value!*]
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetRowNumbersVisible(); **void** CBatchBindingPromptsList::SetRowNumbersVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RowNumbersVisible[= *boolvalue]*
TimeFilter Sets the filter for the Time column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetTimeFilter (); **void** CBatchBindingPromptsList::SetTimeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeFilter [= *text$*]
TimeHeaderText Specifies the header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetTimeHeaderText (); **void** CBatchBindingPromptsList::SetTimeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeHeaderText [= *text$*]
TimeVisible Sets whether or not to display the Time column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetTimeVisible(); **void** CBatchBindingPromptsList::SetTimeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeVisible[= *boolvalue*]
TimeWidth Sets the width of the Time column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetTimeWidth(); **void** CBatchBindingPromptsList::SetTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*TimeWidth[= *value!*]

Column Properties	
Property	**Syntax**
UnitFilter Sets the filter for the Unit column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetUnitFilter (); **void** CBatchBindingPromptsList::SetUnitFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitFilter [= *text$*]
UnitHeaderText Specifies the header text for the Unit column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetUnitHeaderText (); **void** CBatchBindingPromptsList::SetUnitHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitHeaderText [= *text$*]
UnitVisible Sets whether or not to display the Unit column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetUnitVisible(); **void** CBatchBindingPromptsList::SetUnitVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitVisible[= *boolvalue*]
UnitWidth Sets the width of the Unit column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetUnitWidth(); **void** CBatchBindingPromptsList::SetUnitWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitWidth[= *value!*]
ValueFilter Sets the filter for the Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetValueFilter (); **void** CBatchBindingPromptsList::SetValueFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueFilter [= *text$*]
ValueHeaderText Specifies the header text for the Value column.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetValueHeaderText (); **void** CBatchBindingPromptsList::SetValueHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueHeaderText [= *text$*]

Column Properties	
Property	**Syntax**
ValueVisible Sets whether or not to display the Value column.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetValueVisible(); **void** CBatchBindingPromptsList::SetValueVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueVisible[= *boolvalue*]
ValueWidth Sets the width of the Value column.	**C++ Syntax:** **double** CBatchBindingPromptsList::GetValueWidth(); **void** CBatchBindingPromptsList::SetValueWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ValueWidth[= *value!*]

BatchBindingPromptsList Control VBIS Server Properties

The following table lists the VBIS Server properties for the BatchBindingPromptsList control.

BatchBindingPromptsList Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetConnectAtStartup(); **void** CBatchBindingPromptsList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]

BatchBindingPromptsList Control VBIS Server Properties	
Property	**Syntax**
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchBindingPromptsList ActiveX control is 5. ***NOTE:*** *For the BatchAdd and BatchRecipeList controls, the default Refresh Rate is 0, since the information in these lists changes less frequently. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually.*	**C++ Syntax:** **short** CBatchBindingPromptsList::GetRefreshRate(); **void** CBatchBindingPromptsList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchBindingPromptsList::GetVBISServerName(); **void** CBatchBindingPromptsList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

BatchBindingPromptsList Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchBindingPromptsList control.

BatchBindingPromptsList Control Miscellaneous Properties

Property	Syntax
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEnableRightMouseMenu(); **void** CBatchBindingPromptsList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]
MouseDblClickedEnabled Sets whether or not to enable double-click access to commands. When enabled, double-clicking displays a list of valid commands.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetMouseDblClickedEnabled(); **void** CBatchBindingPromptsList::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.MouseDblClickedEnabled[= *boolvalue]*
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetStatusBarEnabled(); **void** CBatchBindingPromptsList::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StatusBarEnabled[= *boolvalue]*
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetToolBarEnabled(); **void** CBatchBindingPromptsList::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarEnabled[= *boolvalue]*
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control. 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchBindingPromptsList::GetToolBarPosition(); **void** CBatchBindingPromptsList::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToolBarPosition[= *value%*]

BatchBindingPromptsList Control Miscellaneous Properties	
Property	**Syntax**
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when the operator executes a command.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetVerifyCommandActions(); **void** CBatchBindingPromptsList::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VerifyCommandActions[= *boolvalue*]

BatchBindingPromptsList Control Security Properties

The following table lists the security properties for the BatchBindingPromptsList control.

BatchBindingPromptsList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetColumnEditEnabled(); **void** CBatchBindingPromptsList::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ColumnEditEnabled[= *boolvalue]*
CommandBtnsEditEnabled Sets whether or not the operator can edit the command buttons that are visible.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetCommandBtnsEditEnabled(); **void** CBatchBindingPromptsList::SetCommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CommandBtnsEditEnabled[= *boolvalue]*

BatchBindingPromptsList Control Security Properties	
Property	**Syntax**
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetEnableIFIXSecurity(); **void** CBatchBindingPromptsList::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue]*
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetMiscEditEnabled(); **void** CBatchBindingPromptsList::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue]*
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetRefreshRateEditEnabled(); **void** CBatchBindingPromptsList::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= *boolvalue]*
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetServerEditEnabled(); **void** CBatchBindingPromptsList::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]

BatchBindingPromptsList Control Security Properties	
Property	**Syntax**
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetSortOrderEditEnabled(); **void** CBatchBindingPromptsList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*SortOrderEditEnabled[= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetToggleConnectionEnabled(); **void** CBatchBindingPromptsList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= *boolvalue]*

BatchBindingPromptsList Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchBindingPromptsList control.

BatchBindingPromptsList Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements Sets a default signature type (None, Performed By, Performed By/Verified By) to all of the commands for this ActiveX control. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchBindingPromptsList::GetUseDefaultSignatureRequirements();**void** CBatchBindingPromptsList::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

BatchBindingPromptsList Control Command Properties

The following table lists the security properties for the BatchBindingPromptsList control

BatchBindingPromptsList Control Security Properties	
Property	**Syntax**
AcknowledgePromptButton Sets whether or not to display the Acknowledge Prompts button.	**C++ Syntax:** **BOOL** CBatchList::GetAcknowledgePromptButton (); **void** CBatchList::SetAcknowledgePromptButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AcknowledgePromptButton[= *boolvalue*]

BatchBindingPromptsList Control Color Properties

The following table lists the color properties for the BatchBindingPromptsList control.

BatchBindingPromptsList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetBackColor(); **void** CBatchBindingPromptsList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color%*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetEvenRowBackColor(); **void** CBatchBindingPromptsList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]

BatchBindingPromptsList Control Color Properties	
Property	**Syntax**
EvenRowTextColor Sets the color of text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetEvenRowTextColor(); **void** CBatchBindingPromptsList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetGridColor(); **void** CBatchBindingPromptsList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetHeaderBackColor(); **void** CBatchBindingPromptsList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color%*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetHeaderTextColor(); **void** CBatchBindingPromptsList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetOddRowBackColor(); **void** CBatchBindingPromptsList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]

BatchBindingPromptsList Control Color Properties	
Property	**Syntax**
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchBindingPromptsList::GetOddRowTextColor(); **void** CBatchBindingPromptsList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

NOTE: *For examples on setting Color properties, refer to the BatchList ActiveX Control section.*

BatchBindingPromptsList Control Font Properties

The following table lists the font properties for the BatchBindingPromptsList control.

BatchBindingPromptsList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchBindingPromptsList::GetHeaderFont(); **void** CBatchBindingPromptsList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderFont[= *StdFontVariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchBindingPromptsList::GetTextFont(); **void** CBatchBindingPromptsList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *StdFontVariable*]

NOTE: *For examples on setting Font properties, refer to the BatchList ActiveX Control section.*

BatchBindingPromptsList Control Methods

The BatchBindingPromptsList control supports the following methods:

- ConnectToServer Method

- DisconnectFromServer Method
- GetIVBIS Method
- SelectRowByID Method
- SelectRowByRowNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method
- RunCommandSelectedRows Method
- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

These methods are the same for each control that contains them. Click on a method above to link to the BatchList ActiveX Control section that describes the common method using the BatchList ActiveX control as an example.

BatchBindingPromptsList Control Events

The BatchBindingPromptsList control generates the following events:

- CommandExecuted Event
- ConnectedToServer Event
- DblClickList Event
- DisconnectedFromServer Event
- Refresh Event
- RowActivated Event
- ServerChanged Event
- RefreshEx Event

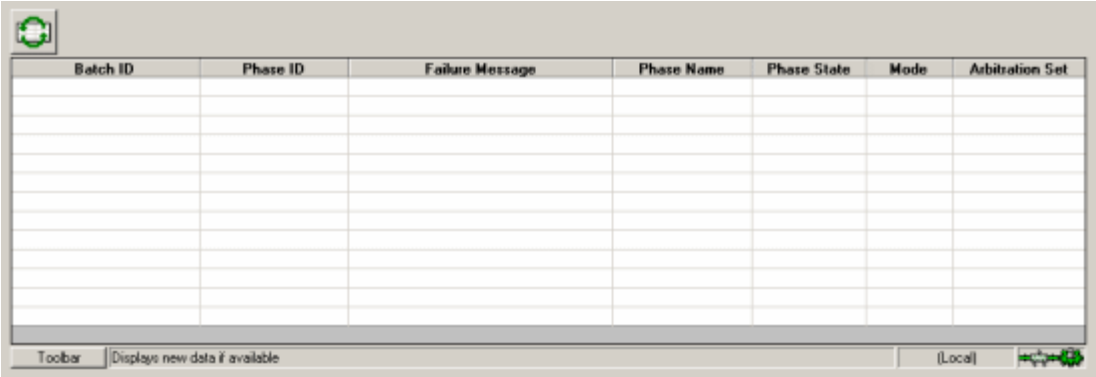
These events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

BatchAlarmList ActiveX Control

The "Intellution BatchAlarmList Control" displays a list of alarms. This control is a component of the BatchList ActiveX control. The following figure shows the BatchAlarmList control.

Using the control's property pages, designers can configure the control's GUI appearance and

functionality for operators. For example, the designer can configure the colors, fonts, and column widths for the alarm list. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.



Intellution BatchAlarmList ActiveX Control

Developers can access the Intellution BatchAlarmList control programmatically through Visual Basic or Visual C++. The properties, methods, and events for the BatchAlarmList control are described in the following sections.

BatchAlarmList Control Properties

The following sections describe each property for the BatchAlarmList control. The properties are grouped into the following categories:

- Column properties
- VBIS Server properties
- Miscellaneous
- Security properties
- Color properties
- Font properties

BatchAlarmList Control Column Properties

The following table lists the properties that control the display of the columns in the BatchAlarmList control.

Column Properties

Property	Syntax
ArbitrationSetFilter Sets the filter for the Arbitration Set column.	**C++ Syntax:** **CString** CBatchAlarmList::GetArbitrationSetFilter(); **void** CBatchAlarmList::SetArbitrationSetFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ArbitrationSetFilter[= *text$*]
ArbitrationSetHeaderText Specifies the column header text for the Arbitration Set column.	**C++ Syntax:** **CString** CBatchAlarmList::GetArbitrationSetHeaderText(); **void** CBatchAlarmList::SetArbitrationSetHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ArbitrationSetHeaderText[= *text$*]
ArbitrationSetVisible Sets whether or not to display the Arbitration Set column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetArbitrationSetVisible(); **void** CBatchAlarmList::SetArbitrationSetVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ArbitrationSetVisible[= *boolvalue*]
ArbitrationSetWidth Sets the width for the Arbitration Set column.	**C++ Syntax:** **double** CBatchAlarmList::GetArbitrationSetWidth(); **void** CBatchAlarmList::SetArbitrationSetWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ArbitrationSetWidth[= *value!*]
BatchIDFilter Sets the filter for the Batch ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetBatchIDFilter(); **void** CBatchAlarmList::SetBatchIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDFilter[= *text$*]

Column Properties	
Property	**Syntax**
BatchIDHeaderText Specifies the column header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetBatchIDHeaderText(); **void** CBatchAlarmList::SetBatchIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDHeaderText[= *text$*]
BatchIDVisible Sets whether or not to display the Batch ID column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetBatchIDVisible(); **void** CBatchAlarmList::SetBatchIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDVisible[= *boolvalue*]
BatchIDWidth Sets the width for the Batch ID column.	**C++ Syntax:** **double** CBatchAlarmList::GetBatchIDWidth(); **void** CBatchAlarmList::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDWidth[= *value!*]
FailMsgFilter Sets the filter for the Failure Message column.	**C++ Syntax:** **CString** CBatchAlarmList::GetFailMsgFilter(); **void** CBatchAlarmList::SetFailMsgFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailMsgFilter[= *text$*]
FailMsgHeaderText Specifies the column header text for the Failure Message column.	**C++ Syntax:** **CString** CBatchAlarmList::GetFailMsgHeaderText(); **void** CBatchAlarmList::SetFailMsgHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailMsgHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
FailMsgVisible Sets whether or not to display the Failure Message column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetFailMsgVisible(); **void** CBatchAlarmList::SetFailMsgVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*FailMsgVisible[= *boolvalue*]
FailMsgWidth Sets the width for the Failure Message column.	**C++ Syntax:** **double** CBatchAlarmList::GetFailMsgWidth(); **void** CBatchAlarmList::SetFailMsgWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*FailMsgWidth[= *value!*]
ModeFilter Sets the filter for the Mode column.	**C++ Syntax:** **CString** CBatchAlarmList::GetModeFilter(); **void** CBatchAlarmList::SetModeFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeFilter[= *text$*]
ModeHeaderText Specifies the column header text for the Mode column.	**C++ Syntax:** **CString** CBatchAlarmList::GetModeHeaderText(); **void** CBatchAlarmList::SetModeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeHeaderText[= *text$*]
ModeVisible Sets whether or not to display the Mode column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetModeVisible(); **void** CBatchAlarmList::SetModeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeVisible[= *boolvalue*]
ModeWidth Sets the width for the Mode column.	**C++ Syntax:** **double** CBatchAlarmList::GetModeWidth(); **void** CBatchAlarmList::SetModeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ModeWidth[= *value!*]

Column Properties	
Property	**Syntax**
OwnerFilter Sets the filter for the Owner column.	**C++ Syntax:** **CString** CBatchAlarmList::GetOwnerFilter(); **void** CBatchAlarmList::SetOwnerFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerFilter[= *text$*]
OwnerHeaderText Specifies the column header text for the Owner column.	**C++ Syntax:** **CString** CBatchAlarmList::GetOwnerHeaderText(); **void** CBatchAlarmList::SetOwnerHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerHeaderText[= *text$*]
OwnerVisible Sets whether or not to display the Owner column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetOwnerVisible(); **void** CBatchAlarmList::SetOwnerVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerVisible[= *boolvalue*]
OwnerWidth Sets the width for the Owner column.	**C++ Syntax:** **double** CBatchAlarmList::GetOwnerWidth(); **void** CBatchAlarmList::SetOwnerWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerWidth[= *value!*]
PhaseIDFilter Sets the filter for the Phase ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseIDFilter(); **void** CBatchAlarmList::SetPhaseIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseIDFilter[= *text$*]
PhaseIDHeaderText Specifies the column header text for the Phase ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseIDHeaderText(); **void** CBatchAlarmList::SetPhaseIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseIDHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
PhaseIDVisible Sets whether or not to display the Phase ID column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetPhaseIDVisible(); **void** CBatchAlarmList::SetPhaseIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseIDVisible[= *boolvalue*]
PhaseIDWidth Sets the width for the Phase ID column.	**C++ Syntax:** **double** CBatchAlarmList::GetPhaseIDWidth(); **void** CBatchAlarmList::SetPhaseIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseIDWidth[= *value!*]
PhaseMsgFilter Sets the filter for the Phase Message column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseMsgFilter(); **void** CBatchAlarmList::SetPhaseMsgFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseMsgFilter[= *text$*]
PhaseMsgHeaderText Specifies the column header text for the Phase Message column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseMsgHeaderText(); **void** CBatchAlarmList::SetPhaseMsgHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseMsgHeaderText[= *text$*]
PhaseMsgVisible Sets whether or not to display the Phase Message column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetPhaseMsgVisible(); **void** CBatchAlarmList::SetPhaseMsgVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseMsgVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
PhaseMsgWidth Sets the width for the Phase Message column.	**C++ Syntax:** **double** CBatchAlarmList::GetPhaseMsgWidth(); **void** CBatchAlarmList::SetPhaseMsgWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseMsgWidth[= *value!*]
PhaseNameFilter Sets the filter for the Phase Name column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseNameFilter(); **void** CBatchAlarmList::SetPhaseNameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseNameFilter[= *text$*]
PhaseNameHeaderText Specifies the column header text for the Phase Name column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseNameHeaderText(); **void** CBatchAlarmList::SetPhaseNameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseNameHeaderText[= *text$*]
PhaseNameVisible Sets whether or not to display the Phase Name column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetPhaseNameVisible(); **void** CBatchAlarmList::SetPhaseNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseNameVisible[= *boolvalue*]
PhaseNameWidth Sets the width for the Phase Name column.	**C++ Syntax:** **double** CBatchAlarmList::GetPhaseNameWidth(); **void** CBatchAlarmList::SetPhaseNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseNameWidth[= *value!*]

Column Properties	
Property	**Syntax**
PhaseStateFilter Sets the filter for the Phase State column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseStateFilter(); **void** CBatchAlarmList::SetPhaseStateFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseStateFilter[= *text$*]
PhaseStateHeaderText Specifies the column header text for the Phase State column.	**C++ Syntax:** **CString** CBatchAlarmList::GetPhaseStateHeaderText(); **void** CBatchAlarmList::SetPhaseStateHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseStateHeaderText[= *text$*]
PhaseStateVisible Sets whether or not to display the Phase State column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetPhaseStateVisible(); **void** CBatchAlarmList::SetPhaseStateVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseStateVisible[= *boolvalue*]
PhaseStateWidth Sets the width for the Phase State column.	**C++ Syntax:** **double** CBatchAlarmList::GetPhaseStateWidth(); **void** CBatchAlarmList::SetPhaseStateWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseStateWidth[= *value!*]
RowNumbersVisible Sets whether or not to display row numbers on the data list.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetRowNumbersVisible(); **void** CBatchAlarmList::SetRowNumbersVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RowNumbersVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
UnitIDFilter Sets the filter for the Unit ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetUnitIDFilter(); **void** CBatchAlarmList::SetUnitIDFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitIDFilter[= *text$*]
UnitIDHeaderText Specifies the column header text for the Unit ID column.	**C++ Syntax:** **CString** CBatchAlarmList::GetUnitIDHeaderText(); **void** CBatchAlarmList::SetUnitIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitIDHeaderText[= *text$*]
UnitIDVisible Sets whether or not to display the Unit ID column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetUnitIDVisible(); **void** CBatchAlarmList::SetUnitIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitIDVisible[= *boolvalue*]
UnitIDWidth Sets the width for the Unit ID column.	**C++ Syntax:** **double** CBatchAlarmList::GetUnitIDWidth(); **void** CBatchAlarmList::SetUnitIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitIDWidth[= *value!*]
UnitNameFilter Sets the filter for the Unit Name column.	**C++ Syntax:** **CString** CBatchAlarmList::GetUnitNameFilter(); **void** CBatchAlarmList::SetUnitNameFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameFilter[= *text$*]
UnitNameHeaderText Specifies the column header text for the Unit Name column.	**C++ Syntax:** **CString** CBatchAlarmList::GetUnitNameHeaderText(); **void** CBatchAlarmList::SetUnitNameHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameHeaderText[= *text$*]

Column Properties	
Property	**Syntax**
UnitNameVisible Sets whether or not to display the Unit Name column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetUnitNameVisible(); **void** CBatchAlarmList::SetUnitNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameVisible[= *boolvalue*]
UnitNameWidth Sets the width for the Unit Name column.	**C++ Syntax:** **double** CBatchAlarmList::GetUnitNameWidth(); **void** CBatchAlarmList::SetUnitNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameWidth[= *value!*]
ValidUnitListFilter Sets the filter for the Valid Unit List column.	**C++ Syntax:** **CString** CBatchAlarmList::GetValidUnitListFilter(); **void** CBatchAlarmList::SetValidUnitListFilter(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ValidUnitListFilter[= *text$*]
ValidUnitListHeaderText Specifies the column header text for the Valid Unit List column.	**C++ Syntax:** **CString** CBatchAlarmList::GetValidUnitListHeaderText(); **void** CBatchAlarmList::SetValidUnitListHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ValidUnitListHeaderText[= *text$*]
ValidUnitListVisible Sets whether or not to display the Valid Unit List column.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetValidUnitListVisible(); **void** CBatchAlarmList::SetValidUnitListVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ValidUnitListVisible[= *boolvalue*]

Column Properties	
Property	**Syntax**
ValidUnitListWidth Sets the width for the Valid Unit List column.	**C++ Syntax:** **double** CBatchAlarmList::GetValidUnitListWidth(); **void** CBatchAlarmList::SetValidUnitListWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ValidUnitListWidth[= *value!*]

BatchAlarmList Control VBIS Server Properties

The following table lists the VBIS Server properties for the BatchAlarmList control.

BatchAlarmList Control VBIS Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetConnectAtStartup(); **void** CBatchAlarmList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchAlarmList ActiveX control is 5. ***NOTE:*** *For the BatchAdd and BatchRecipeList controls, the default Refresh Rate is 0, since the information in these lists changes less frequently. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually.*	**C++ Syntax:** **short** CBatchAlarmList::GetRefreshRate(); **void** CBatchAlarmList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]

BatchAlarmList Control VBIS Server Properties	
Property	**Syntax**
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchAlarmList::GetVBISServerName(); **void** CBatchAlarmList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

BatchAlarmList Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchAlarmList control.

BatchAlarmList Control Miscellaneous Properties	
Property	**Syntax**
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetEnableRightMouseMenu(); **void** CBatchAlarmList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableRightMouseMenu[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetStatusBarEnabled(); **void** CBatchAlarmList::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StatusBarEnabled[= *boolvalue*]
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetToolBarEnabled(); **void** CBatchAlarmList::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarEnabled[= *boolvalue*]

BatchAlarmList Control Miscellaneous Properties	
Property	**Syntax**
ToolBarPosition Sets the location of the toolbar: 0 displays the toolbar at the top of the control. 1 displays the toolbar at the bottom of the control.	**C++ Syntax:** **short** CBatchAlarmList::GetToolBarPosition(); **void** CBatchAlarmList::SetToolBarPosition(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarPosition[= *value%*]

BatchAlarmList Control Security Properties

The following table lists the security properties for the BatchAlarmList control.

BatchAlarmList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** Ccontrol::GetColumnEditEnabled(); **void** Ccontrol::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ColumnEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** Ccontrol::GetEnableIFIXSecurity(); **void** Ccontrol::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue*]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** Ccontrol::GetMiscEditEnabled(); **void** Ccontrol::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]

BatchAlarmList Control Security Properties	
Property	**Syntax**
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** Ccontrol::GetRefreshRateEditEnabled(); **void** Ccontrol::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RefreshRateEditEnabled[= *boolvalue*]
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** Ccontrol::GetServerEditEnabled(); **void** Ccontrol::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ServerEditEnabled [= *boolvalue*]
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetSortOrderEditEnabled(); **void** CBatchAlarmList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SortOrderEditEnabled[= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchAlarmList::GetToggleConnectionEnabled(); **void** CBatchAlarmList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToggleConnectionEnabled[= *boolvalue*]

BatchAlarmList Control Color Properties

The following table lists the color properties for the BatchAlarmList control.

BatchAlarmList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetBackColor(); **void** CBatchAlarmList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color%*]
EvenRowBackColor Sets the background color of even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetEvenRowBackColor(); **void** CBatchAlarmList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]
EvenRowTextColor Sets the color of text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetEvenRowTextColor(); **void** CBatchAlarmList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetGridColor(); **void** CBatchAlarmList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*GridColor[= *color%*]

BatchAlarmList Control Color Properties	
Property	**Syntax**
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetHeaderBackColor(); **void** CBatchAlarmList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderBackColor[= *color%*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetHeaderTextColor(); **void** CBatchAlarmList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetOddRowBackColor(); **void** CBatchAlarmList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowBackColor[= *color%*]
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchAlarmList::GetOddRowTextColor(); **void** CBatchAlarmList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

BatchAlarmList Control Font Properties

The following table lists the font properties for the BatchAlarmList control.

BatchAlarmList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchAlarmList::GetHeaderFont(); **void** CBatchAlarmList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderFont[= *StdFontVariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchAlarmList::GetTextFont(); **void** CBatchAlarmList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *StdFontVariable*]

BatchAlarmList Control Methods

The BatchAlarmList control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- GetIVBIS Method
- SetIVBISPointer Method
- SelectRowByID Method
- SelectRowByRowNumber Method
- GetNumberOfDataRows Method
- GetSelectedRowData Method
- SetColumnOrder Method
- SetSortKeys Method
- SwapColumns Method

These methods are the same for each control. Click on a method above to the BatchList ActiveX Control section that describes the common method using the BatchList ActiveX control as an example.

BatchAlarmList Control Events

The BatchAlarmList control generates the following events:

- ConnectedToServer Event
- DblClickList Event
- DisconnectedFromServer Event
- Refresh Event
- RowActivated Event
- ServerChanged Event
- RefreshEx Event

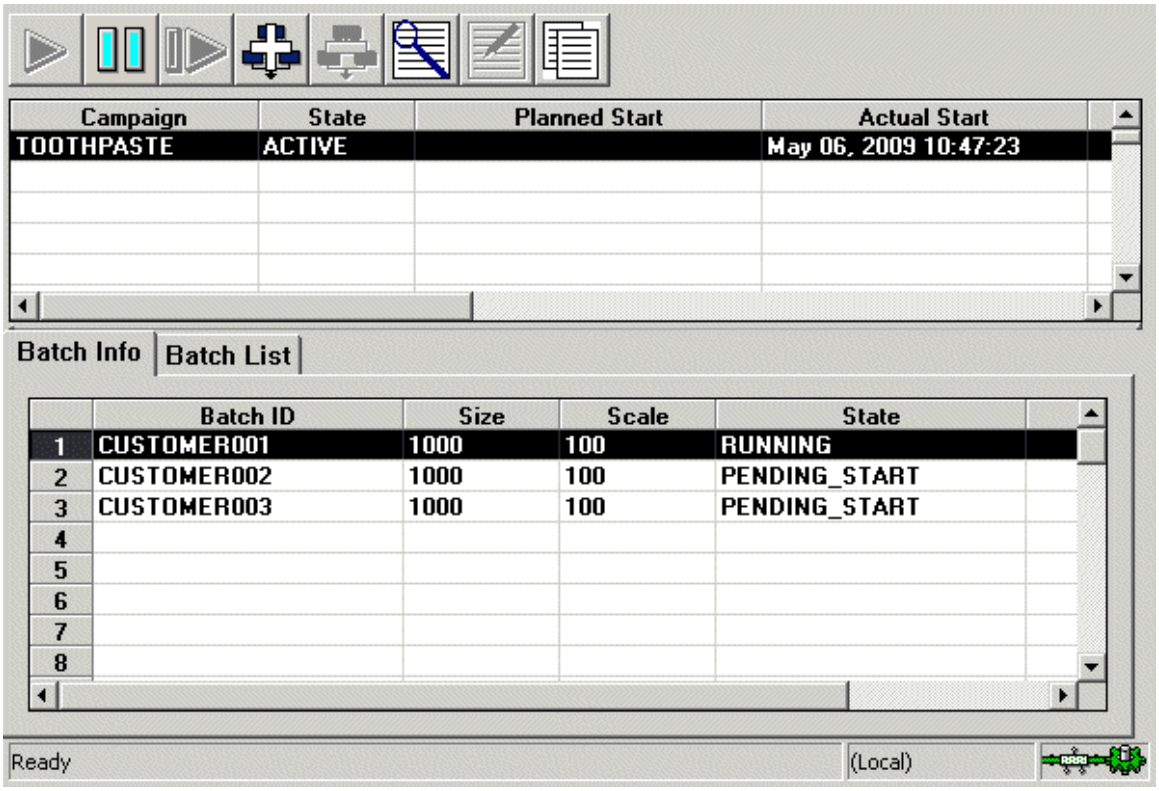
These events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

BatchCampaignClient ActiveX Control

The "Intellution BatchCampaignClient Control" allows you to run and control campaigns. Depending on which commands are enabled, operators can add a campaign, start a campaign and stop a campaign. They can also perform similar functions for batches within the campaign.

Designers can configure the BatchCampaignClient control's GUI run-time appearance and functionality using the control's property pages. For example, the designer can disable command buttons that the operator should not have access to. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.

The following figure shows the BatchCampaignClient control.



BatchCampaignClient ActiveX Control

Developers can access the control programmatically through Visual Basic or Visual C++ using the control's properties and methods. The properties, methods, and events for the BatchCampaignClient control are described in the following sections.

BatchCampaignClient Control Properties

The following sections describe each property for the BatchCampaignClient control. The properties are grouped into the following categories:

- BatchCampaignClient Control Column Properties
- Sort Order Properties
- BatchCampaignClient Control Campaign Server Properties
- BatchCampaignClient Control Command Buttons Properties
- BatchCampaignClient Control Miscellaneous Properties
- BatchCampaignClient Control Security Properties
- BatchCampaignClient Control Electronic Signature Properties
- BatchCampaignClient Control Color Properties
- BatchCampaignClient Control Fonts Properties

BatchCampaignClient Control Column Properties

The following table lists the properties that control the display of the columns in the BatchCampaignClient control.

BatchCampaignClient Control Column Properties	
Property	**Syntax**
CampaignFilter Sets the filter for the Campaign column.	**C++ Syntax:** **CString** CCampaignClient::GetCampaignFilter (); **void** CCampaignClient::SetCampaignFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*CampaignFilter [= *text$*]
CampaignHeaderText Specifies the header text for the Campaign column.	**C++ Syntax:** **CString** CCampaignClient::GetCampaignHeaderText (); **void** CCampaignClient::SetCampaignHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*CampaignHeaderText [= *text$*]
CampaignVisible Sets whether or not to display the Campaign column.	**C++ Syntax:** **BOOL** CCampaignClient::GetCampaignVisible(); **void** CCampaignClient::SetCampaignVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*CampaignVisible[= *boolvalue*]
CampaignWidth Sets the width of the Campaign column.	**C++ Syntax:** **double** CCampaignClient::GetCampaignWidth(); **void** CCampaignClient::SetCampaignWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*CampaignWidth[= *value!*]
StateVisible Sets whether or not to display State column.	**C++ Syntax:** **BOOL** CCampaignClient::GetStateVisible(); **void** CCampaignClient::SetStateVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StateVisible[= *boolvalue*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
StateFilter Sets the filter for the State column.	**C++ Syntax:** **CString** CCampaignClient::GetStateFilter (); **void** CCampaignClient::SetStateFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StateFilter [= *text$*]
StateHeaderText Specifies the header text for the State column.	**C++ Syntax:** **CString** CCampaignClient::GetStateHeaderText (); **void** CCampaignClient::SetStateHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StateHeaderText [= *text$*]
StateWidth Sets the width of the State column.	**C++ Syntax:** **double** CCampaignClient::GetStateWidth(); **void** CCampaignClient::SetStateWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StateWidth[= *value!*]
PlannedStartFilter Sets the filter for the Planned Start column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedStartFilter (); **void** CCampaignClient::SetPlannedStartFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedStartFilter [= *text$*]
PlannedStartHeaderText Specifies the header text for the Planned Start column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedStartHeaderText (); **void** CCampaignClient::SetPlannedStartHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedStartHeaderText [= *text$*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
PlannedStartVisible Sets whether or not to display the Planned Start column.	**C++ Syntax:** **BOOL** CCampaignClient::GetPlannedStartVisible(); **void** CCampaignClient::SetPlannedStartVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedStartVisible[= *boolvalue*]
PlannedStartWidth Sets the width of the Planned Start column.	**C++ Syntax:** **double** CCampaignClient::GetPlannedStartWidth(); **void** CCampaignClient::SetPlannedStartWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedStartWidth[= *value!*]
ActualStartFilter Sets the filter for the Actual Start column.	**C++ Syntax:** **CString** CCampaignClient::GetActualStartFilter (); **void** CCampaignClient::SetActualStartFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*tActualStartFilter [= *text$*]
ActualStartHeaderText Specifies the header text for the Actual Start column.	**C++ Syntax:** **CString** CCampaignClient::GetActualStartHeaderText (); **void** CCampaignClient::SetActualStartHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualStartHeaderText [= *text$*]
ActualStartVisible Sets whether or not to display the Actual Start column.	**C++ Syntax:** **BOOL** CCampaignClient::GetActualStartVisible(); **void** CCampaignClient::SetActualStartVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualStartVisible[= *boolvalue*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
ActualStartWidth Sets the width of the Actual Start column.	**++ Syntax:** **double** CCampaignClient::GetActualStartWidth(); **void** CCampaignClient::SetActualStartWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualStartWidth[= *value!*]
ActualFinishFilter Sets the filter for the Actual Finish column.	**C++ Syntax:** **CString** CCampaignClient::GetActualFinishFilter (); **void** CCampaignClient::SetActualFinishFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualFinishFilter [= *text$*]
ActualFinishHeaderText Specifies the header text for the Actual Finish column.	**C++ Syntax:** **CString** CCampaignClient::GetActualFinishHeaderText (); **void** CCampaignClient::SetActualFinishHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualFinishHeaderText [= *text$*]
ActualFinishVisible Sets whether or not to display the Actual Finish column.	**C++ Syntax:** **BOOL** CCampaignClient::GetActualFinishVisible(); **void** CCampaignClient::SetActualFinishVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualFinishVisible[= *boolvalue*]
ActualFinishWidth Sets the width of the Actual Finish column.	**++ Syntax:** **double** CCampaignClient::GetActualFinishWidth(); **void** CCampaignClient::SetActualFinishWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ActualFinishWidth[= *value!*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
PlannedQuantityFilter Sets the filter for the Planned Quantity column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedQuantityFilter (); **void** CCampaignClient::SetPlannedQuantityFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedQuantityFilter [= *text$*]
PlannedQuantityHeaderText Specifies the header text for the Planned Quantity column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedQuantityHeaderText (); **void** CCampaignClient::SetPlannedQuantityHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedQuantityHeaderText [= *text$*]
PlannedQuantityVisible Sets whether or not to display the Planned Quantity column.	**C++ Syntax:** **BOOL** CCampaignClient::GetPlannedQuantityVisible(); **void** CCampaignClient::SetPlannedQuantityVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedQuantityVisible[= *boolvalue*]
PlannedQuantityWidth Sets the width of the Planned Quantity column.	**C++ Syntax:** **double** CCampaignClient::GetPlannedQuantityWidth(); **void** CCampaignClient::SetPlannedQuantityWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedQuantityWidth[= *value!*]
RemainingQuantityFilter Sets the filter for the Remaining Quantity column.	**C++ Syntax:** **CString** CCampaignClient::GetRemainingQuantityFilter (); **void** CCampaignClient::SetRemainingQuantityFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingQuantityFilter [= *text$*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
RemainingQuantityHeaderText Specifies the header text for the Remaining Quantity column.	**C++ Syntax:** **CString** CCampaignClient::GetRemainingQuantityHeaderText (); **void** CCampaignClient::SetRemainingQuantityHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingQuantityHeaderText [= *text$*]
RemainingQuantityVisible Sets whether or not to display the Remaining Quantity column.	**C++ Syntax:** **BOOL** CCampaignClient::GetRemainingQuantityVisible(); **void** CCampaignClient::SetRemainingQuantityVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingQuantityVisible[= *boolvalue*]
RemainingQuantityWidth Sets the width of the Remaining Quantity column.	**C++ Syntax:** **double** CCampaignClient::GetRemainingQuantityWidth(); **void** CCampaignClient::SetRemainingQuantityWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingQuantityWidth[= *value!*]
EGUFilter Sets the filter for the EGU (Engineering Units) column.	**C++ Syntax:** **CString** CCampaignClient::GetEGUFilter (); **void** CCampaignClient::SetEGUFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*EGUFilter [= *text$*]
EGUHeaderText Specifies the header text for the EGU (Engineering Units) column.	**C++ Syntax:** **CString** CCampaignClient::GetEGUHeaderText (); **void** CCampaignClient::SetEGUHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*EGUHeaderText [= *text$*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
EGUVisible Sets whether or not to display the EGU (Engineering Units) column.	**C++ Syntax:** **BOOL** CCampaignClient::GetEGUVisible(); **void** CCampaignClient::SetEGUVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*EGUVisible[= *boolvalue*]
EGUWidth Sets the width of the EGU (Engineering Units) column.	**C++ Syntax:** **double** CCampaignClient::GetEGUWidth(); **void** CCampaignClient::SetEGUWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*EGUWidth[= *value!*]
PlannedBatchesFilter Sets the filter for the Planned Batches column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedBatchesFilter (); **void** CCampaignClient::SetPlannedBatchesFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedBatchesFilter [= *text$*]
PlannedBatchesHeaderText Specifies the header text for the Planned Batches column.	**C++ Syntax:** **CString** CCampaignClient::GetPlannedBatchesHeaderText (); **void** CCampaignClient::SetPlannedBatchesHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedBatchesHeaderText [= *text$*]
PlannedBatchesVisible Sets whether or not to display the Planned Batches column.	**C++ Syntax:** **BOOL** CCampaignClient::GetPlannedBatchesVisible(); **void** CCampaignClient::SetPlannedBatchesVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedBatchesVisible[= *boolvalue*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
PlannedBatchesWidth Sets the width of the Planned Batches column.	**C++ Syntax:** **double** CCampaignClient::GetPlannedBatchesWidth(); **void** CCampaignClient::SetPlannedBatchesWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*PlannedBatchesWidth[= *value!*]
RemainingBatchesFilter Sets the filter for the Remaining Batches column.	**C++ Syntax:** **CString** CCampaignClient::GetRemainingBatchesFilter (); **void** CCampaignClient::SetRemainingBatchesFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingBatchesFilter [= *text$*]
RemainingBatchesHeaderText Specifies the header text for the Remaining Batches column.	**C++ Syntax:** **CString** CCampaignClient::GetRemainingBatchesHeaderText (); **void** CCampaignClient::SetRemainingBatchesHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingBatchesHeaderText [= *text$*]
RemainingBatchesVisible Sets whether or not to display the Remaining Batches column.	**C++ Syntax:** **BOOL** CCampaignClient::GetRemainingBatchesVisible(); **void** CCampaignClient::SetRemainingBatchesVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingBatchesVisible[= *boolvalue*]
RemainingBatchesWidth Sets the width of the Planned Remaining column.	**C++ Syntax:** **double** CCampaignClient::GetRemainingBatchesWidth(); **void** CCampaignClient::SetRemainingBatchesWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RemainingBatchesWidth[= *value!*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
RecipeFilter Sets the filter for the Recipe column.	**C++ Syntax:** **CString** CCampaignClient::GetRecipeFilter (); **void** CCampaignClient::SetRecipeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeFilter [= *text$*]
RecipeHeaderText Specifies the columns header text for the Recipe Header column.	**C++ Syntax:** **CString** CCampaignClient::GetRecipeHeaderText (); **void** CCampaignClient::SetRecipeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeHeaderText [= *text$*]
RecipeVisible Sets whether or not to display the Recipe column.	**C++ Syntax:** **BOOL** CCampaignClient::GetRecipeVisible(); **void** CCampaignClient::SetRecipeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeVisible[= *boolvalue*]
RecipeWidth Sets the width of the Recipe column.	**C++ Syntax:** **double** CCampaignClient::GetRecipeWidth(); **void** CCampaignClient::SetRecipeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeWidth[= *value!*]
DescriptionFilter Sets the filter for the Description column.	**C++ Syntax:** **CString** CCampaignClient::GetDescriptionFilter (); **void** CCampaignClient::SetDescriptionFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionFilter [= *text$*]
DescriptionHeaderText Specifies the columns header text for the Description Header column.	**C++ Syntax:** **CString** CCampaignClient::GetDescriptionHeaderText (); **void** CCampaignClient::SetDescriptionHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionHeaderText [= *text$*]

BatchCampaignClient Control Column Properties	
Property	**Syntax**
DescriptionVisible Sets whether or not to display the Description column.	**C++ Syntax:** **BOOL** CCampaignClient::GetDescriptionVisible(); **void** CCampaignClient::SetDescriptionVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionVisible[= *boolvalue*]
DescriptionWidth Sets the width of the Description column.	**C++ Syntax:** **double** CCampaignClient::GetDescriptionWidth(); **void** CCampaignClient::SetDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*DescriptionWidth[= *value!*]

BatchList Section

The following table lists the properties that control the display of the columns in the BatchList control section of the BatchCampaignClient control. These properties are available from automation scripting only.

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchBoundHeaderText Specifies the column header text for the Batch Bound column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchBoundHeaderText(); **void** CCampaignClient::SetBatchBoundHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchBoundHeaderText[= *text$*]
BatchBoundVisible Sets whether or not to display the Batch Bound column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchBoundVisible (); **void** CCampaignClient::SetBatchBoundVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchBoundVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchBoundWidth Sets the width of the Batch Bound column.	**C++ Syntax:** **double** CCampaignClient::GetBatchBoundWidth(); **void** CCampaignClient::SetBatchBoundWidth(double value); **Visual Basic Syntax:** [form.]Control.BatchBoundWidth[= *value!*]
BatchCommandMaskHeaderText Specifies the column header text for the Command Mask column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchCommandMaskHeaderText(); **void** CCampaignClient::SetBatchCommandMaskHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchCommandMaskHeaderText[= *text$*]
BatchCommandMaskVisible Sets whether or not to display the Command Mask column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchCommandMaskVisible (); **void** CCampaignClient::SetBatchCommandMaskVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchCommandMaskVisible [= *boolvalue*]
BatchCommandMaskWidth Sets the width of the Command Mask column.	**C++ Syntax:** **double** CCampaignClient::GetBatchCommandMaskWidth(); **void** CCampaignClient::SetBatchCommandMaskWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchCommandMaskWidth[= *value!*]
BatchDefaultBindingHeaderText Specifies the column header text for the Default Binding column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchDefaultBindingHeaderText(); **void** CCampaignClient::SetBatchDefaultBindingHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDefaultBindingHeaderText[= *text$*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchDefaultBindingVisible Sets whether or not to display the Default Binding column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchDefaultBindingVisible (); **void** CCampaignClient::SetBatchDefaultBindingVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDefaultBindingVisible [= *boolvalue*]
BatchDefaultBindingWidth Sets the width of the Default Binding column.	**C++ Syntax:** **double** CCampaignClient::GetBatchDefaultBindingWidth(); **void** CCampaignClient::SetBatchDefaultBindingWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDefaultBindingWidth[= *value!*]
BatchDescriptionHeaderText Specifies the column header text for the Recipe Description column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchDescriptionHeaderText(); **void** CCampaignClient::SetBatchDescriptionHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDescriptionHeaderText[= *text$*]
BatchDescriptionVisible Sets whether or not to display the Recipe Description column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchDescriptionVisible (); **void** CCampaignClient::SetBatchDescriptionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDescriptionVisible [= *boolvalue*]
BatchDescriptionWidth Sets the width of the Recipe Description column.	**C++ Syntax:** **double** CCampaignClient::GetBatchDescriptionWidth(); **void** CCampaignClient::SetBatchDescriptionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchDescriptionWidth[= *value!*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchElapsedTimeHeaderText Specifies the column header text for the Elapsed Time column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchElapsedTimeHeaderText(); **void** CCampaignClient::SetBatchElapsedTimeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchElapsedTimeHeaderText[= *text$*]
BatchElapsedTimeVisible Sets whether or not to display the Elapsed Time column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchElapsedTimeVisible (); **void** CCampaignClient::SetBatchElapsedTimeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchElapsedTimeVisible [= *boolvalue*]
BatchElapsedTimeWidth Sets the width of the Elapsed Time column.	**C++ Syntax:** **double** CCampaignClient::GetBatchElapsedTimeWidth(); **void** CCampaignClient::SetBatchElapsedTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchElapsedTimeWidth[= *value!*]
BatchFailureHeaderText Specifies the column header text for the Failure column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchFailureHeaderText(); **void** CCampaignClient::SetBatchFailureHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchFailureHeaderText[= *text$*]
BatchFailureVisible Sets whether or not to display the Failure column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchFailureVisible (); **void** CCampaignClient::SetBatchFailureVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchFailureVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchFailureWidth Sets the width of the Failure column.	**C++ Syntax:** **double** CCampaignClient::GetBatchFailureWidth(); **void** CCampaignClient::SetBatchFailureWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchFailureWidth[= *value!*]
BatchIDHeaderText Specifies the column header text for the Batch ID column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchIDHeaderText (); **void** CCampaignClient::SetBatchIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDHeaderText [= *text$*]
BatchIDVisible Sets whether or not to display the Batch ID column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchIDVisible (); **void** CCampaignClient::SetBatchIDVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDVisible [= *boolvalue*]
BatchIDWidth Sets the width of the Batch ID column.	**C++ Syntax:** **double** CCampaignClient::GetBatchIDWidth(); **void** CCampaignClient::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control*.BatchIDWidth[= *value!*]
BatchInternalIDHeaderText Specifies the column header text for the Internal Batch ID column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchInternalIDHeaderText(); **void** CCampaignClient::SetBatchInternalIDHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchInternalIDHeaderText[= *text$*]
BatchInternalIDVisible Sets whether or not to display the Internal Batch ID column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchInternalIDVisible (); **void** CCampaignClient::SetBatchInternalIDVisible (**BOOL** value); **Visual Basic Syntax:** VB: [*form.*]*Control.*BatchInternalIDVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchInternalIDWidth Sets the width of the Internal Batch ID column.	**C++ Syntax:** **double** CCampaignClient::GetBatchInternalIDWidth(); **void** CCampaignClient::SetBatchInternalIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchInternalIDWidth[= *value!*]
BatchModeHeaderText Specifies the column header text for the Batch Mode column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchModeHeaderText(); **void** CCampaignClient::SetBatchModeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchModeHeaderText[= *text$*]
BatchModeVisible Sets whether or not to display the Batch Mode column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchModeVisible (); **void** CCampaignClient::SetBatchModeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchModeVisible [= *boolvalue*]
BatchModeWidth Sets the width of the Batch Mode column.	**C++ Syntax:** **double** CCampaignClient::GetBatchModeWidth(); **void** CCampaignClient::SetBatchModeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchModeWidth[= *value!*]
BatchOpBindParametersHeaderText Specifies the column header text for the Operator Bind parameters column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchOpBindParametersHeaderText(); **void** CCampaignClient::SetBatchOpBindParametersHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindParametersHeaderText[= *text$*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchOpBindParametersVisible Sets whether or not to display the Operator Bind Parameters column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchOpBindParametersVisible (); **void** CCampaignClient::SetBatchOpBindParametersVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindParametersVisible [= *boolvalue*]
BatchOpBindParametersWidth Sets the width of the Operator Bind Parameters column.	**C++ Syntax:** **double** CCampaignClient::GetBatchOpBindParametersWidth(); **void** CCampaignClient::SetBatchOpBindParametersWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindParametersWidth[= *value!*]
BatchOpBindUnitsHeaderText Specifies the column header text for the Operator Bind Units column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchOpBindUnitsHeaderText(); **void** CCampaignClient::SetBatchOpBindUnitsHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindUnitsHeaderText[= *text$*]
BatchOpBindUnitsVisible Sets whether or not to display the Operator Bind Units column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchOpBindUnitsVisible (); **void** CCampaignClient::SetBatchOpBindUnitsVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindUnitsVisible [= *boolvalue*]
BatchOpBindUnitsWidth Sets the width of the Operator Bind Units column.	**C++ Syntax:** **double** CCampaignClient::GetBatchOpBindUnitsWidth(); **void** CCampaignClient::SetBatchOpBindUnitsWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpBindUnitsWidth[= *value!*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchOpInteractionHeaderText Specifies the column header text for the Operator Interaction column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchOpInteractionHeaderText(); **void** CCampaignClient::SetBatchOpInteractionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpInteractionHeaderText[= *text$*]
BatchOpInteractionVisible Sets whether or not to display the Operator Interaction column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchOpInteractionVisible (); **void** CCampaignClient::SetBatchOpInteractionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpInteractionVisible [= *boolvalue*]
BatchOpInteractionWidth Sets the width of the Operator Interaction column.	**C++ Syntax:** **double** CCampaignClient::GetBatchOpInteractionWidth(); **void** CCampaignClient::SetBatchOpInteractionWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchOpInteractionWidth[= *value!*]
BatchParametersRequiredHeaderText Specifies the column header text for the Parameters Required column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchParametersRequiredHeaderText(); **void** CCampaignClient::SetBatchParametersRequiredHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersRequiredHeaderText[= *text$*]
BatchParametersRequiredVisible Sets whether or not to display the Parameters Required column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatch ParametersRequiredVisible (); **void** CCampaignClient::SetBatch ParametersRequiredVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersRequiredVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchParametersRequiredWidth Sets the width of the Parameters Required column.	**C++ Syntax:** **double** CCampaignClient::GetBatchParametersRequiredWidth(); **void** CCampaignClient::SetBatchParametersRequiredWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersRequiredWidth[= *value!*]
BatchParametersSupportedHeaderText Specifies the column header text for the Parameters Supported column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchParametersSupportedHeaderText(); **void** CCampaignClient::SetBatchParametersSupportedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersSupportedHeaderText[= *text$*]
BatchParametersSupportedVisible Sets whether or not to display the Parameters Supported column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchParametersSupportedVisible (); **void** CCampaignClient::SetBatchParametersSupportedVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersSupportedVisible [= *boolvalue*]
BatchParametersSupportedWidth Sets the width of the Parameters Supported column.	**C++ Syntax:** **double** CCampaignClient::GetBatchParametersSupportedWidth(); **void** CCampaignClient::SetBatchParametersSupportedWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchParametersSupportedWidth[= *value!*]
BatchPhaseHeaderText Specifies the column header text for the Phase column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchPhaseHeaderText(); **void** CCampaignClient::SetBatchPhaseHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchPhaseHeaderText[= *text$*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchPhaseVisible Sets whether or not to display the Phase column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchPhaseVisible (); **void** CCampaignClient::SetBatchPhaseVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchPhaseVisible [= *boolvalue*]
BatchPhaseWidth Sets the width of the Phase column.	**C++ Syntax:** **double** CCampaignClient::GetBatchPhaseWidth(); **void** CCampaignClient::SetBatchPhaseWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchPhaseWidth[= *value!*]
BatchProcessCellHeaderText Specifies the header text for Process Cell column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchProcessCellHeaderText(); **void** CCampaignClient::SetBatchProcessCellHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchProcessCellHeaderText[= *text$*]
BatchProcessCellVisible Sets whether or not to display the Process Cell column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchProcessCellVisible (); **void** CCampaignClient::SetBatchProcessCellVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchProcessCellVisible [= *boolvalue*]
BatchProcessCellWidth Sets the width of the Process Cell column.	**C++ Syntax:** **double** CCampaignClient::GetBatchProcessCellWidth(); **void** CCampaignClient::SetBatchProcessCellWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchProcessCellWidth[= *value!*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchRecipeAuditVersionHeaderText Specifies the column header text for the Audit Version column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchAuditVersionHeaderText(); **void** CCampaignClient::SetBatchAuditVersionHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchAuditVersionHeaderText[= *text$*]
BatchRecipeAuditVersionVisible Sets whether or not to display the Audit Version column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchAuditVersionVisible (); **void** CCampaignClient::SetBatchAuditVersionVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchAuditVersionVisible [= *boolvalue*]
BatchRecipeAuditVersionWidth Sets the width of the Audit Version column.	**C++ Syntax:** **double** CCampaignClient::GetBatchAuditVersionWidth(); **void** CCampaignClient::SetBatchAuditVersionWidth(double value); **Visual Basic Syntax:** [form.]*Control.*BatchAuditVersionWidth[= *value!*]
BatchRecipeHeaderText Specifies the column header text for the Recipe Name column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchRecipeHeaderText (); **void** CCampaignClient::SetBatchRecipeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeHeaderText [= *text$*]
BatchRecipeVerHeaderText Specifies the column header text for the Recipe Version column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchRecipeVerHeaderText (); **void** CCampaignClient::SetBatchRecipeVerHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeVerHeaderText[= *text$*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchRecipeVerVisible Sets whether or not to display the Recipe Version column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchRecipeVerVisible (); **void** CCampaignClient::SetBatchRecipeVerVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeVerVisible [= *boolvalue*]
BatchRecipeVerWidth Sets the width of the Recipe Version column.	**C++ Syntax:** **double** CCampaignClient::GetBatchRecipeVerWidth(); **void** CCampaignClient::SetBatchRecipeVerWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeVerWidth[= *value!*]
BatchRecipeVisible Sets whether or not to display the Recipe Name column.	**C++ Syntax** **BOOL** CCampaignClient::GetBatchRecipeVisible (); **void** CCampaignClient::SetBatchRecipeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeVisible [= *boolvalue*]
BatchRecipeWidth Sets the width of the Recipe Name column.	**C++ Syntax:** **double** CCampaignClient::GetBatchRecipeWidth(); **void** CCampaignClient::SetBatchRecipeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchRecipeWidth[= *value!*]
BatchScaleHeaderText Specifies the column header text for the Batch Scale column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchScaleHeaderText(); **void** CCampaignClient::SetBatchScaleHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchScaleHeaderText[= *text$*]
BatchScaleVisible Sets whether or not to display the Batch Scale column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchScaleVisible (); **void** CCampaignClient::SetBatchScaleVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchScaleVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchScaleWidth Sets the width of the Batch Scale column.	**C++ Syntax:** **double** CCampaignClient::GetBatchScaleWidth(); **void** CCampaignClient::SetBatchScaleWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchScaleWidth[= *value!*]
BatchStartTimeHeaderText Specifies the column header text for the Start Time column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchStartTimeHeaderText(); **void** CCampaignClient::SetBatchStartTimeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStartTimeHeaderText[= *text$*]
BatchStartTimeVisible Sets whether or not to display the Start Time column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchStartTimeVisible (); **void** CCampaignClient::SetBatchStartTimeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStartTimeVisible [= *boolvalue*]
BatchStartTimeWidth Sets the width of the Start Time column.	**C++ Syntax:** **double** CCampaignClient::GetBatchStartTimeWidth(); **void** CCampaignClient::SetBatchStartTimeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStartTimeWidth[= *value!*]
BatchStateHeaderText Specifies the header text for the State column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchStateHeaderText(); **void** CCampaignClient::SetBatchStateHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStateHeaderText[= *text$*]
BatchStateVisible Sets whether or not to display the State column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchStateVisible (); **void** CCampaignClient::SetBatchStateVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStateVisible [= *boolvalue*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchStateWidth Sets the width of the State column.	**C++ Syntax:** **double** CCampaignClient::GetBatchStateWidth(); **void** CCampaignClient::SetBatchStateWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchStateWidth[= *value!*]
BatchTypeHeaderText Specifies the header text for the Type column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchTypeHeaderText(); **void** CCampaignClient::SetBatchTypeHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchTypeHeaderText[= *text$*]
BatchTypeVisible Sets whether or not to display the Type column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchTypeVisible (); **void** CCampaignClient::SetBatchTypeVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchTypeVisible [= *boolvalue*]
BatchTypeWidth Sets the width of the Type column.	**C++ Syntax:** **double** CCampaignClient::GetBatchTypeWidth(); **void** CCampaignClient::SetBatchTypeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchTypeWidth[= *value!*]
BatchUnitVisible Sets whether or not to display the Unit column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchUnitVisible (); **void** CCampaignClient::SetBatchUnitVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitVisible [= *boolvalue*]
BatchUnitWidth Sets the width of the Unit column.	**C++ Syntax:** **double** CCampaignClient::GetBatchUnitWidth(); **void** CCampaignClient::SetBatchUnitWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitWidth[= *value!*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchUnitHeaderText Specifies the header text for the Unit column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchUnitHeaderText(); **void** CCampaignClient::SetBatchUnitHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitHeaderText[= *text$*]
BatchUnitsRequiredHeaderText Specifies the column header text for the Units Required column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchUnitsRequiredHeaderText(); **void** CCampaignClient::SetBatchUnitsRequiredHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitsRequiredHeaderText[= *text$*]
BatchUnitsRequiredVisible Sets whether or not to display the Units Required column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchUnitsRequiredVisible (); **void** CCampaignClient::SetBatchUnitsRequiredVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitsRequiredVisible [= *boolvalue*]
BatchUnitsRequiredWidth Sets the width of the Units Required column.	**C++ Syntax:** **double** CCampaignClient::GetBatchUnitsRequiredWidth(); **void** CCampaignClient::SetBatchUnitsRequiredWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitsRequiredWidth[= *value!*]
BatchUnitsSupportedHeaderText Specifies the column header text for the Units Supported column.	**C++ Syntax:** **CString** CCampaignClient::GetBatchUnitsSupportedHeaderText(); **void** CCampaignClient::SetBatchUnitsSupportedHeaderText(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitsSupportedHeaderText[= *text$*]

BatchList Column Properties in the BatchCampaignClient Control	
Property	**Syntax**
BatchUnitsSupportedVisible Sets whether or not to display the Units Supported column.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchUnitsSupportedVisible (); **void** CCampaignClient::SetBatchUnitsSupportedVisible (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchUnitsSupportedVisible [= *boolvalue*]

BatchUnitsSupportedWidth
Sets the width of the Units Supported column.
	C++ Syntax:
	double CCampaignClient::GetBatchUnitsSupportedWidth();
	void CCampaignClient::SetBatchUnitsSupportedWidth(double value);
	Visual Basic Syntax:
	[*form.*]*Control.*BatchUnitsSupportedWidth[= *value!*]

BatchCampaignClient Control Campaign Server Properties

The following table lists the Campaign Server properties for the BatchCampaignClient control.

BatchCampaignClient Control Campaign Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the Campaign Server when the control is instantiated.	**C++ Syntax:** **BOOL** CCampaignClient::GetConnectAtStartup(); **void** CCampaignClient::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control (the BatchList control) is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate is 5.	**C++ Syntax:** **short** CCampaignClient::GetRefreshRate(); **void** CCampaignClient::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]

BatchCampaignClient Control Campaign Server Properties	
Property	**Syntax**
CampaignServerName Sets the name of the remote Campaign Server to which the control should connect. An empty string indicates to use the local Campaign Server.	**C++ Syntax:** **CString** CCampaignClient::GetCampaignServerName(); **void** CCampaignClient::SetCampaignServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.CampaignServerName[= *text$*]

BatchCampaignClient Control Command Buttons Properties

The following tables list the properties that control the display of command buttons in the BatchCampaignClient control.

Campaign

BatchCampaignClient Control Command Buttons Properties – Campaign	
Property	**Syntax**
AddCampaignButton Sets whether or not to display the Add Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAddCampaignButton(); **void** CCampaignClient::SetAddCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.AddCampaignButton[= *boolvalue*]
DuplicateCampaignButton Sets whether or not to display the Duplicate Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetDuplicateCampaignButton(); **void** CCampaignClient::SetDuplicateCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.DuplicateCampaignButton[= *boolvalue*]

BatchCampaignClient Control Command Buttons Properties – Campaign	
Property	**Syntax**
ModifyCampaignButton Sets whether or not to display the Modify Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetModifyCampaignButton(); **void** CCampaignClient::SetModifyCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ModifyCampaignButton[= *boolvalue*]
PauseCampaignButton Sets whether or not to display the Pause Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetPauseCampaignButton(); **void** CCampaignClient::SetPauseCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.PauseCampaignButton[= *boolvalue*]
RemoveCampaignButton Sets whether or not to display the Remove Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetRemoveCampaignButton(); **void** CCampaignClient::SetRemoveCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RemoveCamapignButton[= *boolvalue*]
RestartCampaignButton Sets whether or not to display the Restart Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetRestartCampaignButton(); **void** CCampaignClient::SetRestartCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RestartCampaignButton[= *boolvalue*]
StartCampaignButton Sets whether or not to display the Start Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetStartCampaignButton(); **void** CCampaignClient::SetStartCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StartCampaignButton[= *boolvalue*]

BatchCampaignClient Control Command Buttons Properties – Campaign	
Property	**Syntax**
ViewCampaignButton Sets whether or not to display the View Campaign button.	**C++ Syntax:** **BOOL** CCampaignClient::GetViewCampaignButton(); **void** CCampaignClient::SetViewCampaignButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewCampaignButton[= *boolvalue*]

Batch List

BatchCampaignClient Control Command Buttons Properties – Batch List	
Property	**Syntax**
AbortBatchButton Sets whether or not to display the Abort Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAbortBatchButton (); **void** CCampaignClient::SetAbortBatchButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AbortBatchButton[= *boolvalue*]
AddBatchButton Sets whether or not to display the Add Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAddBatchButton(); **void** CCampaignClient::SetAddBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddBatchButton[= *boolvalue*]
AddOperatorCommentsButton Sets whether or not to display the Add Comment button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAddOperatorCommentsButton(); **void** CCampaignClient::SetAddOperatorCommentsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddOperatorCommentsButton[= *boolvalue*]

BatchCampaignClient Control Command Buttons Properties – Batch List	
Property	**Syntax**
AlarmsButton Sets whether or not to display the Alarms button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAlarmsButton(); **void** CCampaignClient::SetAlarmsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AlarmsButton[= *boolvalue*]
AutomaticButton Sets whether or not to display the Automatic Mode button.	**C++ Syntax:** **BOOL** CCampaignClient::GetAutomaticButton(); **void** CCampaignClient::SetAutomaticButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AutomaticButton[= *boolvalue*]
BindingPromptsButton Sets whether or not to display the Binding Prompts button.	**C++ Syntax:** **BOOL** CCampaignClient::GetBindingPromptsButton(); **void** CCampaignClient::SetBindingPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BindingPromptsButton[= *boolvalue*]
ClearAllFailuresButton Sets whether or not to display the Clear All Failures button.	**C++ Syntax:** **BOOL** CCampaignClient::GetClearAllFailuresButton(); **void** CCampaignClient::SetClearAllFailuresButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ClearAllFailuresButton[= *boolvalue*]
HoldBatchButton Sets whether or not to display the Hold Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetHoldBatchButton(); **void** CCampaignClient::SetHoldBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*HoldBatchButton[= *boolvalue*]
ManualButton Sets whether or not to display the Manual Mode button.	**C++ Syntax:** **BOOL** CCampaignClient::GetManualButton(); **void** CCampaignClient::SetManualButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ManualButton[= *boolvalue*]

BatchCampaignClient Control Command Buttons Properties – Batch List	
Property	**Syntax**
OperatorPromptsButton Sets whether or not to display the Operator Prompts button.	**C++ Syntax:** **BOOL** CCampaignClient::GetOperatorPromptsButton(); **void** CCampaignClient::SetOperatorPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.OperatorPromptsButton[= *boolvalue*]
RemoveBatchButton Sets whether or not to display the Remove Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetRemoveBatchButton(); **void** CCampaignClient::SetRemoveBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RemoveBatchButton[= *boolvalue*]
RestartBatchButton Sets whether or not to display the Restart Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetRestartBatchButton(); **void** CCampaignClient::SetRestartBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RestartBatchButton[= *boolvalue*]
SFCButton Sets whether or not to display the SFC button.	**C++ Syntax:** **BOOL** CCampaignClient::GetSFCButton(); **void** CCampaignClient::SetSFCButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.SFCButton[= *boolvalue*]
StartBatchButton Sets whether or not to display the Start Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetStartBatchButton(); **void** CCampaignClient::SetStartBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StartBatchButton[= *boolvalue*]

BatchCampaignClient Control Command Buttons Properties – Batch List	
Property	**Syntax**
StopBatchButton Sets whether or not to display the Stop Batch button.	**C++ Syntax:** **BOOL** CCampaignClient::GetStopBatchButton(); **void** CCampaignClient::SetStopBatchButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StopBatchButton[= *boolvalue*]

BatchCampaignClient Control Miscellaneous Properties

The following table lists the properties that control the miscellaneous settings for the BatchCampaignClient control.

BatchCampaignClient Control Miscellaneous Properties	
Property	**Syntax**
BatchInfoTabVisible Sets whether or not to display the BatchInfo tab.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchInfoTabVisible(); **void** CCampaignClient::SetBatchInfoTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.BatchInfoTabVisible[= *boolvalue*]
BatchListControlTabVisible Sets whether or not to display the BatchList Control tab.	**C++ Syntax:** **BOOL** CCampaignClient::GetBatchListControlTabVisible(); **void** CCampaignClient::SetBatchListControlTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.BatchListControlTabVisible[= *boolvalue*]
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CCampaignClient::GetEnableRightMouseMenu(); **void** CCampaignClient::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]

BatchCampaignClient Control Miscellaneous Properties	
Property	**Syntax**
InformationTabVisible Sets whether or not to display the Information tabs – the Batch Info and Batch List tabs.	**C++ Syntax:** **BOOL** CCampaignClient::GetInformationTabVisible(); **void** CCampaignClient::SetInformationTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*InformationTabVisible[= *boolvalue*]
MouseDblClickedEnabled Sets whether or not to enable operators double-click access to commands. When enabled, double-clicking displays the Create Campaign dialog box.	**C++ Syntax:** **BOOL** CCampaignClient::GetMouseDblClickedEnabled(); **void** CCampaignClient::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MouseDblClickedEnabled[= *boolvalue*]
StatusBarEnabled Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CCampaignClient::GetStatusBarEnabled(); **void** CCampaignClient::SetStatusBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StatusBarEnabled[= *boolvalue*]
ToolBarEnabled Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CCampaignClient::GetToolBarEnabled(); **void** CCampaignClient::SetToolBarEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarEnabled[= *boolvalue*]
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when the operator executes a command.	**C++ Syntax:** **BOOL** CCampaignClient::GetVerifyCommandActions(); **void** CCampaignClient::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VerifyCommandActions[= *boolvalue*]

BatchCampaignClient Control Security Properties

The following tables list the properties that control the security settings for elements on the

BatchCampaignClient control.

Property Access

BatchCampaignClient Control Security Properties – Property Access	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CCampaignClient::GetColumnEditEnabled(); **void** CCampaignClient::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ColumnEditEnabled[= *boolvalue*]
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CCampaignClient::GetSortOrderEditEnabled(); **void** CCampaignClient::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SortOrderEditEnabled[= *boolvalue*]
ServerEditEnabled Sets whether or not the operator can edit the Campaign Server for the control at run time.	**C++ Syntax:** **BOOL** CCampaignClient::GetServerEditEnabled(); **void** CCampaignClient::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ServerEditEnabled [= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control at run time.	**C++ Syntax:** **BOOL** CCampaignClient::GetRefreshRateEditEnabled(); **void** CCampaignClient::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RefreshRateEditEnabled[= booleanvalue]

BatchCampaignClient Control Security Properties – Property Access	
Property	**Syntax**
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page at run time.	**C++ Syntax:** **BOOL** CCampaignClient::GetMiscEditEnabled(); **void** CCampaignClient::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.MiscEditEnabled[= *boolvalue*]
CommandBtnsEditEnabled If TRUE, the operator can edit the property pages settings for the command buttons at run time.	**C++ Syntax:** **BOOL** CCampaignClient::Get CommandBtnsEditEnabled(); **void** CCampaignClient::Set CommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.CommandBtnsEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CCampaignClient::Get EnableIFIXSecurity (); **void** CCampaignClient::Set EnableIFIXSecurity (**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableIFIXSecurity[= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the Campaign Server connection at run time.	**C++ Syntax:** **BOOL** CCampaignClient::GetToggleConnectionEnabled(); **void** CCampaignClient::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ToggleConnectionEnabled[= booleanvalue]

BatchList Viewers

BatchCampaignClient Control Security Properties – BatchList Viewer	
Property	**Syntax**
ViewSFC Sets whether or not the operator can access the SFC ActiveX control by double-clicking a batch in the list or by clicking the BatchList's SFC toolbar button. The SFC control displays information about the selected batch. ***NOTE:*** *To allow for double-clicking in the BatchList control by enabling the DoubleClickAction property on the Miscellaneous tab. For more information, refer to the BatchCampaignClient Control Miscellaneous Properties section.*	**C++ Syntax:** **BOOL** CCampaignClient::GetViewSFC(); **void** CCampaignClient::SetViewSFC(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewSFC[= *boolvalue*]
ViewAlarms Sets whether or not the operator can access the View Alarms dialog box by clicking the View Alarms button.	**C++ Syntax:** **BOOL** CCampaignClient::GetViewAlarms(); **void** CCampaignClient::SetViewAlarms(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewAlarms[= *boolvalue*]
ViewOperatorPrompts Sets whether or not the operator can access the Operator Prompts dialog box by clicking on the Operator Prompts button.	**C++ Syntax:** **BOOL** CCampaignClient::GetViewOperatorPrompts(); **void** CCampaignClient::SetViewOperatorPrompts(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewOperatorPrompts[= *boolvalue*]
ViewBindingPrompts Sets whether or not the operator can access the Binding Prompts dialog by clicking the Binding Prompts button.	**C++ Syntax:** **BOOL** CCampaignClient::GetViewBindingPrompts(); **void** CCampaignClient::SetViewBindingPrompts(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewBindingPrompts[= *boolvalue*]

BatchCampaignClient Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchCampaignClient control.

BatchCampaignClient Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements Sets a default signature type (None, Performed By, Performed By/Verified By) to all of the commands for this ActiveX control. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CCampaignClient::GetUseDefaultSignatureRequirements();**void** CCampaignClient::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

BatchCampaignClient Control Color Properties

The following table lists the color properties for the BatchCampaignClient control.

BatchCampaignClient Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetBackColor(); **void** CCampaignClient::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BackColor[= *color*]
EvenRowBackColor Sets the background color of the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetEvenRowBackColor(); **void** CCampaignClient::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EvenRowBackColor[= *color%*]

BatchCampaignClient Control Color Properties	
Property	**Syntax**
EvenRowTextColor Sets the color of the text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetEvenRowTextColor(); **void** CCampaignClient::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetGridColor(); **void** CCampaignClient::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetHeaderBackColor(); **void** CCampaignClient::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color*]
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetHeaderTextColor(); **void** CCampaignClient::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for the odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetOddRowBackColor(); **void** CCampaignClient::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]

BatchCampaignClient Control Color Properties	
Property	**Syntax**
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CCampaignClient::GetOddRowTextColor(); **void** CCampaignClient::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*OddRowTextColor[= *color%*]

NOTE: *For examples on setting color properties, refer to the Color Property Examples section in the BatchList ActiveX Control chapter.*

BatchCampaignClient Control Fonts Properties

The following table lists the properties that control the fonts in the BatchCampaignClient control.

BatchCampaignClient Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CCampaignClient::GetHeaderFont(); **void** CCampaignClient::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeaderFont[= *stdfontvariable*]
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CCampaignClient::GetTextFont(); **void** CCampaignClient::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *stdfontvariable*]

BatchCampaignClient Control Methods

The BatchCampaignClient control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method

- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method
- SetBatchListSortKeys Method
- SetBatchListColumnOrder Method
- AboutBox Method

ConnectToServer Method

Description

Establishes the connection to the Campaign Server.

C++ Syntax

```
BOOL CCampaignClient:: ConnectToServer();
```

Visual Basic Syntax

```
[form.]Control.ConnectToServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if a connection is made.
- 0 if a connection is not made.

C++ Example

```
BOOL result = pBatchCampaignClient->ConnectToServer();
```

Visual Basic Example

```
BatchCampaignClient1.ConnectToServer
```

DisconnectFromServer Method

Description

Disconnects from the currently connected Campaign Server.

C++ Syntax

```
BOOL CCampaignClient::DisconnectFromServer();
```

Visual Basic Syntax

```
[form.]Control.DisconnectFromServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the Campaign Server is disconnected.
- 0 if the Campaign Server is connected.

C++ Example

```
BOOL result = pBatchCampaignClient->DisconnectFromServer();
```

Visual Basic Example

```
BatchCampaignClient1.DisconnectFromServer
```

GetCommandSignatureRequirements Method

Description

Gets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CCampaignClient::GetCommandSignatureRequirements(long Command, long*
SignatureRequirements, BSTR* PerformedBy, BSTR* VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.GetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchCampaignClient the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcCreateCampaign = 1 • bcDeleteCampaign = 2 • bcStartCampaign = 3 • bcPauseCampaign = 4 • bcRestartCampaign = 5 • bcModifyCampaign = 6 • bcViewCampaign = 7 • bcDuplicateCampaign = 8 • bcAbortBatch = 9 • bcAddBatch = 10 • bcAutoMode = 11 • bcBindingPrompts = 12 • bcClearAllFailures = 13 • bcHoldBatch = 14 • bcManualMode = 15 • bcOperatorPrompts = 16 • bcRemoveBatch = 17 • bcRestartBatch = 18 • bcStartBatch = 19 • bcStopBatch = 20 • bcAddComment = 21 • bcChangeParameter = 22
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2

Parameter	Description
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Remarks

The following constants exist in two places: BatchListlib and CampaignClientLib:

- bcAbortBatch
- bcAddBatch
- bcAutoMode
- bcBindingPrompts
- bcClearAllFailures
- bcHoldBatch
- bcManualMode
- bcOperatorPrompts
- bcRemoveBatch
- bcRestartBatch
- bcStartBatch
- bcStopBatch
- bcAddComment
- bcChangeParameter.

Each constant has a different value. It is recommended that you fully indicate which constant that you intend to use. For example, for bcAddBatch, in Visual Basic, use Command = CampaignClientLib.bcAddBatch or Command = BatchListLib.bcAddBatch.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
```

```
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,
     } SIGNATURETYPE;
     typedef enum _tagCommandID{
          bcDefault = 0,
          bcCreateCampaign = 1,
          bcDeleteCampaign = 2,
          bcStartCampaign = 3,
          bcPauseCampaign = 4,
          bcRestartCampaign = 5,
          bcModifyCampaign = 6,
          bcViewCampaign = 7,
          bcDuplicateCampaign = 8,
          bcAbortBatch = 9,
          bcAddBatch = 10,
          bcAutoMode = 11,
          bcBindingPrompts = 12,
          bcClearAllFailures = 13,
          bcHoldBatch = 14,
          bcManualMode = 15,
          bcOperatorPrompts = 16,
          bcRemoveBatch = 17,
          bcRestartBatch = 18,
          bcStartBatch = 19,
          bcStopBatch = 20,
          bcAddComment = 21,
          bcChangeParameter = 22
     } COMMANDID;
     BSTR bstrPerformedBy;
     BSTR bstrVerifiedBy;
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;
     // Example of reading the signature requirements for the
     // Start Campaign Command.
     // Note, that if the UseDefaultSignatureRequirements is TRUE
     // then the setting for the Default command is used,
     // else the setting for the start batch command is used.
     if (m_pBatchCampaignClient->GetUseDefaultSignatureRequirements ())
     {
          lCommand = bcDefault;
     }
     else
     {
          lCommand = bcStartCampaign;
     }

     m_pBatchCampaignClient->GetCommandSignatureRequirements ( lCommand, &lSignatureType,
&bstrPerformedBy, &bstrVerifiedBy);
     strPerformedBy = bstrPerformedBy;
     strVerifiedBy = bstrVerifiedBy;
     ::SysFreeString(bstrPerformedBy);
     ::SysFreeString(bstrVerifiedBy);
     if (lSignatureType == stNone)
     {
```

```
          AfxMessageBox ("No Signature requirements for the Start Campaign Command");
     }
     else if (lSignatureType == stPerformedBy)
     {
          AfxMessageBox ("Signature requirements for the Start Campaign Command are
Perform By: " + strPerformedBy);
     }
     else if (lSignatureType == stPerformedByVerifiedBy)
     {
          AfxMessageBox ("Signature requirements for the Start Campaign Command are
Perform By: " + strPerformedBy + " Verify By: " + strVerifiedBy);
     }
```

Visual Basic Example

```
Private Sub Command1_Click()
Dim Command As CampaignClientLib.COMMANDID
Dim SignatureType As CampaignClientLib.SignatureType
Dim strPerformedBy As String
Dim strVerifiedBy As String
' example of reading the signature requirements for the
' Start Campaign Command note that if the
' UseDefaultSignatureRequirements is TRUE then
' the setting for the Default command is used,
' else the setting for the start batch command is used.
If BatchList1.UseDefaultSignatureRequirements Then
    Command = bcDefault
Else
    Command = bcStartCampaign
End If
BatchList1.GetCommandSignatureRequirements Command, SignatureType, strPerformedBy,
strVerifiedBy
If SignatureType = stNone Then
    MsgBox ("No Signature requirements for the Start Campaign Command")
ElseIf SignatureType = stPerformedBy Then
    MsgBox ("Signature requirements for the Start Campaign Command are Perform By: " +
strPerformedBy)
ElseIf SignatureType = stVerifiedBy Then
    MsgBox ("Signature requirements for the Start Campaign Command are Perform By: " +
strPerformedBy + " Verified By: " + strVerifiedBy)
End If
```

SetCommandSignatureRequirements Method

Description

Sets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CCampaignClient::SetCommandSignatureRequirements(Long Command, Long
SignatureRequirements, CString PerformedBy, CString VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.SetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchCampaignClient the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcCreateCampaign = 1 • bcDeleteCampaign = 2 • bcStartCampaign = 3 • bcPauseCampaign = 4 • bcRestartCampaign = 5 • bcModifyCampaign = 6 • bcViewCampaign = 7 • bcDuplicateCampaign = 8 • bcAbortBatch = 9 • bcAddBatch = 10 • bcAutoMode = 11 • bcBindingPrompts = 12 • bcClearAllFailures = 13 • bcHoldBatch = 14 • bcManualMode = 15 • bcOperatorPrompts = 16 • bcRemoveBatch = 17 • bcRestartBatch = 18 • bcStartBatch = 19 • bcStopBatch = 20 • bcAddComment = 21 • bcChangeParameter = 22

Parameter	Description
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,
     } SIGNATURETYPE;
     typedef enum _tagCommandID{
          bcDefault = 0,
          bcCreateCampaign = 1,
          bcDeleteCampaign = 2,
          bcStartCampaign = 3,
          bcPauseCampaign = 4,
          bcRestartCampaign = 5,
          bcModifyCampaign = 6,
          bcViewCampaign = 7,
          bcDuplicateCampaign = 8,
          bcAbortBatch = 9,
          bcAddBatch = 10,
          bcAutoMode = 11,
          bcBindingPrompts = 12,
          bcClearAllFailures = 13,
          bcHoldBatch = 14,
          bcManualMode = 15,
          bcOperatorPrompts = 16,
          bcRemoveBatch = 17,
```

```
          bcRestartBatch = 18,
          bcStartBatch = 19,
          bcStopBatch = 20,
          bcAddComment = 21,
          bcChangeParameter = 22
     } COMMANDID;
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;
     strPerformedBy = _T("Operator");
     strVerifiedBy = _T("Supervisor");
     lCommand = bcStartCampaign;
     lSignatureType = stPerformedByVerifiedBy;
     m_pBatchList->SetCommandSignatureRequirements( lCommand, lSignatureType,
strPerformedBy, strVerifiedBy );
```

Visual Basic Example

```
' example to set a specific command signature requirement
Dim Command As BATCHLISTLib.COMMANDID
Dim SignatureType As BATCHLISTLib.SignatureType
Dim strPerformedBy As String
Dim strVerifiedBy As String
Command = bcStopBatch
strPerformedBy = "Operator"
strVerifiedBy = "Supervisor"
SFC1.SetCommandSignatureRequirements Command, stPerformedByVerifiedBy, strPerformedBy,
strVerifiedBy
End Sub
```

SetBatchListSortKeys Method

Description

Sets the sort order for the Campaign Client's Batch List. Each key is the absolute column. The keys correspond to column numbers.

C++ Syntax

```
BOOL CCampaignClient::SetBatchListSortKeys(long lSortKey1, long lSortKey2, long
lSortKey3, short sSortOrder1, short sSortOrder2, short sSortOrder3);
```

Visual Basic Syntax

```
[form.]Control.SetBatchListSortKeys(lSortKey1 As Long, lSortKey2 As Long, lSortKey3 As
Long, sSortOrder1 As Long, sSortOrder2 As Long, sSortOrder3 As Long) As Boolean
```

Parameters

Parameter	Description
lSortKey1	The absolute column number of the sort key. 10. 0 = not used 11. 1 = ascending order 12. 2 = descending order
lSortKey2	The absolute column number of the sort key. 13. 0 = not used 14. 1 = ascending order 15. 2 = descending order
lSortKey3	The absolute column number of the sort key. 16. 0 = not used 17. 1 = ascending order 18. 2 = descending order
sSortOrder1	The absolute column number of sort key 1.
sSortOrder2	The absolute column number of sort key 2.
sSortOrder3	The absolute column number of sort key 3.

Return Type

Boolean.

- 1 = successful
- 0 = sort order was not set

C++ Example

```
void CCampaignDlg::OnChangeBatchlistSortOrderAscButton()
{
      // note that the column numbers refer to the column's position in the columns
property page
      m_pCampaignClient->SetBatchListSortKeys(1, 2, 3, 1, 1, 1); // sort by cols 1,2,3,
ascending

}
```

Visual Basic Example

```
Private Sub SetSortAscendingCommand_Click()
Dim bRet As Boolean
' sort columns 1, 2, 3 in ascending order
bRet = CampaignClient1.SetBatchListSortKeys(1, 2, 3, 1, 1, 1)
End Sub
```

Remarks

Specify 0 (zero) as the sort key if you do not want to use a sort key. A key cannot be 0 unless its successors are also 0.

Each sort key is the absolute column that the sort will be based upon. The order corresponds to the key of the same number. It can be ascending or descending.

SetBatchListColumnOrder Method

Description

Sets the order of the columns of the Campaign Client's Batch List using an array of longs.

C++ Syntax

```
BOOL CCampaignClient::SetBatchListColumnOrder(const VARIANT& vColumnOrderArray);
```

Visual Basic Syntax

```
[form.]Control.SetBatchListColumnOrder(vColumnOrderArray) As Boolean
```

Parameters

Parameter	Description
vColumnOrderArray	An array of columns indexed by absolute order and stored as relative.

Return Type

Boolean.

- TRUE if successful.
- FALSE if not successful.

  NOTE: *The function will not be successful if all twenty-six array values are not set or if any of the values are repeated.*

C++ Example

```
void CCampaignDlg::OnChangeBatchlistColumnOrderButton()
```

```
{

      static BOOL bReversed=FALSE;
      VARIANT vDataRecord;
      VariantInit(&vDataRecord);
      vDataRecord.vt = VT_VARIANT | VT_ARRAY;
      vDataRecord.parray = NULL;
      SAFEARRAY*               psaRecord;
      SAFEARRAYBOUND     rgsabound[1];
      // set the array bounds (for the 26 columns in batchlist)
      rgsabound[0].lLbound = 0;
      rgsabound[0].cElements = 26;
      // create the safe array descriptor
      psaRecord = SafeArrayCreate(VT_I4, 1, rgsabound);
      if (psaRecord == NULL)
                  return;
      long columnOrder[26]; // 26 columns in batch list
      if (bReversed)
      {
                  // restore the column order
                  // init to 1,2,3,...26.
                  for (long i=0;i<26;i++)
                   {
                              columnOrder[i] = i+1;  // one's based column
                              SafeArrayPutElement(psaRecord, &i, &columnOrder[i]);
                  }
                  bReversed=FALSE;
      }
      else
      {
                  // Reverse the column order
                  // init to 26,25,24,...1 so that the order is reversed.
                  for (long i=0;i<26;i++)
                 {
                              columnOrder[i] = 26-i;
                              SafeArrayPutElement(psaRecord, &i, &columnOrder[i]);
                  }
                  bReversed=TRUE;
       }
      // copy the array
      vDataRecord.parray = psaRecord;
      m_pCampaignClient->SetBatchListColumnOrder(vDataRecord);

      VariantClear(&vDataRecord);  // clear the memory

}
```

Visual Basic Example

```
Private Sub SetBatchListColumnOrderCommand_Click()
Dim bRet As Boolean
Static BatchList1_Reversed As Boolean
Dim vRecord As Variant
Dim vArray(25) As Long '26 columns in this control
If (BatchList1_Reversed) Then
    For i = 0 To 25
        vArray(i) = i + 1 ' make it 1,2,3,4,...26
    Next i
```

```
    BatchList1_Reversed = False
Else
    For i = 0 To 25
        vArray(i) = 26 - i  ' make it 26, 25, 24, 23, ...,1
    Next i
    BatchList1_Reversed = True
End If
vRecord = vArray
bRet = CampaignClient1.SetBatchListColumnOrder(vRecord)
End Sub
```

Remarks

The following list provides the twenty-six columns, in relative order, available for the Campaign Client's Batch List:

- BatchID
- Recipe
- RecipeVersion
- Description
- Scale
- StartTime
- ElapsedTime
- Failure
- State
- Mode
- Type
- ParametersRequired
- UnitsRequired
- ParametersSupported
- UnitsSupported
- BatchBound
- DefaultBinding
- OpBindParameters
- OpBindUnits
- OpInteraction
- ProcessCell
- Phase
- Unit
- InternalID
- CommandMask

- RecipeAuditVersion

Determining Column Order

Column order is based upon two situations: relative and absolute. Absolute column order is based upon the order of the columns at that instance and is the order that you set. Relative order is based upon the default order of the columns. Each column is numbered sequentially. For example, the columns listed in the following table show the default order for the first six of the Campaign Client's BatchList control's columns.

Column	**Absolute Order**	**Relative Order**
Batch ID	1	1
Recipe	2	2
Recipe Version	3	3
Description	4	4
Scale	5	5
Start Time	6	6

In this case, the columns appear as follows on the control.

Batch ID	Recipe	Recipe Version	Description	Scale	Start Time

If the user switches the order of columns 1 and 2, and also switches the order of columns 3 and 6, the new order assignments are as follows:

Column	**Absolute Order**	**Relative Order**
Recipe	1	2
Batch ID	2	1
Start Time	3	6
Description	4	4
Scale	5	5

Column	Absolute Order	Relative Order
Recipe Version	6	3

Note that the absolute column order does not change. In this case, specifying the absolute column 1 refers to the Recipe column, while referring to the relative column 1 refers to the Batch ID column.

In this case, the columns would appear as follows on the control:

Recipe	Batch ID	Start Time	Description	Scale	Recipe Version

AboutBox Method

Description

Displays the About box.

C++ Syntax

```
void CCampaignClient::AboutBox()
```

Visual Basic Syntax

```
[form.]CampaignManager1.AboutBox()
```

Parameters

None.

Return Type

Void.

C++ Example

```
CampaignManager1.AboutBox();
```

Visual Basic Example

```
CampaignManager1.AboutBox
```

BatchCampaignClient Control Events

The BatchCampaignClient control supports the following events:

- ConnectedToServer Event
- DisconnectedFromServer Event
- ServerChanged Event

ConnectedToServer Event

Description

Occurs when the control has connected to the Campaign Server.

Event ID

602

C++ Syntax

```
void ConnectedToServer();
```

Visual Basic Syntax

```
Event ConnectedToSever()
```

DisconnectedFromServer Event

Description

Occurs when the control has disconnected from the Campaign Server.

Event ID

603

C++ Syntax

```
void DisconnectedFromServer();
```

Visual Basic Syntax

```
Event DisconnectedFromServer()
```

ServerChanged Event

Description

Occurs when the control has switched Campaign Servers.

Event ID

600

C++ Syntax

```
void ServerChanged();
```

Visual Basic Syntax

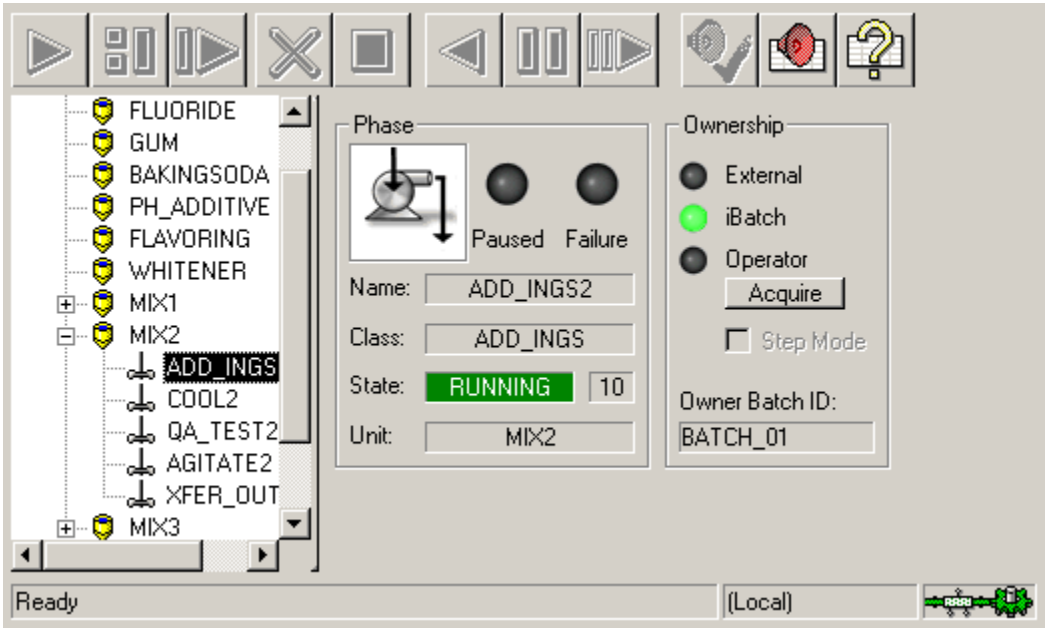
```
Event ServerChanged()
```

BatchManualPhase ActiveX Control

The "Intellution BatchManualPhase Control" allows you to manually run and control a single phase. It provides a tree view (shown in the following figure) of the area model from which you can select the appropriate phase. Once you select a phase, you can use the command buttons to control the phase.

You cannot use the BatchManualPhase ActiveX control to execute recipe level commands. For example, you cannot manually abort a recipe from this control – you can only abort a phase. For information on using recipe level commands, see BatchRecipeList ActiveX Control.

Using the BatchManualPhase ActiveX control's property pages, developers can configure the control's appearance and functionality for operators. For example, the developer can configure the colors of the control states, the command buttons that are displayed, and security for the control.



BatchManualPhase Control with Area Model Tree View

The BatchManualPhase ActiveX control consists of the BatchAreaModelTree and BatchPhasePlate control. The BatchAreaModelTree control should only be used within the BatchManualPhase control. The BatchPhasePlate control can be used outside of the BatchManualPhase control. The BatchPhasePlate is described in the Using the BatchPhasePlate Control section.

NOTE: *You can hide the area model tree, status bar, and toolbar on the BatchManualPhase ActiveX control, leaving only the phase plate visible. You can use the SetCurrentPhase method to instruct the BatchManualPhase control which phase to display. It is much easier to use the phase plate control in this manner, within the BatchManualPhase ActiveX control, than to use the phase plate control by itself as the BatchPhasePlate control.*

Controlling Phases Using BatchManualPhase Control

Similar to the Batch Execution Client's Phase Control screen, the BatchManualPhase control lets you view the current status of a phase, as well as control the execution of phases. Using the BatchManualPhase control you can view information about the current phase, including:

- Phase icon
- Paused or failure conditions
- Phase name
- Phase class
- Phase state

  NOTE: *Refer to* Operations Manual *for a description of each phase state.*
- Current step index.
- Unit the phase is running on.

You can also view and control phase ownership parameters, including:

- Owner of the selected phase (External, Batch Execution, or Operator).
- Operator's ability to acquire or release the selected phase.
- Operator's ability to put the acquired phase into step mode, or to run the phase automatically.
- Batch ID that owns this phase.

Using the BatchManualPhase Control

By clicking on any of the recipe's units in the tree, you can select the phases that run on that unit. When you select a phase, the right side of the screen displays information about the selected phase, including its owner and current state.

From the BatchManualPhase control screen, you can manually control the execution of phases. By manually controlling the phase, you can diagnose any problems or failures that occur during batch production, isolating the single phase that is the cause of the problem. For more information, refer to Manually Controlling Phases.

Phase Ownership

A phase may be manipulated only by its owner. To control a phase, you must first acquire the phase. A phase may be acquired only if it has not exceeded the maximum number of owners configured for that phase. The number of its potential owners is configured in the Equipment Editor. For additional information on defining resource ownership, refer to the Equipment Configuration Manual.

Command Toolbar

You can use the command buttons to control the selected phase. The command toolbar is displayed at the top of the control. The following figure shows the command toolbar; the table that follows describes the function of each button.

NOTE: *The developer can configure if the command toolbar will appear on the BatchManualPhase control and what buttons will appear on it.*

BatchManualPhase Command Toolbar

Command Button Description	
This button...	**Does this...**
	• Starts the selected phase. You are prompted to enter the name of the batch to be used to run this phase. • In step mode, increments the step index.
	Pauses the phase at its current step index.
	Restarts a held phase.
	Aborts the running phase entirely.
	Stops the running phase.
	Resets the currently running phase.

Command Button Description	
This button...	**Does this...**
	Pauses the currently running phase.
	Resumes a paused phase.
	Clears all failures.
	Displays the Alarms dialog box with alarms for this phase.
	Displays the Operator Prompts dialog box for this phase.

Tree View

The tree view shows the process cells, units, and phases represented in the area model with which you are working.

Context Menu

Right-clicking anywhere in the BatchManualPhase control displays a context menu that contains the commands (the same as in the command toolbar) and a Properties menu item (to display the property pages for the BatchManualPhase control).

Status Bar

The status bar at the bottom of the BatchManualPhase Control provides the following:

- Status information.
- Name of the current server the control is (or will be) connected to.
- Icon for connecting or disconnecting to VBIS and the server. The icon also reports the status of the connection.
- Server name, that when clicked, displays the VBIS Server property page.

NOTE: *The developer can configure if the status bar will appear on the BatchManualPhase control.*

Step Mode

If a phase is owned by the operator, the Step Mode check box is enabled. If checked, clicking the Start button in the command toolbar increments the step index for the selected phase.

Manually Controlling Phases

Operators can manually control any phases defined for the current area model from the BatchManualPhase control screen. This can be a useful mechanism for cleaning equipment, or for completing a batch that was interrupted.

Example: Manually Controlling Phases

For example, during the production of a batch of pudding, the operator realizes that the milk used in the recipe is spoiled. The operator then:

1. Stops the batch and removes the mixture from its tank.
2. Acquires the phase ADD_WATER.
3. Manually executes the phase. This fills the tank with water.
4. Acquires a second phase, AGITATE_1.
5. Executes the AGITATE_1 phase which agitates the water in the tank for five minutes.
6. Acquires the phase DRAIN_2.
7. Manually executes this phase and the tank is emptied.

The tank is now clean and is ready to run a new batch.

Single-Step Mode

Developers can place phases in single-step mode. By stepping through a phase one step at a time, a developer can troubleshoot a phase that is not executing properly.

In the phase logic, phases can be defined with a pause transition between each of its individual steps. Only phases defined this way in the phase logic can be placed in single step mode.

Refer to the Phase Programming Manual for more information on phase logic.

BatchManualPhase Control Properties

The BatchManualPhase control provides property pages that you can use to view and change the control's properties. The following sections describe each property for the BatchManualPhase control. The properties are grouped into the following categories:

- Server properties
- Phase Plate properties
- Command Buttons properties
- Security properties

- Miscellaneous properties
- Electronic Signature properties
- Colors properties

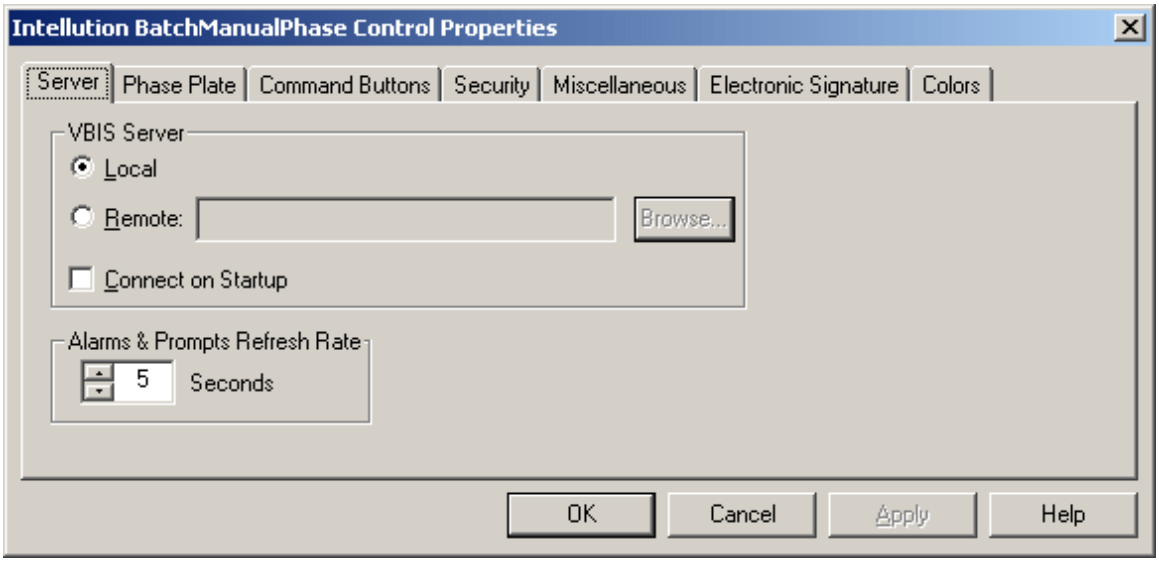
Server Property Page

Server Property Page

The Server property page:

- Lets you configure the VBIS Server settings, such as whether the VBIS Server is local or remote, for the BatchManualPhase control.
- Is available at design time and may be enabled at run time using the Security property page.

The following figure shows the Server property page.



VBIS Server Property Page

The following table lists the items that are available on the Server property page.

BatchManualPhase Control Server Property Page		
Item	**Description**	**Associated Property**
Local button	When selected, indicates that the VBIS Server is running on the same computer as the BatchManualPhase control.	VBISServerName

BatchManualPhase Control Server Property Page		
Item	**Description**	**Associated Property**
Remote button	When selected, indicates that the VBIS Server is running on a different computer from the BatchManualPhase control.	VBISServerName
Remote field	When Remote is selected, lets you specify the computer on which the VBIS Server is running.	VBISServerName
Browse button	When Remote is selected, lets you browse through the network to select the computer on which the VBIS Server is running.	None
Connect on Startup check box	When selected, the BatchManualPhase control automatically connects to the VBIS Server when the control is opened.	ConnectAtStartup
Alarms and Prompts Refresh Rate field	Lets you specify the interval, in seconds, that data in the Alarms and Operator Prompts dialog boxes is updated in the control. Valid refresh rates range from 0 to 120 seconds. If you enter 0 seconds, the control is not automatically refreshed and you must refresh the data manually. The default Refresh Rate is 5 seconds. ***NOTE:*** *On computers with 166 Megahertz or less, you may experience problems if you set the refresh rate to less than 3 seconds.*	RefreshRate

Configuring VBIS Server Settings

The steps that follow explain how to configure the VBIS Server settings for the BatchManualPhase ActiveX control.

To configure the VBIS Server settings for the BatchManualPhase control:

1. Open the Manual Phase control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Server property page.
4. Select Local if the VBIS Server is running on the same computer as the control.
5. Select Remote if the VBIS Server is running on a different computer from the control.
6. If you selected Remote, click the Browse button to select the computer name on which the

VBIS Server is running. You can also manually enter the computer name, IP address, or other DCOM-compatible computer name.

7. Select the Connect on Startup check box if you want the control to automatically connect to the VBIS Server when the control is instantiated in a run-time environment.
8. In the Alarms and Process Refresh Rate field, enter the interval, in seconds, at which you want to refresh data in the Alarms and Operator Prompts dialog boxes.
9. Click OK to save your changes.

NOTE: *You can also click the VBIS Server name in the status bar to display the property page.*

VBIS Server Properties

The following table lists the properties that control the VBIS Server settings for the BatchManualPhase control.

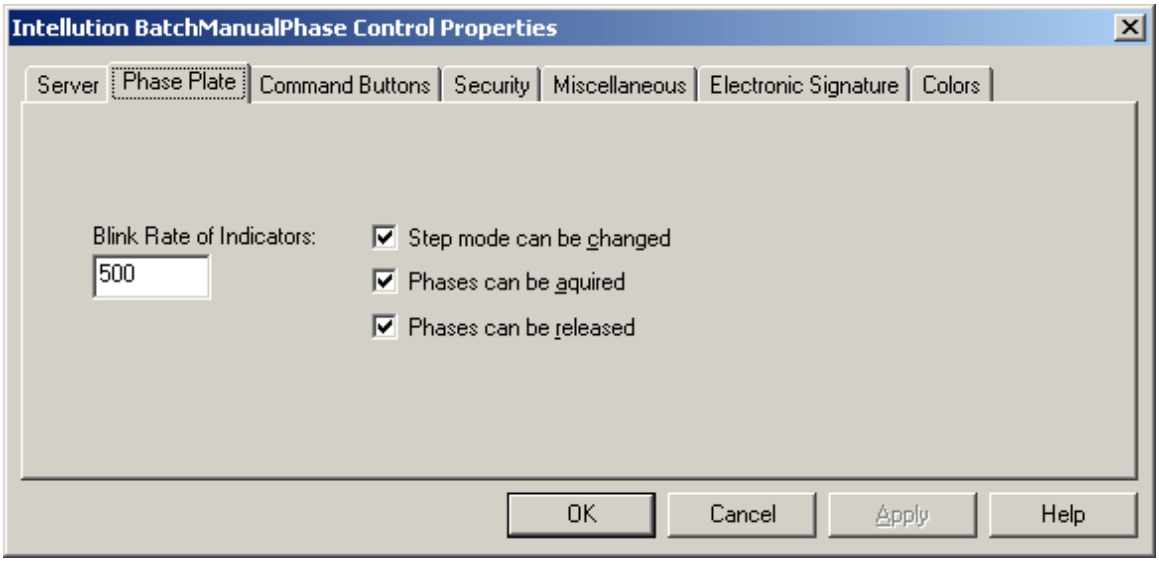
BatchManualPhase Control Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetConnectAtStartup(); **void** CBatchManualPhase::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default Refresh Rate is 5 seconds.	**C++ Syntax:** **short** CBatchManualPhase::GetRefreshRate(); **void** CBatchManualPhase::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchManualPhase::GetVBISServerName(); **void** CBatchManualPhase::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

Phase Plate Property Page

The Phase Plate property page:

- Lets you configure the blinking rate for the indicator lights; and whether or not the operator can change the step mode, acquire a phase, or release a phase.
- Is available at design time and may be enabled at run time using the Security property page.

The following figure shows the Phase Plate property page.



Phase Plate Property Page

The following table lists the items that are available on the Phase Plate property page.

BatchManualPhase Control Phase Plate Property Page		
Item	**Description**	**Associated Property**
Blink Rate of Indicators field	The blink rate, in milliseconds, of the Paused and Failure indicator lights. Valid refresh rates are greater than 0 milliseconds (ms). The default is 500 ms.	IndicatorBlinkRate
Step mode can be Changed check box	When selected, allows the operator to change the step mode.	AllowStepModeChange
Phases can be Acquired check box	When selected, allows the operator to acquire a phase.	AllowPhaseAcquire

BatchManualPhase Control Phase Plate Property Page		
Item	**Description**	**Associated Property**
Phases can be Released check box	When selected, allows the operator to release a phase.	AllowPhaseRelease

Configuring Phase Plate Settings

The steps that follow explain how to configure the Phase Plate settings for the BatchManualPhase ActiveX control.

To configure the Phase Plate settings for the BatchManualPhase control:

1. Open the BatchManualPhase control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Phase Plate property page.
4. Enter the blink rate interval, in milliseconds, for the Paused and Failure indicators.
5. Select Step Mode can be Changed to allow the operator to change the step mode; deselect it to disable the Step Mode check box in the control.
6. Select Phases can be Acquired to allow the operator to acquire a phase; deselect it to disable the Acquire button on the control. The Release button is not affected by this check box setting.
7. Select Phases can be Released to allow the operator to acquire a phase; deselect it to disable the Release button on the control. The Acquire button is not affected by this check box setting.
8. Click OK to save your changes.

Phase Plate Properties

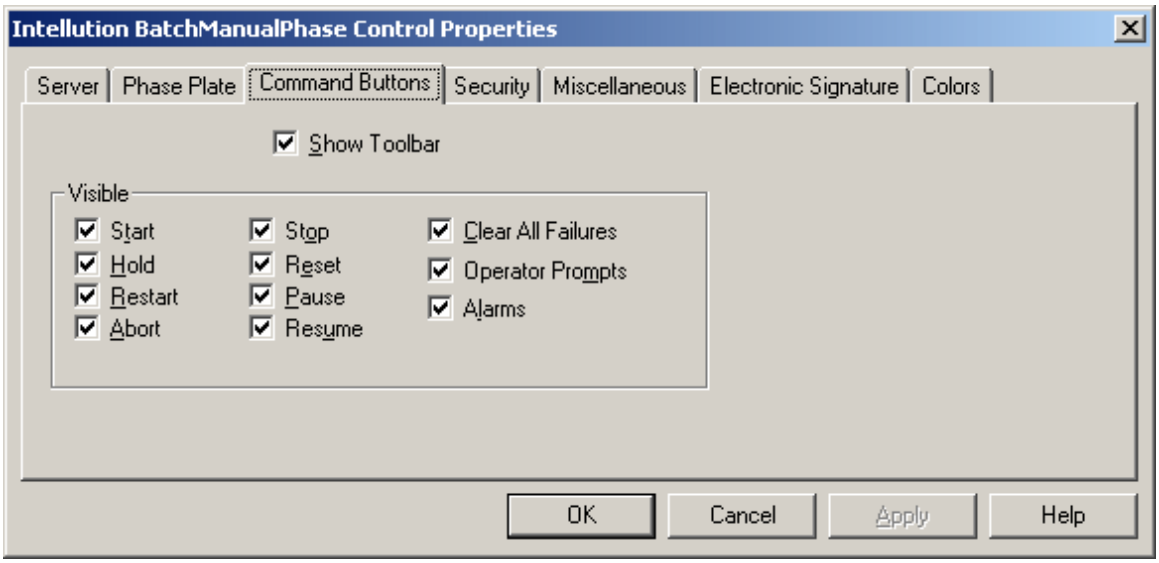
The following table lists the properties that control the Phase Plate settings for the BatchManualPhase control.

BatchManualPhase Control Phase Plate Properties	
Property	**Syntax**
AllowPhaseAcquire Sets whether or not the operator can acquire the phase.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAllowPhaseAcquire(); **void** CBatchManualPhase::SetAllowPhaseAcquire(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowPhaseAcquire[= *boolvalue*]
AllowPhaseRelease Sets whether or not the operator can release the phase.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAllowPhaseRelease(); **void** CBatchManualPhase::SetAllowPhaseRelease(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowPhaseRelease[= *boolvalue*]
AllowStepModeChange Sets whether or not the operator can change the step mode.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAllowStepModeChange(); **void** CBatchManualPhase::SetAllowStepModeChange(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowStepModeChange[= *boolvalue*]
IndicatorBlinkRate Sets the blink rate, in milliseconds, of the Paused and Failure indicator lights.	**C++ Syntax:** **short** CBatchManualPhase::GetIndicatorBlinkRate(); **void** CBatchManualPhase::SetIndicatorBlinkRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*IndicatorBlinkRate[= *value%*]

Command Buttons Property Page

The Command Buttons property page lets the developer configure which command buttons are visible on the command toolbar. The developer can also hide the entire toolbar.

The following figure shows the Command Buttons property page.



Command Buttons Property Page

The following table lists the items that are available on the Command Buttons property page.

BatchManualPhase Control Command Buttons Property Page		
Item	**Description**	**Properties**
Show Toolbar	Selecting the check box displays the command toolbar in the BatchManualPhase control. Deselecting the check box removes the toolbar from the control.	ToolBarVisible

BatchManualPhase Control Command Buttons Property Page		
Item	**Description**	**Properties**
Start Hold Restart Abort Stop Reset Pause Resume Clear All Failures Operator Prompts Alarms	Selecting a check box displays the associated button on the command toolbar and the right-click context menu in the BatchManualPhase control. For example, selecting the Start check box lets operators use the Start button to start batches. Deselecting the check box removes the button from the toolbar.	StartButton HoldButton RestartButton AbortButton StopButton ResetButton PauseButton ResumeButton ClearAllFailuresButton OperatorPromptsButton AlarmsButton

Configuring Command Buttons Properties

The steps that follow explain how to configure the command buttons for the BatchManualPhase ActiveX control.

To configure the command buttons for the BatchManualPhase control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Command Buttons property page.
4. Select the Show ToolBar check box to display the command toolbar in the BatchManualPhase control; deselect it to remove the toolbar from the control.
5. In the Visible area, select the buttons that you want displayed on the control.
6. Click OK to save your changes.

Command Buttons Properties

The following table lists the properties that control the display of command buttons in the BatchManualPhase control.

ManualPhase Control Command Buttons Properties	
Property	**Syntax**
AbortButton Sets whether or not to display the Abort button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAbortButton (); **void** CBatchManualPhase::SetAbortButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AbortButton[= *boolvalue*]
AlarmsButton Sets whether or not to display the Alarms button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAlarmsButton(); **void** CBatchManualPhase::SetAlarmsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AlarmsButton[= *boolvalue*]
ClearAllFailuresButton Sets whether or not to display the Clear All Failures button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetClearAllFailuresButton(); **void** CBatchManualPhase::SetClearAllFailuresButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ClearAllFailuresButton[= *boolvalue*]
HoldButton Sets whether or not to display the Hold button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetHoldButton(); **void** CBatchManualPhase::SetHoldButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*HoldButton[= *boolvalue*]
OperatorPromptsButton Sets whether or not to display the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetOperatorPromptsButton(); **void** CBatchManualPhase::SetOperatorPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OperatorPromptsButton[= *boolvalue*]

ManualPhase Control Command Buttons Properties	
Property	**Syntax**
PauseButton Sets whether or not to display the Pause button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetPauseButton(); **void** CBatchManualPhase::SetPauseButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*PauseButton[= *boolvalue*]
ResetButton Sets whether or not to display the Reset button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetResetButton(); **void** CBatchManualPhase::SetResetButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ResetButton[= *boolvalue*]
RestartButton Sets whether or not to display the Restart button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetRestartButton(); **void** CBatchManualPhase::SetRestartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RestartButton[= *boolvalue*]
ResumeButton Sets whether or not to display the Resume button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetResumeButton(); **void** CBatchManualPhase::SetResumeButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ResumeButton[= *boolvalue*]
StartButton Sets whether or not to display the Start Batch button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetStartButton(); **void** CBatchManualPhase::SetStartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StartButton[= *boolvalue*]
StopButton Sets whether or not to display the Stop Batch button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetStopButton(); **void** CBatchManualPhase::SetStopButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StopButton[= *boolvalue*]

ManualPhase Control Command Buttons Properties	
Property	**Syntax**
ToolBarVisible Sets whether or not to display the command toolbar.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetToolBarVisible(); **void** CBatchManualPhase::SetToolBarVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarVisible[= *boolvalue*]

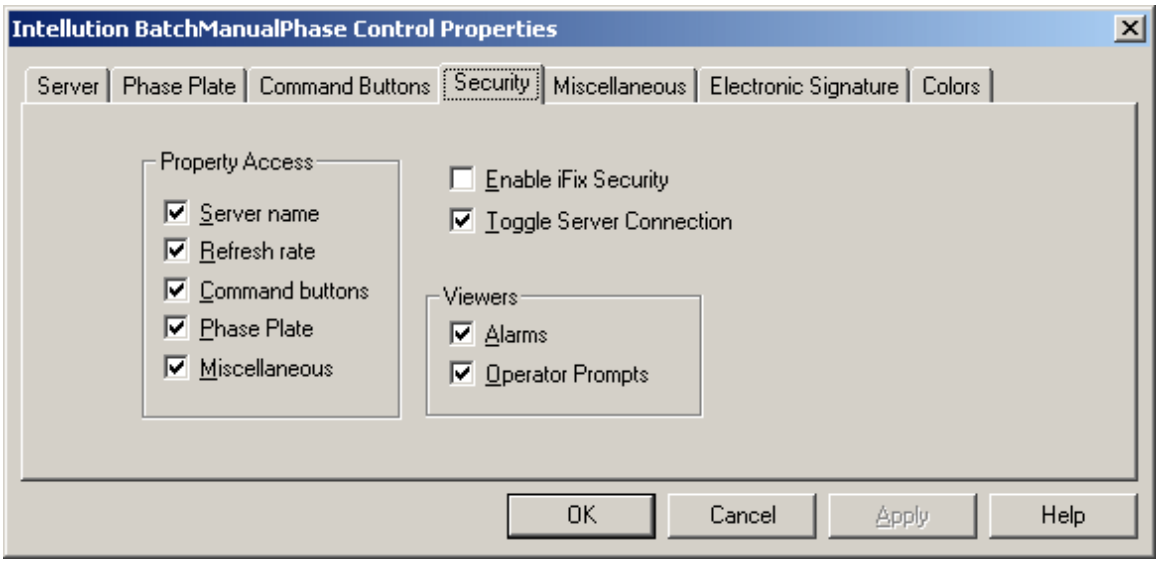
Security Property Page

The Security property page lets the developer configure the security settings, such as which properties the operator can configure, for the BatchManualPhase control. The Security property page is available only at design time. The following containers are design-time containers:

- Visual Basic
- Visual C++
- Proficy iFIX WorkSpace

If the developer turns off the refresh rate in the security property page, the refresh rate will not be changeable at run time from the Server property page dialog box. At run time, however, a developer can change the refresh rate through an automation function.

The following figure shows the Security property page.



Security Property Page

The following table lists the items that are available on the Security property page.

BatchManualPhase Control Security Property Page		
Item	**Description**	**Associated Property**
Server Name check box	When selected, lets the operator change the server name on the Server property page. When deselected, prohibits the operator from changing the server name or the Server property page at run-time.	ServerEditEnabled
Refresh Rate check box	When selected, lets the operator modify the alarms and process refresh rate on the Server property page. When deselected, data is still refreshed at the set time, but the operator cannot change the refresh rate or the Server property page at run-time.	RefreshRateEditEnabled
Command Buttons check box	When selected, lets the operator modify the settings on the Command Buttons property page at run-time.	CommandBtnsEditEnabled
Phase Plate check box	When selected, lets the operator modify the settings on the Phase Plate property page at run-time.	PhasePlateEditEnabled
Miscellaneous check box	When selected, lets the operator modify the settings on the Miscellaneous property page at run-time.	MiscEditEnabled
Toggle Server Connection check box	When selected, lets the operator connect and disconnect from the VBIS Server by selecting the Connection icon. The Connection icon appears at the bottom-right corner of the control on the status bar.	ToggleConnectionEnabled
Enable iFIX Security check box	When selected, the BatchManualPhase ActiveX control uses iFIX security to check if the current user is authorized to execute a command. ***IMPORTANT:*** *If you enabled electronic signatures for a command on the Electronic Signature tab, then iFIX security is bypassed. Windows security is checked instead. Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	EnableIFIXSecurity

BatchManualPhase Control Security Property Page		
Item	**Description**	**Associated Property**
Alarms check box	When selected, the operator can click the Display Alarms button in the BatchManualPhase control to access the View Alarms dialog box.	ViewAlarms
Operator Prompts check box	When selected, the operator can click the Operator Prompts button in the BatchManualPhase control to access the Operator Prompts dialog box.	ViewOperatorPrompts

Configuring Security Properties

The steps that follow explain how to configure the security settings for the BatchManualPhase ActiveX control.

To configure the security for the BatchManualPhase control:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Security property page.
4. In the Property Access area, select the properties you want operators to be able to modify.
5. Select the Toggle Server Connection check box if you want operators to be able to disconnect and connect from the VBIS Server.
6. In the Viewers area, select which dialog boxes you want operators to be able to access during run time.
7. Click OK to save your changes.

Security Properties

The following table lists the properties that control the security settings for properties on the BatchManualPhase control.

BatchManualPhase Control Security Properties	
Property	**Syntax**
CommandBtnsEditEnabled Sets whether or not the operator can edit the command buttons that are visible.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetCommandBtnsEditEnabled(); **void** CBatchManualPhase::SetCommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CommandBtnsEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchManualPhase::GetEnableIFIXSecurity(); **void** CBatchManualPhase::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue*]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetMiscEditEnabled(); **void** CBatchManualPhase::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MiscEditEnabled[= *boolvalue*]
PhasePlateEditEnabled Sets whether or not the operator can edit properties on the Phase Plate properties page.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetPhasePlateEditEnabled(); **void** CBatchManualPhase::SetPhasePlateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*PhasePlateEditEnabled[= *boolvalue*]

BatchManualPhase Control Security Properties	
Property	**Syntax**
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetRefreshRateEditEnabled(); **void** CBatchManualPhase::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= *boolvalue*]
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetServerEditEnabled(); **void** CBatchManualPhase::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetToggleConnectionEnabled(); **void** CBatchManualPhase::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= *boolvalue*]
ViewAlarms Sets whether or not the operator can access the View Alarms dialog box by clicking the View Alarms button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetViewAlarms(); **void** CBatchManualPhase::SetViewAlarms(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewAlarms[= *boolvalue*]
ViewOperatorPrompts Sets whether or not the operator can access the Operator Prompts dialog box by clicking the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetViewOperatorPrompts(); **void** CBatchManualPhase::SetViewOperatorPrompts(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewOperatorPrompts[= *boolvalue*]

Electronic Signature Property Page

The Electronic Signature property page specifies the electronic signature requirements associated with each of the commands available on the BatchManualPhase ActiveX control. The signature requirements define whether or not a user or users must "sign-off" on a command before it is performed. You can define the following signature types for each command:

- None (no signature required)
- Performed By (only "Performed By" signature required)
- Performed By/Verified By (both "Performed By" and "Verified By" signatures required)

If you do not define a signature type, no signature is requested when the command is performed; NONE is the default signature type if you do not select one. If you do not want to define these signature types individually, you can specify one signature type for all commands by using the default signature row.

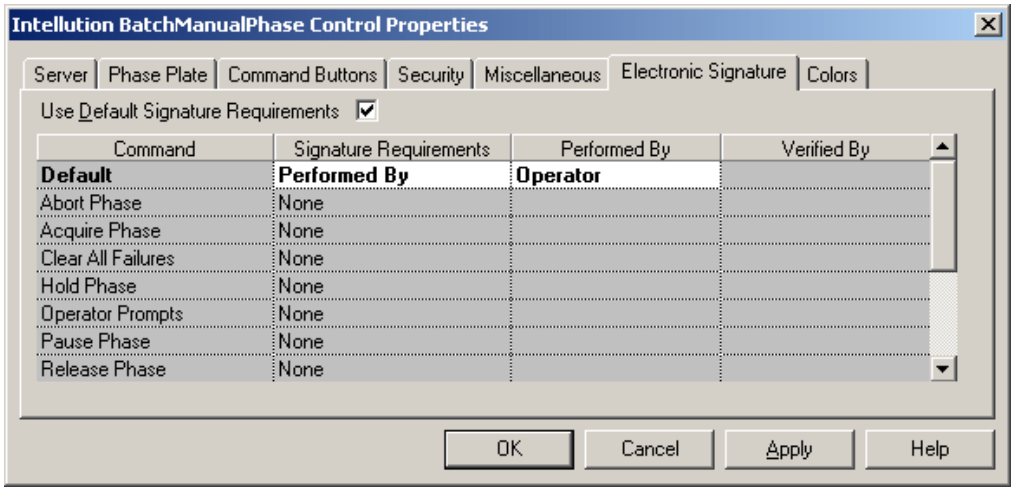
If you specified a Performed By or Performed By/Verified By signature type, Windows security groups are used to validate the user who performs the command. The user must be a member of the specified group to perform or verify the command. The same user *cannot* sign both the Performed By and Verified By signature requirements, even if that user resides in both the Performed By and Verified By groups. What that means is that the user performing a command cannot be the same user who verifies that command. Likewise, the user verifying a command cannot be the same user who performed that command.

To enter the groups from which the user is authorized, you specify the Performed By and Verified By groups in the Electronic Signature property page. This property page is enabled at design time only. The number of times the user can attempt to enter a failed signature is defined by your administrator through Windows security. Refer to account policy information in your Microsoft Windows Help system for more details.

Batch Execution supports both Local and Domain groups. Upon receiving a request to validate a signature, Batch Execution checks the local computer for the specified group. If the group is not found, it then checks for a Domain level group to verify the user name and password. You can choose to configure your security groups and user names on the Batch Execution Server computer or as part of your plant's overall Domain Security Configuration.

When the ActiveX control captures an electronic signature, the signature is recorded in the BATCH_CMD_SIGNATURE_SUCCESS table, along with the computer name and time stamp of the signature, as well as other data. If a signature fails, the event is not logged, nor is the command performed. For more information on the fields in this table refer to the BATCH_CMD_SIGNATURE_SUCCESS Table section.

The following figure shows the Electronic Signature property page for the BatchManualPhase control.



BatchManualPhase Electronic Signature Property Page

The following table lists the items that are available on the Electronic Signature property page.

Electronic Signature Property Page		
Item	**Description**	**Associated Property or Method**
Use Default Signature Requirements check box	When selected the user applies a single signature requirement for all methods on a control.	UseDefaultSignatureRequirements
Signature Requirements Grid Control	Contains the signature requirements for each command or the default.	GetSignatureRequirements SetSignatureRequirements

Configuring Electronic Signature Properties

The steps that follow explain how to configure the electronic signature settings for the BatchManualPhase ActiveX control.

NOTE: *You can only configure the electronic signature properties in design mode. You cannot configure these properties in run mode.*

To configure the electronic signature properties for BatchManualPhase:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Electronic Signature property page.
4. Select the Use Default Signature Requirements check box if you want to use a default signature.

5. Select the signature type for the default, or for each command listed (if default is not selected):
    - Performed By
    - Performed By / Verified By
    - None
6. Select the Windows security group for each Performed By and Verified By signature specified.

    NOTE: *Right-click on the edit box and select the Browse option to open the Windows Security Groups dialog box. Make sure that the cursor is not displaying in the text box when you right-click on it. Select a group from the Windows Security Groups dialog box and click OK. You can only browse local security groups, but you can manually enter a domain group.*

7. Click OK to save your changes.

Electronic Signature Properties

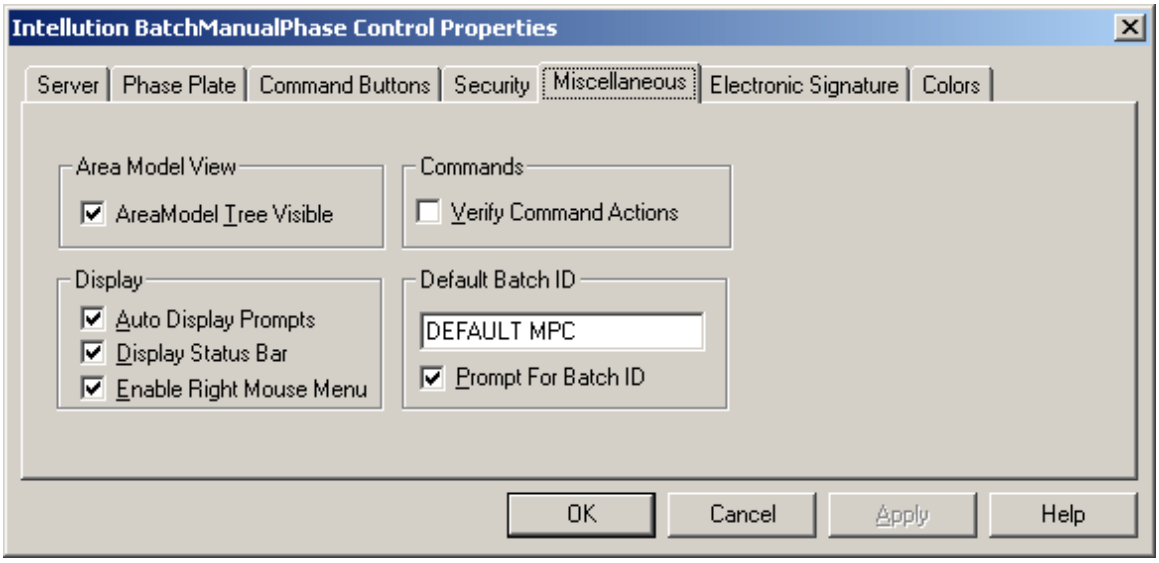
The following table lists the properties that control the electronic signature settings for the BatchManualPhase control.

BatchManualPhase Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements If this property is True, it indicates that the default signature settings will be used for all commands for the ActiveX control. If this property is False, then the signature setting of each command is used. The default value of UseDefaultSignatureRequirements is True. The default signature requirements are None. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchManualPhase:GetUseDefaultSignatureRequirements();**void** CBatchManualPhase::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

Miscellaneous Property Page

The Miscellaneous property page lets the developer configure a variety of settings for the BatchManualPhase control, such as whether or not to display the status bar and whether to display the area model tree view.

The Miscellaneous property page is available at design time and may be enabled at run time using the Security property page. The following figure shows the Miscellaneous property page for the BatchManualPhase control.



Miscellaneous Property Page

The following table lists the items that are available on the Miscellaneous property page.

BatchManualPhase Control Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Auto Display Prompts check box	When selected, the Operator Prompts dialog box automatically displays when there is an operator prompt for the selected phase.	AutoDisplayPromptsEnabled
Display Status Bar check box	When selected, the status bar is visible; when deselected, the status bar does not appear in the BatchManualPhase control.	StatusBar
Enable Right Mouse Menu check box	When selected, the operator can right-click the running ActiveX control to view a right-click menu. If you remove the checkmark, the right mouse menu does not display when right clicking the running ActiveX control.	EnableRightMouseMenu
Tree View check box	When selected, the area model tree view is displayed to the left of the phase plate.	AreaModelTreeVisible
Verify Command Actions check box	When selected, the operator is prompted for confirmation when executing a command.	VerifyCommandActions

BatchManualPhase Control Miscellaneous Property Page		
Item	**Description**	**Associated Property**
Default Batch ID field	The Batch ID to be automatically displayed in the Start Phase dialog box.	DefaultBatchID
Prompt for Batch ID check box	When selected, the Start Phase dialog box is displayed when the operator starts a phase.	PromptForBatchID

Configuring Miscellaneous Properties

The steps that follow explain how to configure the miscellaneous properties for the BatchManualPhase ActiveX control.

To configure the miscellaneous properties for the BatchManualPhase control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Miscellaneous property page.
4. In the Display area, specify if you want to display the Operator Prompts dialog box and if you want the status bar to be displayed at the bottom of the control.
5. In the Area Model View area, select area model view tree check box to have the tree appear in the BatchManualPhase control.
6. Select the Verify Command Actions check box to prompt operators to verify commands before they are executed.
7. Enter the default batch ID, and specify if the Start Phase dialog box should automatically be displayed when the operator starts a phase.
8. Click OK to save your changes.

Miscellaneous Properties

The following table lists the properties that control the miscellaneous settings for the BatchManualPhase control.

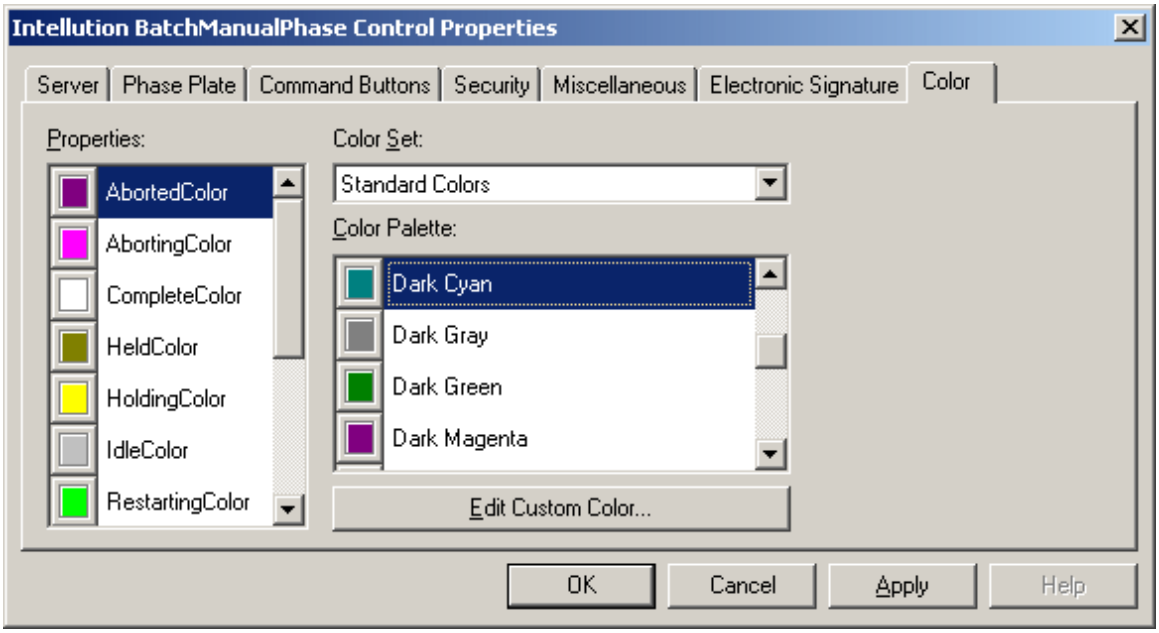
BatchManualPhase Control Miscellaneous Properties

Property	Syntax
AreaModelTreeVisible Sets whether the area model tree view displays in the BatchManualPhase control.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAreaModelTreeVisible (); **void** CBatchManualPhase::SetAreaModelTreeVisible (**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AreaModelTreeVisible[= *boolvalue*]
AutoDisplayPromptsEnabled Sets whether or not to automatically display the Operator Prompts dialog box when there is an operator prompt for the selected phase.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetAutoDisplayPromptsEnabled (); **void** CBatchManualPhase::SetAutoDisplayPromptsEnabled (**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AutoDisplayPromptsEnabled[= *boolvalue*]
DefaultBatchID Sets whether or not to automatically display the Batch ID in the Start Phase dialog box.	**C++ Syntax:** **CString** CBatchManualPhase::DefaultBatchID (); **void** CBatchManualPhase::DefaultBatchID (**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*DefaultBatchID[= *text$*]
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetEnableRightMouseMenu(); **void** CBatchManualPhase::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableRightMouseMenu[= *boolvalue*]
PromptForBatchID Sets whether or not to display the Start Phase dialog box when the operator starts a phase.	**C++ Syntax:** **BOOL** CBatchManualPhase::PromptForBatchID (); **void** CBatchManualPhase::PromptForBatchID (**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*PromptForBatchID[= *boolvalue*]

BatchManualPhase Control Miscellaneous Properties	
Property	**Syntax**
StatusBarVisible Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetStatusBarVisible(); **void** CBatchManualPhase::SetStatusBarVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StatusBarVisible[= *boolvalue*]
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when executing a command.	**C++ Syntax:** **BOOL** CBatchManualPhase::GetVerifyCommandActions(); **void** CBatchManualPhase::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.VerifyCommandActions[= *boolvalue*]

BatchManualPhase Control Colors Property Page

The Colors property page lets you set the color for the BatchManualPhase control states. You can choose a color from the displayed color chart or you can use the System Color drop-down list box to display and select from a pre-defined system color chart. The Colors property page, shown in the following figure, is available at design time and at run time.



Colors Property Page

The following table lists the items that are available on the Colors property page.

BatchManualPhase Control Colors Property Page		
Item	**Description**	**Corresponding Properties**
Properties list box	Lists all the available properties for which you can set the color.	AbortedColor AbortingColor CompleteColor HeldColor HoldingColor IdleColor RestartingColor RunningColor StartingColor StoppedColor StoppingColor
Color Set drop-down list	List the colors sets that you can choose from. For example: Windows Standard Colors, Windows System Colors.	None
Color Palette list box	Displays the available colors that you can assign to a property. Double-click the <Custom> option to display the Color dialog box and add a custom color to the Color Palette list.	None
Edit Custom Color button	Click this button to display the Color dialog box and add a custom color to the Color Palette list.	None

Setting Colors

The steps that follow explain how to configure the colors for the BatchManualPhase ActiveX control.

To set the colors for the BatchManualPhase control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Colors property page.
4. Select a property from the Property Name drop-down list box.

5. Click a color in the color palette or select a pre-defined system color from the System Color drop-down list box.

Colors Properties

The following table lists the properties that control the colors in the BatchManualPhase control.

BatchManualPhase Control Colors Properties	
Property	**Syntax**
AbortedColor Sets the color of the Aborted phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetAbortedColor(); **void** CBatchManualPhase::SetAbortedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AbortedColor[= *color%*]
AbortingColor Sets the color of the Aborting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetAbortingColor(); **void** CBatchManualPhase::SetAbortingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AbortingColor[= *color%*]
CompleteColor Sets the color of the Complete phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetCompleteColor(); **void** CBatchManualPhase::SetCompleteColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CompleteColor[= *color%*]
HeldColor Sets the color of the Held phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetHeldColor(); **void** CBatchManualPhase::SetHeldColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeldColor[= *color%*]

BatchManualPhase Control Colors Properties	
Property	**Syntax**
HoldingColor Sets the color of the Holding phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetHoldingColor(); **void** CBatchManualPhase::SetHoldingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HoldingColor[= *color%*]
IdleColor Sets the color of the Idle phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetIdleColor(); **void** CBatchManualPhase::SetIdleColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*IdleColor[= *color%*]
RestartingColor Sets the color of the Restarting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetRestartingColor(); **void** CBatchManualPhase::SetRestartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RestartingColor[= *color%*]
RunningColor Sets the color of the Running phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetRunningColor(); **void** CBatchManualPhase::SetRunningColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RunningColor[= *color%*]
StartingColor Sets the color of the Starting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetStartingColor(); **void** CBatchManualPhase::SetStartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StartingColor[= *color%*]

BatchManualPhase Control Colors Properties	
Property	**Syntax**
StoppedColor Sets the color of the Stopped phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetStoppedColor(); **void** CBatchManualPhase::SetStoppedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StoppedColor[= *color%*]
StoppingColor Sets the color of the Stopping phase state.	**C++ Syntax:** **OLE_COLOR** CBatchManualPhase::GetStoppingColor(); **void** CBatchManualPhase::SetStoppingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StoppingColor[= *color%*]

Color Property Examples

The following show examples for setting color properties.

C++ Example

```
OLE_COLOR StartingColor = pManualPhase->GetStartingColor();
OLE_COLOR newStartingColor = 0x0; // black
pManualPhase->SetStartingColor(newStartingColor);
```

Visual Basic Example

```
Static MP_OriginalStartingColor As Long
MP_OriginalStartingColor = ManualPhaseControl1.StartingColor
     ' get the current color
ManualPhaseControl1.StartingColor = &H808080
     ' set to dark gray
```

BatchManualPhase Control Methods

The BatchManualPhase control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- GetIVBIS Method
- GetPhaseData Method
- RunCommand Method

- SetCurrentPhase Method
- SetIVBISPointer Method
- ToggleStepMode
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

ConnectToServer Method

Description

Establishes the connection to the VBIS Server.

C++ Syntax

```
BOOL CBatchManualPhase::ConnectToServer();
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.ConnectToServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if a connection is made.
- 0 if a connection is not made.

C++ Example

```
BOOL result = pManualPhase->ConnectToServer();
```

Visual Basic Example

```
[form.]ManualPhaseControl1.ConnectToServer
```

DisconnectFromServer Method

Description

Disconnects from the currently connected VBIS Server.

C++ Syntax

```
BOOL CBatchManualPhase::DisconnectFromServer();
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.DisconnectFromServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the VBIS Server is successfully disconnected.
- 0 if the VBIS Server is not successfully disconnected.

C++ Example

```
BOOL result = pManualPhase->DisconnectFromServer();
```

Visual Basic Example

```
[form.]ManualPhaseControl1.DisconnectFromServer
```

GetIVBIS Method

Description

Returns the IVBIS pointer to which the control is connected. If the control is not connected, it returns NULL. Call Release () on the pointer when done with it.

C++ Syntax

```
LPDISPATCH CBatchManualPhase::GetIVBIS();
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.GetIVBIS() As Objects
```

C++ Example

```
IVBIS8* pIVBIS = pManualPhase->GetIVBIS();
...
///Release it when done.
pIVBIS->Release();
```

Visual Basic Example

```
Set VBISP = ManualPhaseControl1.GetIVBIS
...
Set VBISP = Nothing
```

GetPhaseData Method

Description

Gets information about the current phase from the VBISPhase2 object.

C++ Syntax

```
LPDISPATCH CBatchManualPhase::GetPhaseData();
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.GetPhaseData() As Variant
```

Parameters

None.

Return Type

- In C++, the return type is LPDispatch; this contains a VBISPhase2 pointer.
- In Visual Basic, the return type is Object.

C++ Example

```
VBISPhase2* pPhase2=NULL;
BSTR bsCurrPhaseName;
CString strPhaseName;

pPhase2 = ManualPhaseControl1.GetPhaseData();
if (pPhase2 != NULL)
{
     pPhase2->get_Name( &bsCurrPhaseName );
     strPhaseName = bsCurrPhaseName;
     ::SysFreeString( bsCurrPhaseName );
     AfxMessageBox(strPhaseName);
     pPhase2->Release();
}
```

Visual Basic Example

```
Dim pPhase2 As Variant
Set pPhase2 = ManualPhaseControl1.GetPhaseData
If (pPhase2 Is Nothing) Then
MsgBox "No Phase Selected"
Else
```

```
    MsgBox "Phase = " + pPhase2.Name
    Set pPhase2 = Nothing
End If
```

RunCommand Method

Description

Executes a command against the selected phase.

C++ Syntax

```
LONG RunCommand(long command);
```

Visual Basic Syntax

```
Object.RunCommand(command As Long) As Long
```

Parameters

Parameter	Description
command	One of the following commands: 0 = START_PHASE 1 = HOLD_PHASE 2 = RESTART_PHASE 3 = ABORT_PHASE 4 = STOP_PHASE 5 = RESET_PHASE 6 = PAUSE_PHASE 7 = RESUME_PHASE 8 = CLEAR_ALL_FAILURES_PHASE 9 = ACQUIRE_PHASE 10 = RELEASE_PHASE

Return Type

The function returns the following error codes:

- 0 = SUCCESS
- 1001 = COMMANDNOTSUPPORTED
- 1003 = COMMANDNOTVALID

C++ Example

```
LONG status = pManualPhase->RunCommand(0);
```

Visual Basic Example

```
Dim status as Long
Status = ManualPhaseControl1.RunCommand(0)
```

SetCurrentPhase Method

Description

This method searches for a phase and sets the control to select it. The method displays the phase information in the control for the phase that is passed in by this method. This method also accepts a unit name or process cell name and sets the control to have that item selected. If the phase, unit, or process cell is not found, then the currently viewed item remains selected.

C++ Syntax

```
long CBatchManualPhase::SetCurrentPhase(LPCTSTR lpstrPhaseName);
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.SetCurrentPhase(lpstrPhaseName as String) As Boolean
```

Parameters

Parameter	Description
lpstrPhaseName	Name of the phase to display.

Return Type

Long.

- 0 = SUCCESS
- 1001 = BATCHXLIST_E_COMMANDNOTSUPPORTED
- 1002 = BATCHXLIST_E_DATAERROR, phase name not found
- 1003 = BATCHXLIST_E_COMMANDNOTVALID

SetIVBISPointer Method

Description

Sets the internal VBIS pointer of the control to the given VBIS pointer. VBIS Server name is also passed, but only for display information. Make sure that the VBIS Server name is correct.

C++ Syntax

```
BOOL CManualPhase::SetIVBISPointer(IDispatch* pIVBIS,
CString lpstrServerName);
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.SetIVBISPointer(pIVBIS As Object lpstrServerName As String) As
Boolean
```

Parameters

Parameter	Description
pIVBIS	The pointer to the VBIS Server.
lpstrServerName	The name of the VBIS Server.

Return Type

Boolean.

- 1 if successful.
- 0 if not successful.

C++ Example

```
CString VBISServerName = pManualPhase->GetVBISServerName();
// get pIVBIS
IVBIS8* pIVBIS=pManualPhase->GetIVBIS();
pManualPhase->SetIVBISPointer(pIVBISP, VBISServerName);
pIVBIS->Release();
```

Visual Basic Example

```
Dim VBISServerName as String
VBISServerName = ManualPhaseControl1.VBISServerName
Set VBISP=ManualPhaseControl1.GetIVBIS
' get VBISP
ManualPhaseControl2.SetIVBISPointer VBISP, VBISServerName
Set VBISP = Nothing
```

NOTE: *These examples set the IVBIS pointer from one control and use it to set the IVBIS pointer of the second control.*

ToggleStepMode

Description

This control toggles the Step Mode check box in the Phase Plate. If the check box is disabled, it is ignored.

C++ Syntax

```
BOOL CBatchManualPhase::ToggleStepMode()
```

Visual Basic Syntax

```
[form.]ManualPhaseControl1.ToggleStepMode() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the step is successfully toggled.
- 0 if the step is not successfully toggled.

C++ Example

```
BOOL result = pManualPhase->ToggleStepMode();
```

Visual Basic Example

```
[form.]ManualPhaseControl1.ToggleStepMode
```

GetCommandSignatureRequirements Method

Description

Gets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchManualPhase::GetCommandSignatureRequirements(long Command, long*
SignatureRequirements, BSTR* PerformedBy, BSTR* VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.GetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchManualPhase the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortPhase = 1 • bcAcquirePhase = 2 • bcClearAllFailures = 3 • bcHoldPhase = 4 • bcOperatorPrompts = 5 • bcPausePhase = 6 • bcReleasePhase = 7 • bcResetPhase = 8 • bcRestartPhase = 9 • bcResumePhase = 10 • bcStartPhase = 11 • bcStepMode = 12 • bcStopPhase = 13
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

typedef enum _tagCommandID{
     bcDefault = 0,
     bcAbortPhase = 1,
     bcAcquirePhase = 2,
     bcClearAllFailures = 3,
     bcHoldPhase = 4,
     bcOperatorPrompts = 5,
     bcPausePhase = 6,
     bcReleasePhase = 7,
     bcResetPhase = 8,
     bcRestartPhase = 9,
     bcResumePhase = 10,
     bcStartPhase = 11,
     bcStepMode = 12,
     bcStopPhase = 13
} COMMANDID;


     BSTR bstrPerformedBy;
     BSTR bstrVerifiedBy;
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     // Example of reading the signature requirements for the
     // Acquire Phase command.
     // Note, that if the UseDefaultSignatureRequirements is TRUE
     // then the setting for the Default command is used,
     // else the setting for the start batch command is used.

     if (m_pBatchManualPhase->GetUseDefaultSignatureRequirements ())
     {
          lCommand = bcDefault;
     }
     else
     {
          lCommand = bcAcquirePhase;
```

```
     }

     m_pBatchManualPhase->GetCommandSignatureRequirements ( lCommand, &lSignatureType,
&bstrPerformedBy, &bstrVerifiedBy);

     strPerformedBy = bstrPerformedBy;
     strVerifiedBy = bstrVerifiedBy;

     ::SysFreeString(bstrPerformedBy);
     ::SysFreeString(bstrVerifiedBy);

     if (lSignatureType == stNone)
     {
     AfxMessageBox ("No Signature requirements for the Acquire Phase Command");
     }
     else if (lSignatureType == stPerformedBy)
     {
     AfxMessageBox ("Signature requirements for the Acquire Phase Command are Perform By:
" + strPerformedBy);
     }
     else if (lSignatureType == stPerformedByVerifiedBy)
     {
     AfxMessageBox ("Signature requirements for the Acquire Phase Command are Perform By:
" + strPerformedBy + " Verify By: " + strVerifiedBy);
     }
```

Visual Basic Example

```
Private Sub Command2_Click()
Dim Command As MANUALPHASELib.COMMANDID
Dim SignatureType As MANUALPHASELib.SignatureType
Dim bRetVal As Boolean
Dim strPerformedBy As String
Dim strVerifiedBy As String
' example of reading the signature requirements for the Start phase Command
' note that if the UseDefaultSignatureRequirements is TRUE then the setting for the
Default command is used,
' else the setting for the start batch command is used.

If BatchList1.UseDefaultSignatureRequirements Then
    Command = bcDefault
Else
    Command = bcStartPhase
End If
BatchManualPhase1.GetCommandSignatureRequirements Command, SignatureType, strPerformedBy,
strVerifiedBy

If SignatureType = stNone Then
    MsgBox ("No Signature requirements for the Start Phase Command")
ElseIf SignatureType = stPerformedBy Then
    MsgBox ("Signature requirements for the Start Phase Command are Perform By: " +
strPerformedBy)
ElseIf SignatureType = stVerifiedBy Then
    MsgBox ("Signature requirements for the Start Phase Command are Perform By: " +
strPerformedBy + " Verified By: " + strVerifiedBy)
End If
```

SetCommandSignatureRequirements Method

Description

Sets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchManualPhase::SetCommandSignatureRequirements(Long Command, Long
SignatureRequirements, CString PerformedBy, CString VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.SetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchManualPhase the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortPhase = 1 • bcAcquirePhase = 2 • bcClearAllFailures = 3 • bcHoldPhase = 4 • bcOperatorPrompts = 5 • bcPausePhase = 6 • bcReleasePhase = 7 • bcResetPhase = 8 • bcRestartPhase = 9 • bcResumePhase = 10 • bcStartPhase = 11 • bcStepMode = 12 • bcStopPhase = 13

Parameter	Description
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

typedef enum _tagCommandID{
     bcDefault = 0,
     bcAbortPhase = 1,
     bcAcquirePhase = 2,
     bcClearAllFailures = 3,
     bcHoldPhase = 4,
     bcOperatorPrompts = 5,
     bcPausePhase = 6,
     bcReleasePhase = 7,
     bcResetPhase = 8,
     bcRestartPhase = 9,
     bcResumePhase = 10,
     bcStartPhase = 11,
     bcStepMode = 12,
     bcStopPhase = 13
} COMMANDID;
```

```
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     strPerformedBy = _T("Operator");
     strVerifiedBy = _T("Supervisor");

     lCommand = bcAcquirePhase;
     lSignatureType = stPerformedByVerifiedBy;
     m_pBatchManualPhase ->SetCommandSignatureRequirements( lCommand, lSignatureType,
strPerformedBy, strVerifiedBy );
```

Visual Basic Example

```
' example to set a specific command signature requirement

Dim Command As MANUALPHASELib.COMMANDID
Dim SignatureType As MANUALPHASELib.SignatureType
Dim bRetVal As Boolean
Dim strPerformedBy As String
Dim strVerifiedBy As String

Command = bcStopPhase
strPerformedBy = "Operator"
strVerifiedBy = "Supervisor"
ManualPhase1.SetCommandSignatureRequirements Command, stPerformedByVerifiedBy,
strPerformedBy, strVerifiedBy
End Sub
```

BatchManualPhase Control Events

The BatchManualPhase control generates the following events:

- AbortPhasePressed Event
- ConnectedToServer Event
- ClearAllFailuresPressed Event
- DisconnectedFromServer Event
- HoldPhasePressed Event
- PausePhasePressed Event
- PhaseAcquirePressed Event
- PhaseReleasePressed Event
- ResetPhasePressed Event
- RestartPhasePressed Event
- ResumePhasePressed Event
- ServerChanged Event
- StartPhasePressed Event

- StepModeChangedPressed Event
- StopPhasePressed Event

AbortPhasePressed Event

Description

Occurs when the operator issues an Abort Phase command.

Event ID

4

C++ Syntax

```
void AbortPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event AbortPhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

ConnectedToServer Event

Description

Occurs when the control has connected to the VBIS Server.

Event ID

14

C++ Syntax

```
void ConnectedToServer();
```

Visual Basic Syntax

```
Event ConnectedToSever()
```

ClearAllFailuresPressed Event

Description

Occurs when the operator issues a ClearAllFailures command.

Event ID

9

C++ Syntax

```
void ClearAllFailuresPressed(long phaseID, CString strPhaseName, CString strParentUnit,
CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event ClearAllFailuresPressed(phaseID as Long, strPhaseName as String, strParentUnit as
String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

DisconnectedFromServer Event

Description

Occurs when the control has disconnected from the VBIS Server.

Event ID

15

C++ Syntax

```
void OnDisconnectedFromServer();
```

Visual Basic Syntax

```
Event DisconnectedFromServer()
```

HoldPhasePressed Event

Description

Occurs when the operator issues a HoldPhase command.

Event ID

2

C++ Syntax

```
void HoldPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString
strOwnerName);
```

Visual Basic Syntax

```
Event HoldPhasePressed(ByVal phaseID as Long, ByVal strPhaseName as String, strParentUnit
as String, ByVal strOwnerName as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerName	The ID of the batch that owns the phase.

PausePhasePressed Event

Description

Occurs when the operator issues a PausePhase command.

Event ID

7

C++ Syntax

```
void PausePhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event PausePhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

PhaseAcquirePressed Event

Description

Occurs when the operator issues a PhaseAcquire command.

Event ID

11

C++ Syntax

```
void PhaseAcquirePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID, BOOL bStepMode);
```

Visual Basic Syntax

```
Event PhaseAcquirePressed(phaseID as Long, strPhaseName as String, ParentUnit as String,
OwnerBatchID as String, bStepMode as boolean)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.
bStepMode	Flag that indicates whether the phase is in manual or auto mode.

PhaseReleasePressed Event

Description

Occurs when the operator issues a PhaseRelease command.

Event ID

12

C++ Syntax

```
void PhaseReleasePressed(long phaseID, CString strPhaseName, CString strParentUnit,
CString strOwnerBatchID, BOOL bStepMode);
```

Visual Basic Syntax

```
Event PhaseReleasePressed(phaseID as Long, strPhaseName as String, ParentUnit as String,
OwnerBatchID as String, bStepMode as boolean)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.

Parameter	Description
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.
bStepMode	Flag that indciates wheter the phase is in manual or auto mode.

ResetPhasePressed Event

Description

Occurs when the operator issues a ResetPhase command.

Event ID

6

C++ Syntax

```
void ResetPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event ResetPhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

RestartPhasePressed Event

Description

Occurs when the operator issues a RestartPhase command.

Event ID

3

C++ Syntax

```
void RestartPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit,
CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event RestartPhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as
String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

ResumePhasePressed Event

Description

Occurs when the operator issues a ResumePhase command.

Event ID

8

C++ Syntax

```
void ResumePhasePressed(long phaseID, CString strPhaseName, CString strParentUnit,
CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event ResumePhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as
String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

ServerChanged Event

Description

Occurs when the control has switched VBIS Servers.

Event ID

13

C++ Syntax

```
void OnServerChanged();
```

Visual Basic Syntax

```
Event ServerChanged()
```

StartPhasePressed Event

Description

Occurs when the operator issues a StartPhase command.

Event ID

1

C++ Syntax

```
void StartPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event StartPhasePressed(phaseID as Long, strPhaseName as String, strParentUnit as String, strOwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.

StepModeChangedPressed Event

Description

Occurs when the operator issues a StepModeChanged command.

Event ID

10

C++ Syntax

```
void StepModeChangedPressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID, BOOL bStepMode);
```

Visual Basic Syntax

```
Event StepModeChangedPressed(phaseID as Long, strPhaseName as String, strParentUnit as String, strOwnerBatchID as String, bStepMode as boolean)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.
bStepMode	Flag that indicates whether the batch is in manual or auto mode.

StopPhasePressed Event

Description

Occurs when the operator issues a StopPhase command.

Event ID

5

C++ Syntax

```
void StopPhasePressed(long phaseID, CString strPhaseName, CString strParentUnit, CString strOwnerBatchID);
```

Visual Basic Syntax

```
Event StopPhasePressed(phaseID as Long, strPhaseName as String, ParentUnit as String, OwnerBatchID as String)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.

Parameter	Description
strOwnerBatchID	The ID of the batch that owns the phase.

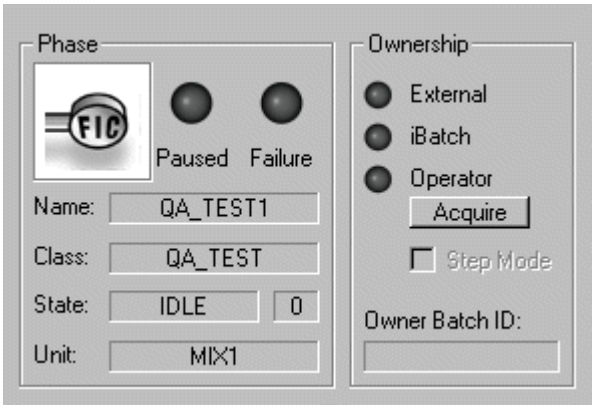
C++ Event Sink Map

The following shows an example of an event sink map.

```
BEGIN_EVENTSINK_MAP(CManualPhaseDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CManualPhaseDlg)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 1 /* StartPhasePressed */,
OnStartPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 2 /* HoldPhasePressed */,
OnHoldPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 3 /* RestartPhasePressed */,
OnRestartPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 4 /* AbortPhasePressed */,
OnAbortPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 5 /* StopPhasePressed */,
OnStopPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 6 /* ResetPhasePressed */,
OnResetPhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 7 /* PausePhasePressed */,
OnPausePhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 8 /* ResumePhasePressed */,
OnResumePhasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 9 /* ClearAllFailuresPressed
*/, OnClearAllFailuresPressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 10 /* StepModeChangePressed */,
OnStepModeChangePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR VTS_I4)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 11 /* PhaseAcquirePressed */,
OnPhaseAcquirePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 12 /* PhaseReleasePressed */,
OnPhaseReleasePressedBatchmanualphasectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 13 /* ServerChanged */,
OnServerChangedBatchmanualphasectrl1, VTS_NONE)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 14 /* ConnectedToServer */,
OnConnectedToServerBatchmanualphasectrl1, VTS_NONE)
     ON_EVENT(CManualPhaseDlg, IDC_BATCHMANUALPHASECTRL1, 15 /* DisconnectedFromServer
*/, OnDisconnectedFromServerBatchmanualphasectrl1, VTS_NONE)
     //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()
```

Using the BatchPhasePlate Control

The BatchPhasePlate ActiveX control allows you to program your own manual phase control. It provides properties and methods for displaying phase data within the phase plate. It also provides events for operator actions such as acquire, release, and step mode selected. The following figure shows an example of the BatchPhasePlate ActiveX control.



BatchPhasePlate Control

The BatchPhasePlate contains two items that the operator can use to control the phase:

- Acquire/Release button
- Step Mode check box

The Acquire/Release button displays Acquire on the button face if the current phase is not active, else it displays Release. Clicking on the Acquire button blinks the Operator indicator and fires the OwnerButtonClick (True) event. See the OwnerButtonClick Event section for more details. Clicking on the Release button fires the OwnerButtonClick (False) event. Selecting the Step Mode check box fires the StepModeChanged event. Refer to the StepModeChanged Event section for more details. As a developer, you need to process the acquire, release, and step mode changed events.

BatchPhasePlate Control Properties

The BatchPhasePlate control provides property pages that you can use to view and change the control's properties. There are also some properties that you can use that are not on the property pages. The following sections describe each property for the BatchPhasePlate control. The properties are grouped into the following categories:

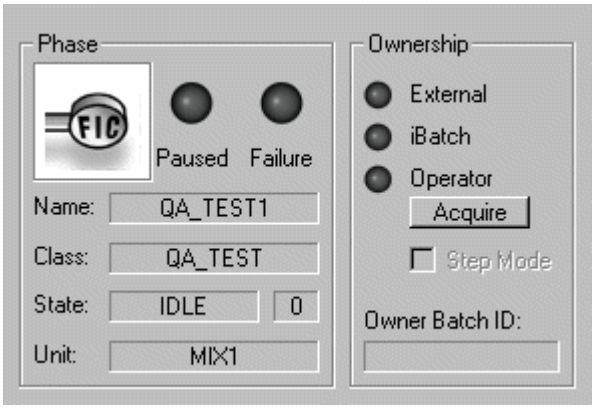
- Run-time phase properties
- Phase Plate properties
- Colors properties

Run-time Phase Properties

The run-time phase properties allow you to set the phase name, class, state, step index, parent unit, batch ID of the owner, name of the owner, whether the phase has failed or is paused, the step mode, phase ID, and arbitration mask. You cannot access any of these properties from a property page.

NOTE: *You must call the CreatePhasefromVBISPhase2 method to set up the phase plate with all of these values when you first start. You then use these properties to update the phase plate as the these values change.*

The following figure shows the run-time phase properties available for the BatchPhasePlate.



BatchPhasePlate Run-time Phase Properties

The following table lists the run-time phase items that are available for the Phase Plate, but are not listed on a property page.

BatchPhasePlate Run-time Phase Properties		
Item	**Description**	**Associated Property**
Name field	Name of the phase displayed.	PhaseName
Class field	Class of the phase displayed.	PhaseClass
State field	Current state of the active phase: • 0 = Idle • 1 = Stopping • 2 = Stopped • 3 = Starting • 4 = Running • 5 = Holding • 6 = Held • 7 = Aborting • 8 = Aborted • 9 = Complete	State
Step Index field	Current step of the active phase.	StepIndex
Unit field	The unit that the phase is running.	ParentUnit

BatchPhasePlate Run-time Phase Properties		
Item	**Description**	**Associated Property**
Owner Batch ID field	ID of the batch that the phase is running.	OwnerBatchID
External, Batch Execution, or Operator indicator	Owner of the phase. This field can be set to: • 0 = No Owner • 1 = External • 2 = Batch Execution • 3 = Operator If set to 1, 2, or 3, the indicator displays as green.	Owner
Failure indicator	Indicates a failure within the phase. This field can be set to: • True, which means that the indicator is blinking red • False, which means that the indicator is off	Failure
Paused indicator	Indicates a pause within the phase. This field can be set to: • 0 = Off, indicator is shaded • 1 = Pausing, indicator blinks yellow • 2 = Paused, indicator displays as yellow	Pause
Step Mode check box	Indicates if the phase is in O-Auto or P-Auto mode. True indicates semi-automatic (O-Auto) mode. When a step is in O-Auto mode, its transition executes and an operator can send commands to its procedure. False indicates automatic (P-Auto) mode. When a step is in P-Auto mode, its transition executes but an operator cannot send commands to its procedure.	StepMode
Not displayed	The ID of the phase.	PhaseID
Not displayed	The mask that is used by the phase plate to determine if the Acquire or Release button should be enabled. If the owner of the phase is the operator (Owner=3) and the arbitration mask is not zero, then it enables the Step Mode check box.	ArbitrationMask

BatchPhasePlate Run-time Phase Properties		
Item	**Description**	**Associated Property**
Phase icon	The icon of the currently selected phase.	PhaseIcon

Configuring the Run-time Phase

To configure the Phase Plate settings for the BatchPhasePlate control, insert the BatchPhasePlate control in any OLE container, such as Visual Basic. Write a Visual Basic scripts to set these values. For example, to set the BatchPhasePlate state value, type the following:

```
BatchPhasePlate1.State = 0 `IDLE
```

BatchPhasePlate Run-time Phase Properties

The following table lists the properties that control the run-time phase settings for the BatchPhasePlate control.

BatchPhasePlate Run-time Phase Properties	
Property	**Syntax**
PhaseName Name of the phase displayed.	**C++ Syntax:** **CString** CBatchPhasePlate::GetPhaseName(); **void** CBatchPhasePlate::SetPhaseName(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseName[= *text$*]
PhaseClass Class of the phase displayed.	**C++ Syntax:** **CString** CBatchPhasePlate::GetPhaseClass(); **void** CBatchPhasePlate::SetPhaseClass(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*PhaseClass[= *text$*]

BatchPhasePlate Run-time Phase Properties	
Property	**Syntax**
State Current state of the active phase: • 0 = Idle • 1 = Stopping • 2 = Stopped • 3 = Starting • 4 = Running • 5 = Holding • 6 = Held • 7 = Aborting • 8 = Aborted • 9 = Complete	**C++ Syntax:** **short** CBatchPhasePlate::GetState(); **void** CBatchPhasePlate::SetState(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.State[= *value%*]
StepIndex Current step of the active phase.	**C++ Syntax:** **short** CBatchPhasePlate::GetStepIndex(); **void** CBatchPhasePlate::SetStepIndex(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StepIndex[= *value%*]
ParentUnit The unit that the phase is running.	**C++ Syntax:** **CString** CBatchPhasePlate::GetParentUnit(); **void** CBatchPhasePlate::SetParentUnit(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.ParentUnit[= *text$*]
OwnerBatchID ID of the batch that the phase is running.	**C++ Syntax:** **CString** CBatchPhasePlate::GetOwnerBatchID(); **void** CBatchPhasePlate::SetOwnerBatchID(**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control*.OwnerBatchID[= *text$*]

BatchPhasePlate Run-time Phase Properties	
Property	**Syntax**
Owner Owner of the phase. This field can be set to: • 0 = No Owner • 1 = External • 2 = Batch Execution • 3 = Operator If set to 1, 2, or 3, the indicator displays as green.	**C++ Syntax:** **short** CBatchPhasePlate::GetOwner(); **void** CBatchPhasePlate::SetOwner(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.Owner[= *value%*]
Failure Indicates a failure within the phase. This field can be set to: • True, which means that the indicator is blinking red • False, which means that the indicator is off	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetFailure(); **void** CBatchPhasePlate::SetFailure(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.Failure[= *boolvalue*]
Pause Indicates a pause within the phase. This field can be set to: • 0 = Off, indicator is shaded • 1 = Pausing, indicator blinks yellow • 2 = Paused, indicator displays as yellow	**C++ Syntax:** **short** CBatchPhasePlate::GetPause(); **void** CBatchPhasePlate::SetPause(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control*.Pause[= *value%*]
StepMode Indicates if the phase is in O-Auto or P-Auto mode. True indicates semi-automatic (O-Auto) mode. When a step is in O-Auto mode, its transition executes and an operator can send commands to its procedure. False indicates automatic (P-Auto) mode. When a step is in P-Auto mode, its transition executes but an operator cannot send commands to its procedure.	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetStepMode(); **void** CBatchPhasePlate::SetStepMode(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StepMode[= *boolvalue*]

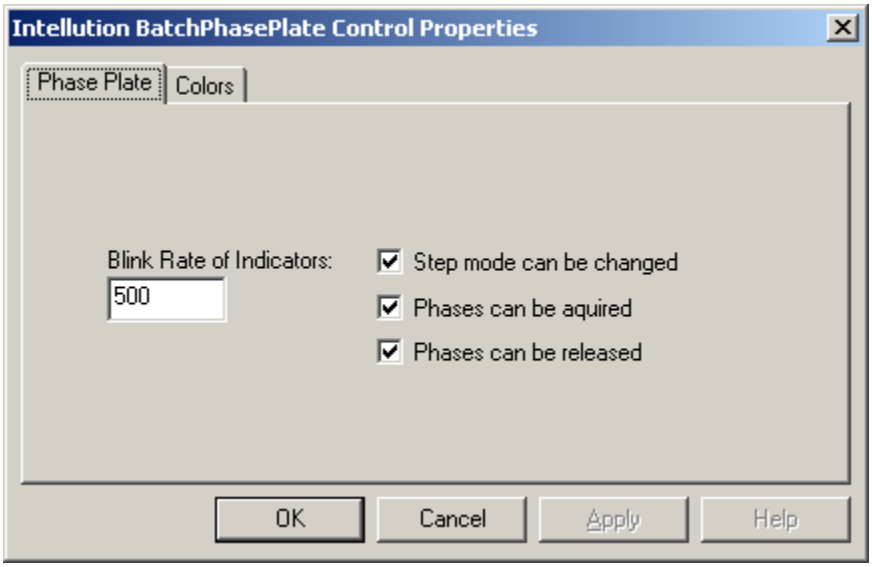
BatchPhasePlate Run-time Phase Properties	
Property	**Syntax**
PhaseID The ID of the phase.	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetAllowStepModeChange(); **void** CBatchPhasePlate::SetAllowStepModeChange(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.AllowStepModeChange[= *boolvalue*]
ArbitrationMask The mask that is used by the phase plate to determine if the Acquire or Release button should be enabled. If the owner of the phase is the operator (Owner=3) and the arbitration mask is not zero, then it enables the Step Mode check box.	**C++ Syntax:** **long** CBatchPhasePlate::GetArbitrationMask(); **void** CBatchPhasePlate::SetArbitrationMask(**long** value); **Visual Basic Syntax:** [*form.*]*Control*.ArbitrationMask[= *value!*]
PhaseIcon The icon of the currently selected phase.	**C++ Syntax:** IPictureDisp* PhaseIcon(BSTR strIconFilename); **void** PhaseIcon(**BSTR** strIconFilename, IPictureDisp* newValue); **Visual Basic Syntax:** [*form.*]*Control*.PhaseIcon(strIconFilename) = *icon* *icon* = BatchPhasePlate1.PhaseIcon(strIconFilename) where *icon* is of type StdPicture.

Phase Plate Property Page

The Phase Plate property page:

- Lets you configure the blinking rate for the indicator lights; and whether or not the operator can change the step mode, acquire a phase, or release a phase.
- Is available at design time.

The following figure shows the Phase Plate property page.



Phase Plate Property Page

The following table lists the items that are available on the Phase Plate property page.

BatchPhasePlate Control Phase Plate Property Page		
Item	**Description**	**Associated Property**
Blink Rate of Indicators field	The blink rate, in milliseconds, of the Paused and Failure indicator lights. Valid refresh rates are greater than 0 milliseconds (ms). The default is 500 ms.	IndicatorBlinkRate
Step mode can be Changed check box	When selected, allows the operator to change the step mode.	AllowStepModeChange
Phases can be Acquired check box	When selected, allows the operator to acquire a phase.	AllowPhaseAcquire
Phases can be Released check box	When selected, allows the operator to release a phase.	AllowPhaseRelease

Configuring Phase Plate Settings

The steps that follow explain how to configure the Phase Plate settings for the BatchPhasePlate ActiveX control.

To configure the Phase Plate settings for the BatchPhasePlate control:

1. Open the BatchPhasePlate control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Phase Plate property page.
4. Enter the blink rate interval, in milliseconds, for the Paused and Failure indicators.
5. Select Step Mode can be Changed to allow the operator to change the step mode; deselect it to disable the Step Mode check box in the control.
6. Select Phases can be Acquired to allow the operator to acquire a phase; deselect it to disable the Acquire button on the control. The Release button is not affected by this check box setting.
7. Select Phases can be Released to allow the operator to acquire a phase; deselect it to disable the Release button on the control. The Acquire button is not affected by this check box setting.
8. Click OK to save your changes.

Phase Plate Properties

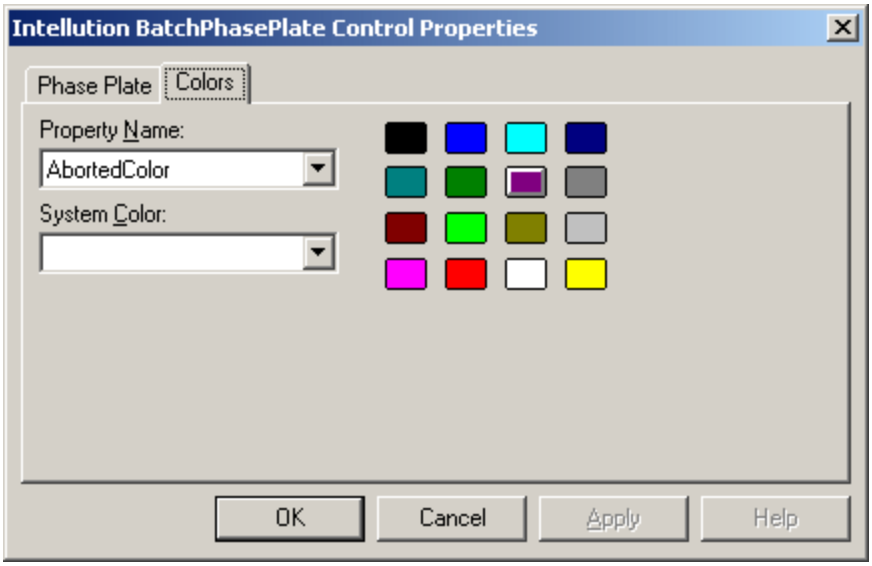
The following table lists the properties that control the Phase Plate settings for the BatchPhasePlate control.

BatchPhasePlate Control Phase Plate Properties	
Property	**Syntax**
AllowPhaseAcquire Sets whether or not the operator can acquire the phase.	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetAllowPhaseAcquire(); **void** CBatchPhasePlate::SetAllowPhaseAcquire(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowPhaseAcquire[= *boolvalue*]
AllowPhaseRelease Sets whether or not the operator can release the phase.	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetAllowPhaseRelease(); **void** CBatchPhasePlate::SetAllowPhaseRelease(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowPhaseRelease[= *boolvalue*]

BatchPhasePlate Control Phase Plate Properties	
Property	**Syntax**
AllowStepModeChange Sets whether or not the operator can change the step mode.	**C++ Syntax:** **BOOL** CBatchPhasePlate::GetAllowStepModeChange(); **void** CBatchPhasePlate::SetAllowStepModeChange(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowStepModeChange[= *boolvalue*]
IndicatorBlinkRate Sets the blink rate, in milliseconds, of the Paused and Failure indicator lights.	**C++ Syntax:** **short** CBatchPhasePlate::GetIndicatorBlinkRate(); **void** CBatchPhasePlate::SetIndicatorBlinkRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*IndicatorBlinkRate[= *value%*]

BatchPhasePlate Control Colors Property Page

The Colors property page lets you set the color for the BatchPhasePlate control states. You can choose a color from the displayed color chart or you can use the System Color drop-down list box to display and select from a pre-defined system color chart. The Colors property page, shown in the following figure, is available at design time and at run time.



Colors Property Page

The following table lists the items that are available on the Colors property page.

BatchPhasePlate Control Colors Property Page		
Item	**Description**	**Corresponding Properties**
Property Name drop-down list box	Lists all the available properties for which you can set the color.	IdleColor StoppingColor StoppedColor StartingColor RunningColor HoldingColor HeldColor AbortingColor AbortedColor CompleteColor RestartingColor
System Color drop-down list box	Lists the available system colors that you can assign to a property.	None

Setting Colors

The steps that follow explain how to configure the colors for the BatchPhasePlate ActiveX control.

To set the colors for the BatchPhasePlate control:

1. Open the control in any OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Colors property page.
4. Select a property from the Property Name drop-down list box.
5. Click a color in the color palette or select a pre-defined system color from the System Color drop-down list box.

Colors Properties

The following table lists the properties that control the colors in the BatchPhasePlate control.

BatchPhasePlate Control Colors Properties	
Property	**Syntax**
AbortedColor Sets the color of the Aborted phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetAbortedColor(); **void** CBatchPhasePlate::SetAbortedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.AbortedColor[= *color%*]
AbortingColor Sets the color of the Aborting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetAbortingColor(); **void** CBatchPhasePlate::SetAbortingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.AbortingColor[= *color%*]
CompleteColor Sets the color of the Complete phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetCompleteColor(); **void** CBatchPhasePlate::SetCompleteColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.CompleteColor[= *color%*]
HeldColor Sets the color of the Held phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetHeldColor(); **void** CBatchPhasePlate::SetHeldColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeldColor[= *color%*]
HoldingColor Sets the color of the Holding phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetHoldingColor(); **void** CBatchPhasePlate::SetHoldingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HoldingColor[= *color%*]

BatchPhasePlate Control Colors Properties	
Property	**Syntax**
IdleColor Sets the color of the Idle phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetIdleColor(); **void** CBatchPhasePlate::SetIdleColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.IdleColor[= *color%*]
RestartingColor Sets the color of the Restarting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetRestartingColor(); **void** CBatchPhasePlate::SetRestartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RestartingColor[= *color%*]
RunningColor Sets the color of the Running phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetRunningColor(); **void** CBatchPhasePlate::SetRunningColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RunningColor[= *color%*]
StartingColor Sets the color of the Starting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetStartingColor(); **void** CBatchPhasePlate::SetStartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StartingColor[= *color%*]
StoppedColor Sets the color of the Stopped phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetStoppedColor(); **void** CBatchPhasePlate::SetStoppedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StoppedColor[= *color%*]

BatchPhasePlate Control Colors Properties	
Property	**Syntax**
StoppingColor Sets the color of the Stopping phase state.	**C++ Syntax:** **OLE_COLOR** CBatchPhasePlate::GetStoppingColor(); **void** CBatchPhasePlate::SetStoppingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StoppingColor[= *color%*]

Color Property Examples

The following show examples for setting color properties.

C++ Example

```
OLE_COLOR StartingColor = pBatchPhasePlate->GetStartingColor();
OLE_COLOR newStartingColor = 0x0; // black
pBatchPhasePlate->SetStartingColor(newStartingColor);
```

Visual Basic Example

```
Static MP_OriginalStartingColor As Long
MP_OriginalStartingColor = BatchPhasePlate1.StartingColor
     ' get the current color
BatchPhasePlate1.StartingColor = &H808080
     ' set to dark gray
```

BatchPhasePlate Control Methods

The BatchPhasePlate control supports the following methods:

- ClearPhase Method
- CreatePhaseFromVBISPhase2 Method
- BlinkOperatorOwnerIndicator Method
- AboutBox Method

ClearPhase Method

Description

Clears the phase from the phase plate.

C++ Syntax

```
void CBatchPhasePlate::ClearPhase();
```

Visual Basic Syntax

```
[form.]BatchPhasePlateControl1.ClearPhase()
```

Parameters

None.

Return Type

Void.

C++ Example

```
BatchPhasePlate1.ClearPhase();
```

Visual Basic Example

```
BatchPhasePlate1.ClearPhase
```

CreatePhaseFromVBISPhase2 Method

Description

This method sets the text fields, indicators, buttons, check box, phase icon, and run-time phase properties of the BatchPhasePlate control with the information contained in the lpVBISPhase2 and pAreaModel objects. Use this method to set everything initially, and then use the run-time phase properties to update the phase plate with the latest individual values of the phase. It is up to you to update the phase plate with the latest phase values. The BatchManualPhase control does this for you automatically.

C++ Syntax

```
void CBatchPhasePlate::CreatePhaseFromVBISPhase2(VBISPhase2* lpVBISPhase2,
VBISAreaModel3* pAreaModel);
```

Visual Basic Syntax

```
[form.]BatchPhasePlateControl1.CreatePhaseFromVBISPhase2(lpVBISPhase2 As Object,
pAreaModel As Object) As Boolean
```

Parameters

Parameter	Description
lpVBISPhase2	The pointer to the VBISPhase2 object.
pAreaModel	The pointer to the VBISAreaModel3 object.

Return Type

Boolean.

- 1 if successful.
- 0 if not successful.

C++ Example

```
VBISPhase2 lpVBISPhase2;
VBISAreaModel3 pAreaModel;

// get the pAreaModel and lpVBISPhase2 from VBIS
// ...

BOOL result = m_BatchPhasePlate1.CreatePhaseFromVBISPhase2(lpVBISPhase2, pAreaModel);
```

Visual Basic Example

```
Dim result As Boolean
Dim lpVBISPhase2 As VBISPhase2
Dim pAreaModel As VBISAreaModel3
' get the pAreaModel and lpVBISPhase2 from VBIS
' ...

Set result = BatchPhasePlate1.CreatePhaseFromVBISPhase2(lpVBISPhase2, pAreaModel)
```

BlinkOperatorOwnerIndicator Method

Description

This method flashes the operator owner indicator. The indicator stops blinking once ownership or arbitration parameters change. This method should be called after the container application gets the FireOwnerButtonClick event and successfully sends the command to VBIS.

C++ Syntax

```
void CBatchPhasePlate::BlinkOperatorOwnerIndicator();
```

Visual Basic Syntax

```
[form.]BatchPhasePlateControl1.BlinkOperatorOwnerIndicator()
```

Parameters

None.

Return Type

Void.

C++ Example

```
BatchPhasePlate1.BlinkOperatorOwnerIndicator();
```

Visual Basic Example

```
BatchPhasePlate1.BlinkOperatorOwnerIndicator
```

AboutBox Method

Description

Displays the About box.

C++ Syntax

```
void CBatchPhasePlate::AboutBox()
```

Visual Basic Syntax

```
[form.]BatchPhasePlate1.AboutBox()
```

Parameters

None.

Return Type

Void.

C++ Example

```
BatchPhasePlate1.AboutBox();
```

Visual Basic Example

```
BatchPhasePlate1.AboutBox
```

BatchPhasePlate Control Events

The BatchPhasePlate control generates the following events:

- OwnerButtonClick Event
- StepModeChanged Event

OwnerButtonClick Event

Description

Occurs when the user clicks on the Owner button.

Event ID

1

C++ Syntax

```
void OwnerButtonClick(BOOL bAcquiring);
```

Visual Basic Syntax

```
Event OwnerButtonClick(bAcquiring as boolean)
```

Parameters

Parameter	Description
bAcquiring	Set to True if the user clicked the button to acquire a phase. Set to False if the user clicked the button to release a phase.

StepModeChanged Event

Description

Occurs when the operator issues a StepModeChanged command.

Event ID

2

C++ Syntax

```
void StepModeChanged(long phaseID, LPCTSTR strPhaseName, LPCTSTR strParentUnit, LPCTSTR
strOwnerBatchID, BOOL bStepMode);
```

Visual Basic Syntax

```
Event StepModeChanged(phaseID as Long, strPhaseName as String, strParentUnit as String,
strOwnerBatchID as String, bStepMode as Boolean)
```

Parameters

Parameter	Description
phaseID	The numeric ID of the phase being aborted.
strPhaseName	The name of the phase.
strParentUnit	The name of the parent unit of the phase.
strOwnerBatchID	The ID of the batch that owns the phase.
bStepMode	Set to True if the user checked the step mode, or False if the user cleared the Step Mode check box.

C++ Event Sink Map for BatchPhasePlate

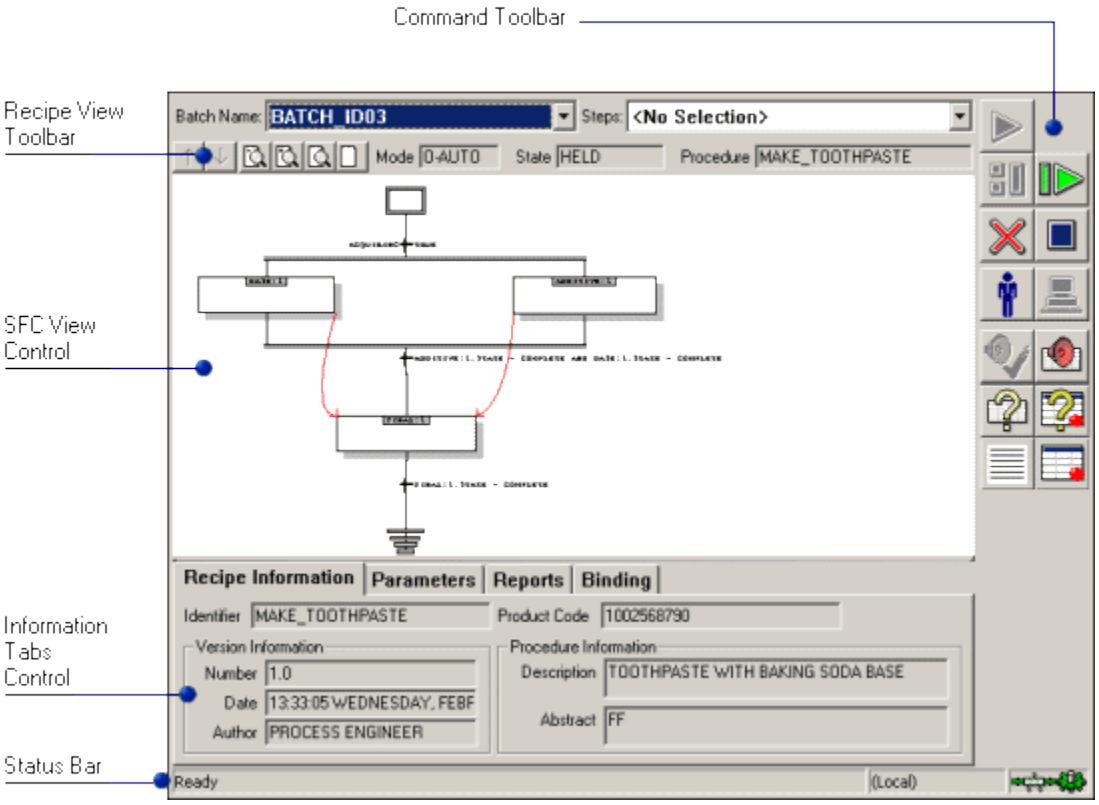
The following shows an example of an event sink map.

```
BEGIN_EVENTSINK_MAP(CBatchPhatePlateDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CBatchPhatePlateDlg)
     ON_EVENT(CBatchPhatePlateDlg, IDC_BATCHPHASEPLATECTRL1, 1 /* OwnerButtonClick */,
OnOwnerButtonClickBatchphaseplatectrl1, VTS_BOOL)
     ON_EVENT(CBatchPhatePlateDlg, IDC_BATCHPHASEPLATECTRL1, 2 /* StepModeChanged */,
OnStepModeChangedBatchphaseplatectrl1, VTS_I4 VTS_BSTR VTS_BSTR VTS_BSTR VTS_BOOL)
     //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()
```

BatchSFC ActiveX Control

The "Intellution BatchSFC Control," shown in the following figure, allows you to monitor and control a batch from a sequential function chart representation of the batch. The BatchSFC control contains two controls: BatchSFCView and BatchSFCInfoTab. The BatchSFCView and BatchSFCInfoTab controls, within the BatchSFC control, function in the same manner as the Batch Execution Client's SFC View. For information on the SFC View, refer to the Operator's Manual. The BatchSFCView and BatchSFCInfoTab controls should only be used within the BatchSFC control.

NOTE: *The BatchSFC Control does not support Active Step Change.*



BatchSFC Control

Using the BatchSFC ActiveX Control

The following table describes the fields in the BatchSFC ActiveX Control.

BatchSFC Window Descriptions	
Field	**Description**
Batch Name	Lists all batches listed in the batch list.
Steps	Lists all steps from the currently selected batch in the Batch Name list.
Recipe View Toolbar	Refer to the Recipe View Toolbar section.
Command Toolbar	Refer to the Command Toolbar section.
Status Bar	Refer to the Status Bar section.

For more information on the BatchSFC screens refer to the Operations Manual.

Recipe View Toolbar

The following list describes the fields in the BatchSFC ActiveX Control:

Recipe View Toolbar	
Item	**Description**
Drill Up button	Use this button to drill up in the SFC View window.
Drill Down button	Use this button to drill down in the SFC View window.
Zoom Out button	Use this button to get a smaller view of the sequential function chart; you can zoom down to 50 percent of the original size.
Zoom In button	Use this button to get a closer view of the sequential function chart; you can zoom in up to 300 percent of the original size.
Zoom Full button	Use this button to make the sequential function chart fit the window.
Mode	Describes whether the step is in automatic or manual mode.
State	Describes the state of the currently selected step. Possible states are Aborted, Aborting, Complete, Held, Holding, Idle, Ready, Restarting, Running, Stopping, and Stopped.
Unit Procedure	The operation that controls the function of a single piece of equipment.

Command Toolbar

You can use the command buttons to control the selected phase. The command toolbar is displayed on the right side of the control. The following table describes the function of each button.

NOTE: *From the General property page the developer can configure if the command toolbar will appear on the BatchSFC control and what buttons will appear on it.*

Command Button Description	
This button...	**Does this...**
	Starts a batch.
	Holds a batch.
	Restarts a batch.
	Aborts the running batch entirely.
	Stops the running batch.
	Switches to manual mode.
	Returns to automatic mode.
	Clears all failed phases.
	Displays the alarms summary.
	Displays all unacknowledged operator prompts.
	Displays all unacknowledged transition breakpoint prompts.
	Displays the Add Operator Comment to the Batch Record dialog box, so that you can add a comment to the selected batch, or all batches.

Command Button Description	
This button...	**Does this...**
	Displays the Transition Breakpoints dialog box, so that you can clear all transition breakpoints.

Recipe Information Tab Properties

The following table lists and describes the properties associated with the Recipe Information tab.

Recipe Information Tab Properties	
Field	**Description**
Identifier field	Displays the recipe name defined in the recipe header. This field is display-only.
Number field	Displays the recipe's version number as defined in the recipe header. This field is display-only.
Date field	Displays the date the recipe was created as defined in the recipe header. This field is display-only.
Author field	Displays the name of the recipe's author as defined in the recipe header. This field is display-only.
Product Code	Displays the recipe's product code as defined in the recipe header. This field is display only.
Description field	Displays a description of the recipe as defined in the recipe header. This field is display-only.
Abstract field	Displays an abstract of the recipe as defined in the recipe header. This field is display only.

Parameters Tab Properties

The Parameters tab shows parameter information (name, low, value, high, and engineering unit) for the currently highlighted step of the running batch.

Reports Tab Properties

The Reports tab displays the recipe's name, value, and engineering units.

Binding Tab Properties

The following table lists and describes the properties associated with the Binding tab.

Binding Tab Properties	
Item	**Description**
Batch ID field	Displays the batch ID of the batch prompting the operator for binding information.
Recipe Step field	Displays the name of the recipe step prompting the operator for binding information.
Bind Type	Displays the bind type defined for the unit procedure. Valid entries include: Operator or Automatic.
Unit Class	Displays the unit class from which you can choose a unit to bind to the unit procedure.
Current Binding	Displays the unit currently selected to bind to the unit procedure or the method of binding (Automatic or Operator) selected to perform the binding.
Unit Used	Displays the unit that was bound to the unit procedure.
Binding button	Click to bind the selected unit or to specify the method of binding to the unit procedure.
Binding field	Select the bind type or the unit you want the unit procedure to use.

Status Bar

The status bar at the bottom of the BatchSFC control (shown in the BatchSFC ActiveX Control section) provides the following:

- Status information.
- Name of the current server, or Local for the local computer. Double-clicking this field displays the Server property page.
- Selectable icon for connecting or disconnecting to VBIS and the server. The icon also reports the status of the connection.

NOTE: *From the General property page the developer can configure if the status bar will appear on the BatchSFC control.*

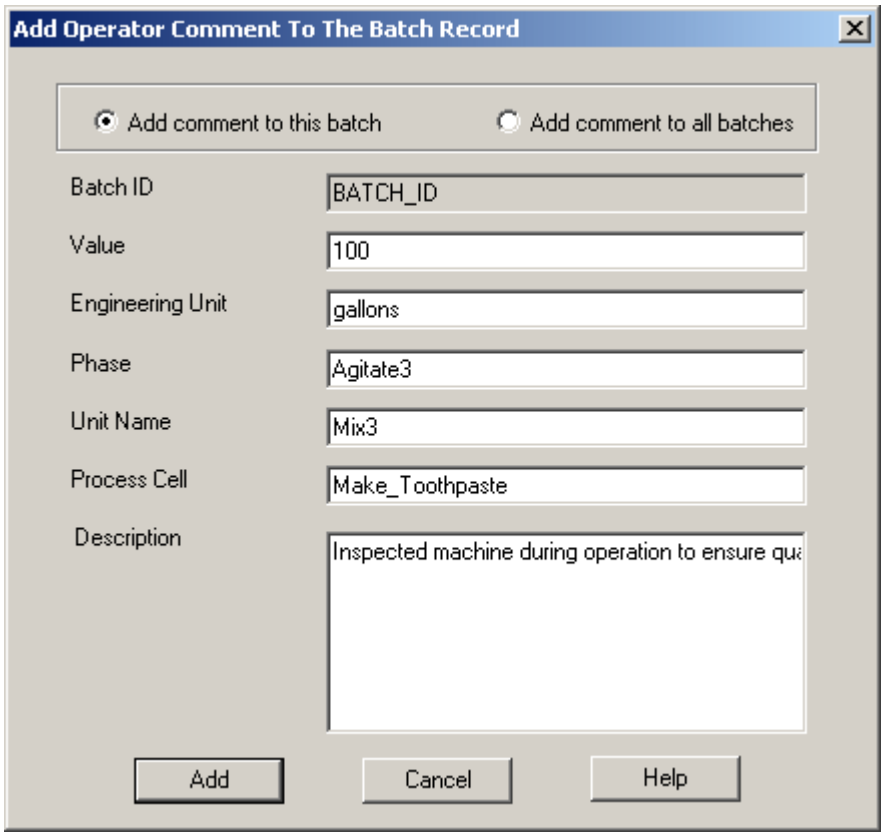
Context Menu

Right-clicking anywhere in the BatchSFC control displays a context menu that contains the commands (the same as in the command toolbar) and a Properties menu item (to display the property pages for the BatchSFC control). If you right-click a step, a Binding Menu item is also displayed.

Add Operator Comment to the Batch Record Dialog Box

To display the Add Operator Comment to the Batch Record dialog box in the BatchSFC or BatchList control, click the Add Comment button on the command toolbar. The note that you add can be applied to the selected batch or all batches.

In the dialog box that appears, you can add information about the current value, its engineering units, the phase name, the unit name, the process cell, or enter an extended note in the description field. You can choose to enter information in all of these fields, just a few of them, or only one of them, such as the Description field. The following figure shows an example of this dialog in use:



The information is recorded as part of the electronic batch record in the OPERATOR_COMMENTS Table, as a USER event.

BatchSFC Control Properties

Using the BatchSFC control's property pages, developers can configure the control's appearance and functionality for operators. For example, the developer can configure the colors of the step states, the command buttons that are displayed, and the security for the control.

The BatchSFC control provides property pages that you can use to view and change the control's properties. The following sections describe each property for the BatchSFC control. The properties are grouped into the following categories:

- Server properties
- General properties
- Information Tabs properties
- Recipe View properties
- Colors properties
- Security properties
- Electronic Signature properties

Configuring BatchSFC Control Properties

The steps that follow explain how to configure the property settings for the BatchSFC ActiveX control.

To configure the property settings for the BatchSFC control:

1. Open the BatchSFC control in any OLE container:
    - Proficy iFIX WorkSpace
    - Visual Basic
    - Visual C++
    - Web browsers, such as Internet Explorer

    NOTE: *The Security property page is available only at design time. The first three containers in the preceding list are design-time containers.*

2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.

    NOTE: *To display the Server property page, you can also click the VBIS Server name in the status bar.*

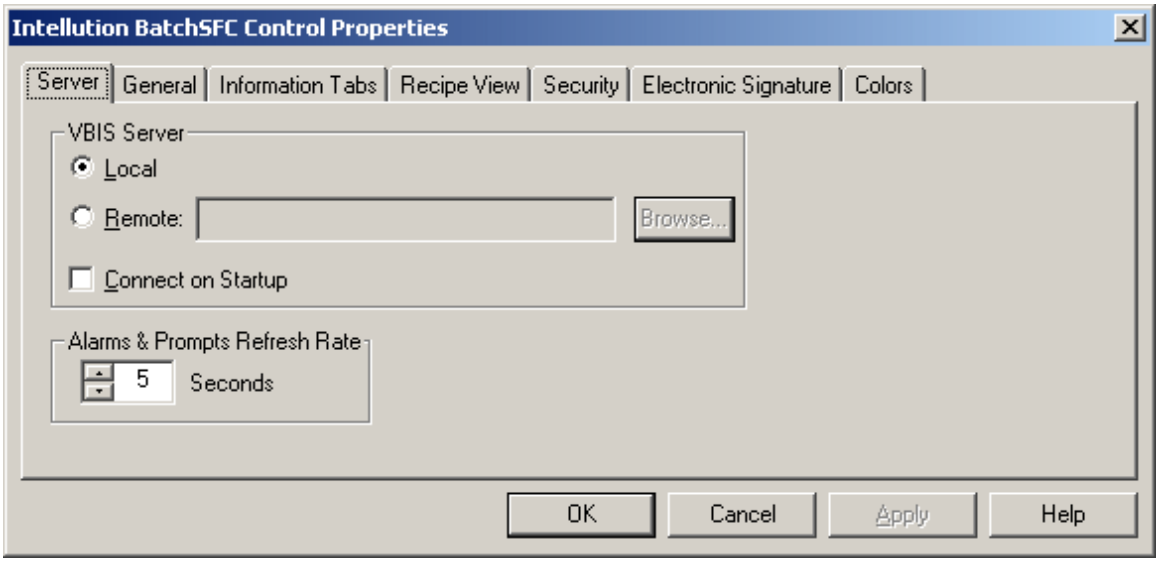
3. Select the tab for the property whose settings you want to configure.
4. Configure the settings, as described in the following sections.
5. Click OK to save your changes.

Server Property Page

The Server property page:

- Lets you configure the VBIS Server settings, such as whether the VBIS Server is local or remote, for the BatchSFC control.
- Is available at design time and may be enabled at run time using the Security property page.

The following figure shows the Server property page.



Server Property Page

The following table lists the items that are available on the Server property page.

BatchSFC Control Server Property Page		
Item	**Description**	**Associated Property**
Local button	When selected, indicates that the VBIS Server is running on the same computer as the BatchSFC control.	VBISServerName
Remote button	When selected, indicates that the VBIS Server is running on a different computer from the BatchSFC control.	VBISServerName
Remote field	When Remote is selected, lets you specify the computer on which the VBIS Server is running.	VBISServerName

BatchSFC Control Server Property Page		
Item	**Description**	**Associated Property**
Browse button	When Remote is selected, lets you browse through the network to select the computer on which the VBIS Server is running.	None
Connect on Startup check box	When selected, the BatchSFC control automatically connects to the VBIS Server when the control is opened.	ConnectAtStartup
Alarms & Prompts Refresh Rate field	Lets you specify the interval, in seconds, that data in the Alarms and Operator Prompts dialog boxes is updated in the control. Valid refresh rates range from 1 to 120 seconds. The default Refresh Rate is 5 seconds. ***NOTE:*** *On computers with 166 Megahertz or less, you may experience problems if you set the refresh rate to less than 3 seconds.*	RefreshRate

Server Properties

The following table lists the C++ and Visual Basic syntax for the properties that control the Server settings for the BatchSFC control.

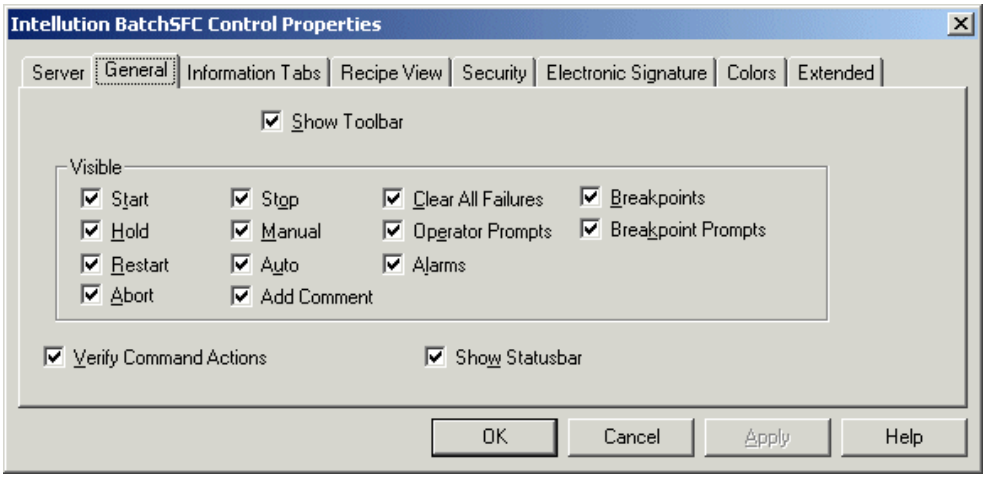
BatchSFC Control Server Properties	
Property	**Syntax**
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchSFC::GetConnectAtStartup(); **void** CBatchSFC::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ConnectAtStartup[= *boolvalue*]

BatchSFC Control Server Properties	
Property	**Syntax**
RefreshRate Sets the rate, in seconds, that the data in the Alarms and Prompts dialog is updated. The range of values you can enter is 1 to 120. A value of zero for the Refresh Rate is not supported for the BatchSFC control. This means that manual refresh also is not supported in the BatchSFC control. The default Refresh Rate is 5 seconds.	**C++ Syntax:** **short** CBatchSFC::GetRefreshRate(); **void** CBatchSFC::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **CString** CBatchSFC::GetVBISServerName(); **void** CBatchSFC::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]

General Property Page

The General property page lets the developer configure which command buttons are visible on the command toolbar and whether the operator is prompted to verify command execution. The developer can also hide the entire command toolbar or status bar.

The following figure shows the General property page.



General Property Page

The following table lists the items that are available on the General property page.

BatchSFC Control General Property Page		
Item	**Description**	**Properties**
Show Toolbar	Selecting the check box displays the command toolbar in the BatchSFC control. Deselecting the check box removes the toolbar from the control.	ToolbarVisible
Start Hold Restart Abort Stop Manual Auto Add Comment Clear All Failures Operator Prompts Alarms Breakpoints Breakpoint Prompt	Selecting a check box displays the associated button on the command toolbar in the BatchSFC control. For example, selecting the Start check box lets operators start batches using the Start button. Deselecting the check box removes the button from the toolbar.	StartButtonVisible HoldButtonVisible RestartButtonVisible AbortButtonVisible StopButtonVisible ManualButtonVisible AutoButtonVisible AddCommentButtonVisible ClearAllFailuresButtonVisible OperatorPromptsButtonVisible AlarmsButtonVisible BreakpointsButtonVisible BreakpointsPromptButtonVisible
Verify Command Actions check box	When selected, the operator is prompted for confirmation when executing a command.	VerifyCommandActions
Show Statusbar check box	Sets whether or not to display the status bar.	StatusBarVisible

General Properties

The following table lists the C++ and Visual Basic syntax for the properties that control the display of command buttons and the status bar in the BatchSFC control.

BatchSFC Control General Properties	
Property	**Syntax**
AbortButton Sets whether or not to display the Abort button.	**C++ Syntax:** **BOOL** CBatchSFC::GetAbortButton (); **void** CBatchSFC::SetAbortButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AbortButton[= *boolvalue*]
AlarmsButton Sets whether or not to display the Alarms button.	**C++ Syntax:** **BOOL** CBatchSFC::GetAlarmsButton(); **void** CBatchSFC::SetAlarmsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AlarmsButton[= *boolvalue*]
AutoButton Sets whether or not to display the Auto button.	**C++ Syntax:** **BOOL** CBatchSFC::GetAutoButton(); **void** CBatchSFC::SetAutoButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AutoButton[= *boolvalue*]
AddCommentButton Sets whether or not to display the Add Comment button.	**C++ Syntax:** **BOOL** CBatchSFC::GetAddCommentButton(); **void** CBatchSFC::SetAddCommentButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddCommentButton[= *boolvalue*]
BreakpointsButton Sets whether or not to display the Breakpoints button.	**C++ Syntax:** **BOOL** CBatchSFC::GetBreakpointButton(); **void** CBatchSFC::SetBreakpointButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BreakpointButton[= *boolvalue*]

BatchSFC Control General Properties	
Property	**Syntax**
BreakpointPromptsButton Sets whether or not to display the Breakpoint Prompt button.	**C++ Syntax:** **BOOL** CBatchSFC::GetBreakpointPromptsButton(); **void** CBatchSFC::SetBreakpointPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BreakpointPromptsButton[= *boolvalue*]
ClearAllFailuresButton Sets whether or not to display the Clear All Failures button.	**C++ Syntax:** **BOOL** CBatchSFC::GetClearAllFailuresButton(); **void** CBatchSFC::SetClearAllFailuresButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ClearAllFailuresButton[= *boolvalue*]
HoldButton Sets whether or not to display the Hold button.	**C++ Syntax:** **BOOL** CBatchSFC::GetHoldButton(); **void** CBatchSFC::SetHoldButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.HoldButton[= *boolvalue*]
ManualButton Sets whether or not to display the Manual button.	**C++ Syntax:** **BOOL** CBatchSFC::GetManualButton(); **void** CBatchSFC::SetManualButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ManualButton[= *boolvalue*]
OperatorPromptsButton Sets whether or not to display the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchSFC::GetOperatorPromptsButton(); **void** CBatchSFC::SetOperatorPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.OperatorPromptsButton[= *boolvalue*]

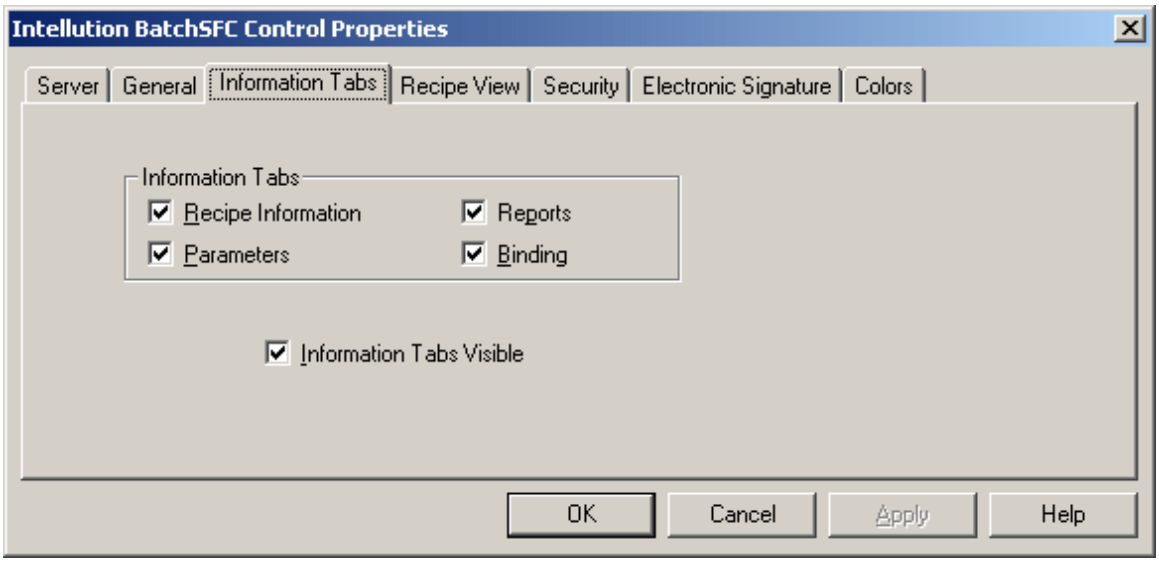
BatchSFC Control General Properties	
Property	**Syntax**
RestartButton Sets whether or not to display the Restart button.	**C++ Syntax:** **BOOL** CBatchSFC::GetRestartButton(); **void** CBatchSFC::SetRestartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.RestartButton[= *boolvalue*]
StartButton Sets whether or not to display the Start Batch button.	**C++ Syntax:** **BOOL** CBatchSFC::GetStartButton(); **void** CBatchSFC::SetStartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StartButton[= *boolvalue*]
StatusBarVisible Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchSFC::GetStatusBarVisible(); **void** CBatchSFC::SetStatusBarVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.StatusBarVisible[= *boolvalue*]
StopButton Sets whether or not to display the Stop Batch button.	**C++ Syntax:** **BOOL** CBatchSFC::GetStopButton(); **void** CBatchSFC::SetStopButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StopButton[= *boolvalue*]
ToolbarVisible Sets whether or not to display the command toolbar.	**C++ Syntax:** **BOOL** CBatchSFC::GetToolbarVisible(); **void** CBatchSFC::SetToolbarVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.ToolbarVisible[= *boolvalue*]

BatchSFC Control General Properties	
Property	**Syntax**
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when executing a command.	**C++ Syntax:** **BOOL** CBatchSFC::GetVerifyCommandActions(); **void** CBatchSFC::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VerifyCommandActions[= *boolvalue*]

Information Tabs Property Page

The Information Tabs property page lets the developer configure which tabs will appear in the SFC Information Tab control. The developer can also hide the SFC Information Tab control completely.

The following figure shows the Information Tabs property page.



Information Tabs Property Page

The following table lists the items that are available on the Information Tabs property page.

SFC Control Information Tabs Property Page		
Item	**Description**	**Associated Property**
Recipe Information check box Parameters check box Reports check box Binding check box	Selecting a check box displays the associated tab on the SFC Information Tabs View in the BatchSFC control. Deselecting the check box removes the tab from the Information Tabs control.	RecipeInfoTabVisible ParametersTabVisible ReportsTabVisible BindingTabVisible
Information Tabs Visible check box	Selecting this check box displays the Information Tabs control. Deselecting the check box removes the Information Tabs control from the BatchSFC control.	InformationTabsVisible

Information Tabs Properties

The following table provides the C++ and Visual Basic syntax for the properties that control the Information Tabs settings for the BatchSFC control.

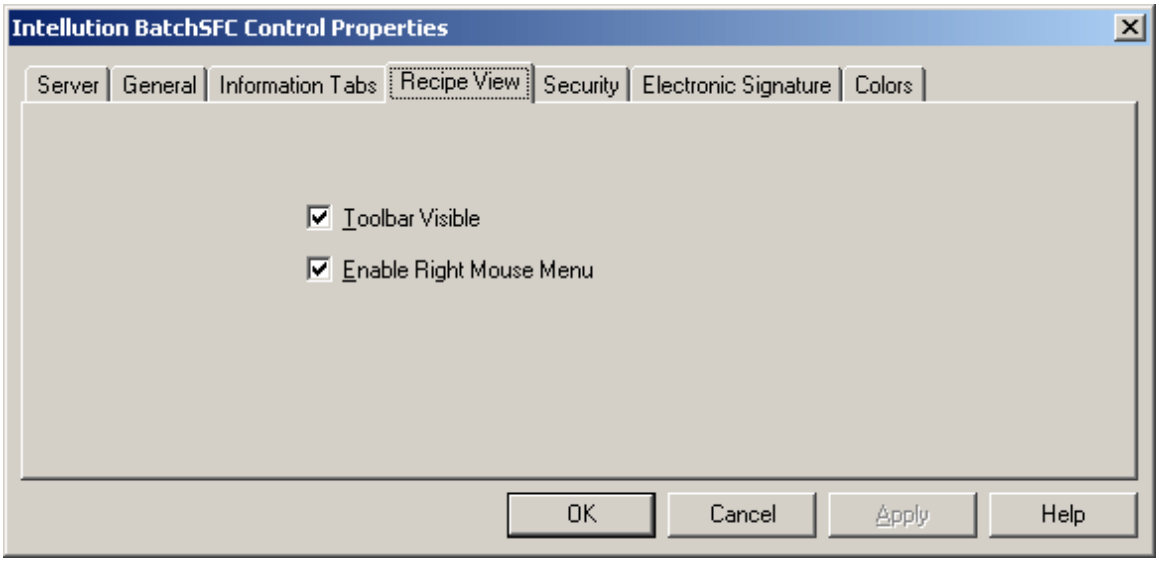
SFC Control Information Tabs Properties	
Property	**Syntax**
BindingTabVisible Sets whether or not to display the Binding tab in the SFC Information View.	**C++ Syntax:** **BOOL** CBatchSFC::GetBindingTabVisible(); **void** CBatchSFC::SetBindingTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*BindingTabVisible[= *boolvalue*]
InformationTabsVisible Sets whether or not to display the SFC Information View in the BatchSFC Control.	**C++ Syntax:** **BOOL** CBatchSFC::GetInformationTabsVisible(); **void** CBatchSFC::SetInformationTabsVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*InformationTabsVisible[= *boolvalue*]

SFC Control Information Tabs Properties	
Property	**Syntax**
ParametersTabVisible Sets whether or not to display the Parameters tab in the SFC Information View.	**C++ Syntax:** **BOOL** CBatchSFC::GetParametersTabVisible(); **void** CBatchSFC::SetParametersTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ParametersTabVisible[= *boolvalue*]
RecipeInfoTabVisible Sets whether or not to display the Recipe Information tab in the SFC Information View.	**C++ Syntax:** **BOOL** CBatchSFC::GetRecipeInfoTabVisible(); **void** CBatchSFC::SetRecipeInfoTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RecipeInfoTabVisible[= *boolvalue*]
ReportsTabVisible Sets whether or not to display the Reports tab in the SFC Information View.	**C++ Syntax:** **BOOL** CBatchSFC::GetReportsTabVisible(); **void** CBatchSFC::SetReportsTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ReportsTabVisible[= *boolvalue*]

Recipe View Property Page

The Recipe View property page lets the developer choose whether to display the Recipe View toolbar near the top of the SFC View. Refer to the BatchSFC ActiveX Control section for an illustration of the SFC View.

The following figure shows the Recipe View property page.



Recipe View Property Page

The following table lists the item that is available on the Recipe View property page.

BatchSFC Control Recipe View Property Page		
Item	**Description**	**Associated Property**
Toolbar Visible check box	Sets whether or not the Recipe View toolbar is displayed in the BatchSFC Control.	ViewToolBarVisible
Enable Right Mouse Menu check box	Sets whether or not the right-mouse menu is available to the operator at run-time.	EnableRightMouseMenu

Recipe View Properties

The following table provides the C++ and Visual Basic syntax for the properties that control the Recipe View settings for the BatchSFC control.

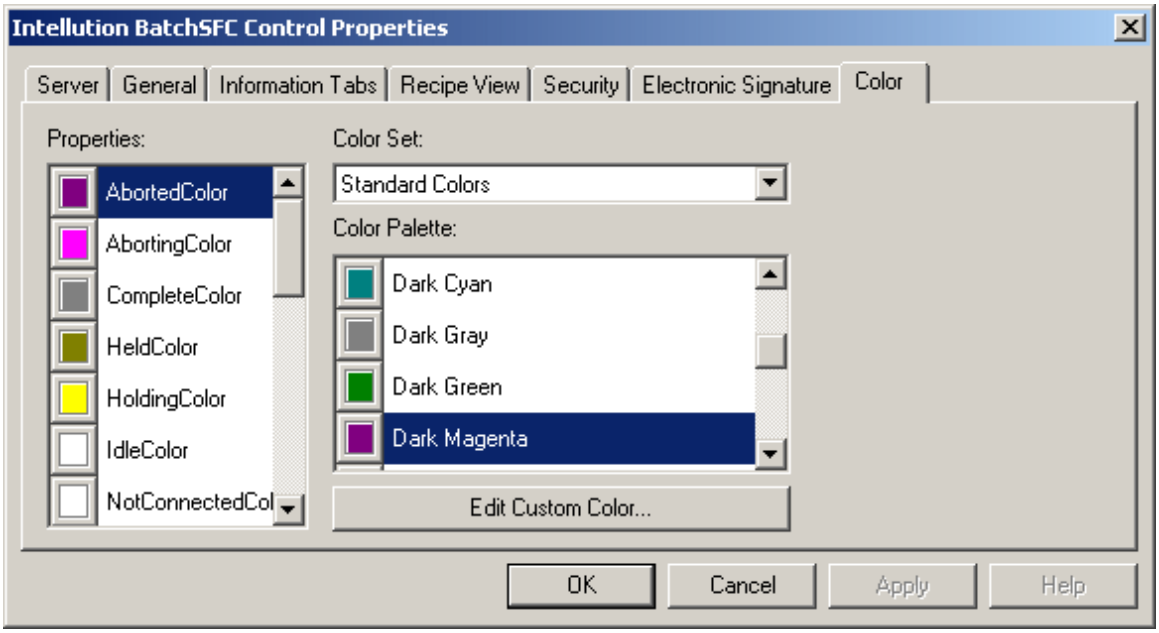
BatchSFC Control Recipe View Properties	
Property	**Syntax**
ViewToolbarVisible Sets whether or not to display the Recipe View toolbar.	**C++ Syntax:** **BOOL** CBatchSFC::GetViewToolbarVisible(); **void** CBatchSFC::SetViewToolbarVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ViewToolbarVisible[= *boolvalue*]

BatchSFC Control Recipe View Properties	
Property	**Syntax**
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchSFC::GetEnableRightMouseMenu(); **void** CBatchSFC::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableRightMouseMenu[= *boolvalue*]

Colors Property Page

BatchSFC Control Colors Property Page

The Colors property page lets you set the color for the BatchSFC control states. You can choose a color from the displayed color chart or you can use the System Color drop-down list box to display and select from a pre-defined system color chart. The Colors property page, shown in the following figure, is available at design time and at run time.



Colors Property Page

The following table lists the items that are available on the Colors property page.

Colors Property Page		
Item	**Description**	**Corresponding Properties**
Properties list box	Lists all the available properties for which you can set the color.	AbortedColor AbortingColor CompleteColor HeldColor HoldingColor IdleColor NotConnected RestartingColor RunningColor StartingColor StoppedColor StoppingColor TransitionAbortedColor TransitionArmedColor TransitionFiringColor TransitionHoldColor TransitionIdleColor TransitionStoppedColor
Color Set drop-down list	List the colors sets that you can choose from. For example: Windows Standard Colors, Windows System Colors.	None
Color Palette list box	Displays the available colors that you can assign to a property. Double-click the <Custom> option to display the Color dialog box and add a custom color to the Color Palette list.	None
Edit Custom Color button	Click this button to display the Color dialog box and add a custom color to the Color Palette list.	None

Colors Properties

The following table provides the C++ and Visual Basic syntax for the properties that control the colors in the BatchSFC control.

BatchSFC Control Colors Properties	
Property	**Syntax**
AbortedColor Sets the color of the Aborted phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetAbortedColor(); **void** CBatchSFC::SetAbortedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AbortedColor[= *color%*]
AbortingColor Sets the color of the Aborting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetAbortingColor(); **void** CBatchSFC::SetAbortingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AbortingColor[= *color%*]
CompleteColor Sets the color of the Complete phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetCompleteColor(); **void** CBatchSFC::SetCompleteColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CompleteColor[= *color%*]
HeldColor Sets the color of the Held phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetHeldColor(); **void** CBatchSFC::SetHeldColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HeldColor[= *color%*]
HoldingColor Sets the color of the Holding phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetHoldingColor(); **void** CBatchSFC::SetHoldingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*HoldingColor[= *color%*]

BatchSFC Control Colors Properties	
Property	**Syntax**
IdleColor Sets the color of the Idle phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetIdleColor(); **void** CBatchSFC::SetIdleColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*IdleColor[= *color%*]
NotConnectedColor Sets the color of the Not Connected phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetNotConnectedColor(); **void** CBatchSFC::SetNotConnectedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*NotConnectedColor[= *color%*]
RestartingColor Sets the color of the Restarting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetRestartingColor(); **void** CBatchSFC::SetRestartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RestartingColor[= *color%*]
RunningColor Sets the color of the Running phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetRunningColor(); **void** CBatchSFC::SetRunningColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RunningColor[= *color%*]
StartingColor Sets the color of the Starting phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetStartingColor(); **void** CBatchSFC::SetStartingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StartingColor[= *color%*]

BatchSFC Control Colors Properties	
Property	**Syntax**
StoppedColor Sets the color of the Stopped phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetStoppedColor(); **void** CBatchSFC::SetStoppedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StoppedColor[= *color%*]
StoppingColor Sets the color of the Stopping phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetStoppingColor(); **void** CBatchSFC::SetStoppingColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StoppingColor[= *color%*]
TransitionAbortedColor Sets the color of the Transition Aborted phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionAbortedColor(); **void** CBatchSFC::SetTransitionAbortedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionAbortedColor[= *color%*]
TransitionArmedColor Sets the color of the Transition Armed phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionArmedColor(); **void** CBatchSFC::SetTransitionArmedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionArmedColor[= *color%*]
TransitionFiringColor Sets the color of the Transition Firing phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionFiringColor(); **void** CBatchSFC::SetTransitionFiringColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionFiringColor[= *color%*]

BatchSFC Control Colors Properties	
Property	**Syntax**
TransitionHoldColor Sets the color of the Transition Hold phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionHoldColor(); **void** CBatchSFC::SetTransitionHoldColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionHoldColor[= *color%*]
TransitionIdleColor Sets the color of the Transition Idle phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionIdleColor(); **void** CBatchSFC::SetTransitionIdleColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionIdleColor[= *color%*]
TransitionStoppedColor Sets the color of the Transition Stopped phase state.	**C++ Syntax:** **OLE_COLOR** CBatchSFC::GetTransitionStoppedColor(); **void** CBatchSFC::SetTransitionStoppedColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TransitionStoppedColor[= *color%*]

Color Property Examples

The following show examples for setting color properties.

C++ Example

```
OLE_COLOR StartingColor = pSFC->GetStartingColor();
OLE_COLOR newStartingColor = 0x0; // black
pSFC->SetStartingColor(newStartingColor);
```

Visual Basic Example

```
Static SFC_OriginalStartingColor As Long
SFC_OriginalStartingColor = SFC1.StartingColor ' get the current color
SFC1.StartingColor = &H808080        ' set to dark gray
```

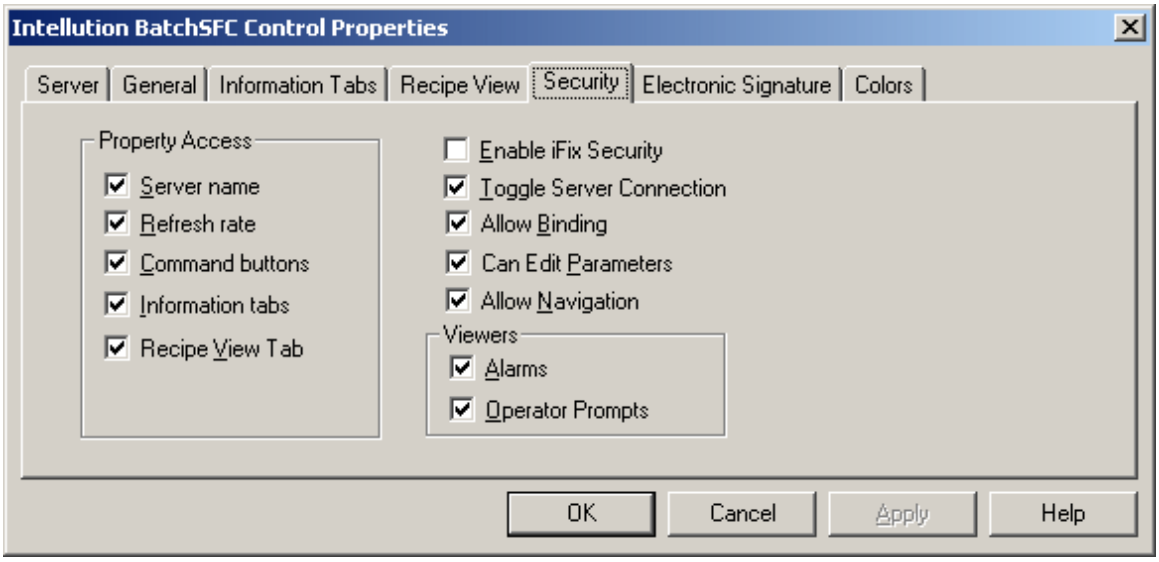
Security Property Page

Security Property Page

The Security property page lets the developer configure the security settings for the BatchSFC control, such as which properties the operator can configure. The Security property page is available only at design time.

If the developer turns off the refresh rate in the security property page, you cannot change the refresh rate at run time from the Server property page dialog box. At run time, however, a developer can change the refresh rate through an automation function.

The following figure shows the Security property page.



Security Property Page

The following table lists the items that are available on the Security property page.

BatchSFC Control Security Property Page		
Item	**Description**	**Associated Property**
Server Name check box	When selected, lets the operator change the server name on the Server property page. When deselected, prohibits the end user from changing the server name.	ServerEditEnabled
Refresh Rate check box	When selected, lets the operator modify the alarms and process refresh rate on the Server property page. When deselected, data is still refreshed at the set time, but the operator cannot change the refresh rate.	RefreshRateEditEnabled

BatchSFC Control Security Property Page		
Item	**Description**	**Associated Property**
Command Buttons checkbox	Set whether or not the operator can execute commands using the Command buttons.	CommandBtnsEditEnabled
Information Tabs check box	Sets whether or not the operator can edit information on the Information tabs.	InfotabEditEnabled
Recipe View Tab check box	Sets whether or not the operator can edit the SFC View.	ViewEditEnabled
Enable iFIX Security check box	Sets whether or not the BatchSFC ActiveX control uses iFIX security to check if the current user is authorized to execute a command. ***IMPORTANT:*** *If you enabled electronic signatures for a command on the Electronic Signature tab, then iFIX security is bypassed. Windows security is checked instead. Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	EnableIFIXSecurity
Toggle Server Connection check box	When selected, lets the operator connect and disconnect from the VBIS Server by selecting the Connection icon. The Connection icon appears at the bottom-right corner of the control on the status bar.	ToggleConnectionEnabled
Allow Binding check box	When selected, lets the operator perform unit binding.	AllowBinding
Can Edit Parameters check box	When selected, lets the operator edit parameters on the Parameters tab.	AllowEditParameters
Allow Navigation check box	When selected, allows the operator to navigate up and down in the recipe hierarchy.	AllowViewNavigation
View Alarms check box	When selected, allows the operator to view alarms.	ViewAlarms

BatchSFC Control Security Property Page		
Item	**Description**	**Associated Property**
View Operator Prompts check box	When selected, allows the operator to view operator prompts.	View OperatorPrompts

Security Properties

The following table provides the C++ and Visual Basic syntax for the properties that control the security settings for properties on the BatchSFC control.

BatchSFC Control Security Properties	
Property	**Syntax**
AllowBinding Sets whether or not the operator can perform unit binding.	**C++ Syntax:** BOOL CBatchSFC::GetAllowBinding(); void CBatchSFC::SetAllowBinding(BOOL value); **Visual Basic Syntax:** [*form.*]*Control.*AllowBinding[= *boolvalue*]
AllowEditParameters Sets whether or not the operator can edit parameters on the Parameters tab.	**C++ Syntax:** **BOOL** CBatchSFC::GetAllowEditParameters(); **void** CBatchSFC::SetAllowEditParameters(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AllowEditParameters[= *boolvalue*]
AllowViewNavigation Sets whether or not the operator can navigate up and down in the recipe hierarchy.	**C++ Syntax:** **BOOL** CBatchSFC::GetAllowViewNavigation(); **void** CBatchSFC::SetAllowViewNavigation(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*AllowViewNavigation[= *boolvalue*]

BatchSFC Control Security Properties	
Property	**Syntax**
CommandBtnsEditEnabled Sets whether or not the operator can execute commands using the command toolbar.	**C++ Syntax:** **BOOL** CBatchSFC::GetCommandBtnsEditEnabled(); **void** CBatchSFC::SetCommandBtnsEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*CommandBtnsEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchSFC::GetEnableIFIXSecurity(); **void** CBatchSFC::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableIFIXSecurity[= *boolvalue*]
InfotabEditEnabled Sets whether or not the operator can edit the properties on the Information Tabs tab.	**C++ Syntax:** **BOOL** CBatchSFC::GetInfotabEditEnabled(); **void** CBatchSFC::SetInfotabEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*InfotabEditEnabled[= *boolvalue*]
RecipeViewEditEnabled Sets whether or not the operator can edit the properties on the Recipe View tab.	**C++ Syntax:** **BOOL** CBatchSFC::GetRecipeViewEditEnabled(); **void** CBatchSFC::SetRecipeViewEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RecipeViewEditEnabled[= *boolvalue*]

BatchSFC Control Security Properties	
Property	**Syntax**
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchSFC::GetRefreshRateEditEnabled(); **void** CBatchSFC::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRateEditEnabled[= *boolvalue*]
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchSFC::GetServerEditEnabled(); **void** CBatchSFC::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ServerEditEnabled [= *boolvalue*]
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchSFC::GetToggleConnectionEnabled(); **void** CBatchSFC::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= *boolvalue*]
ViewAlarms Sets whether or not the operator can view the alarms using the Alarms button on the command toolbar.	**C++ Syntax:** **BOOL** CBatchSFC::GetViewAlarms(); **void** CBatchSFC::SetViewAlarms(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewAlarms[= *boolvalue*]

BatchSFC Control Security Properties	
Property	**Syntax**
ViewOperatorPrompts Sets whether or not the operator can view the operator prompts using the Operator Prompts button on the command toolbar.	**C++ Syntax:** **BOOL** CBatchSFC::GetViewOperatorPrompts(); **void** CBatchSFC::SetViewOperatorPrompts(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewOperatorPrompts[= *boolvalue*]

Electronic Signature Property Page

The Electronic Signature property page specifies the electronic signature requirements associated with each of the commands available on the BatchSFC ActiveX control. The signature requirements define whether or not a user or users must "sign-off" on a command before it is performed. You can define the following signature types for each command:

- None (no signature required)
- Performed By (only "Performed By" signature required)
- Performed By/Verified By (both "Performed By" and "Verified By" signatures required)

If you do not define a signature type, no signature is requested when the command is performed; NONE is the default signature type if you do not select one. If you do not want to define these signature types individually, you can specify one signature type for all commands by using the default signature row.

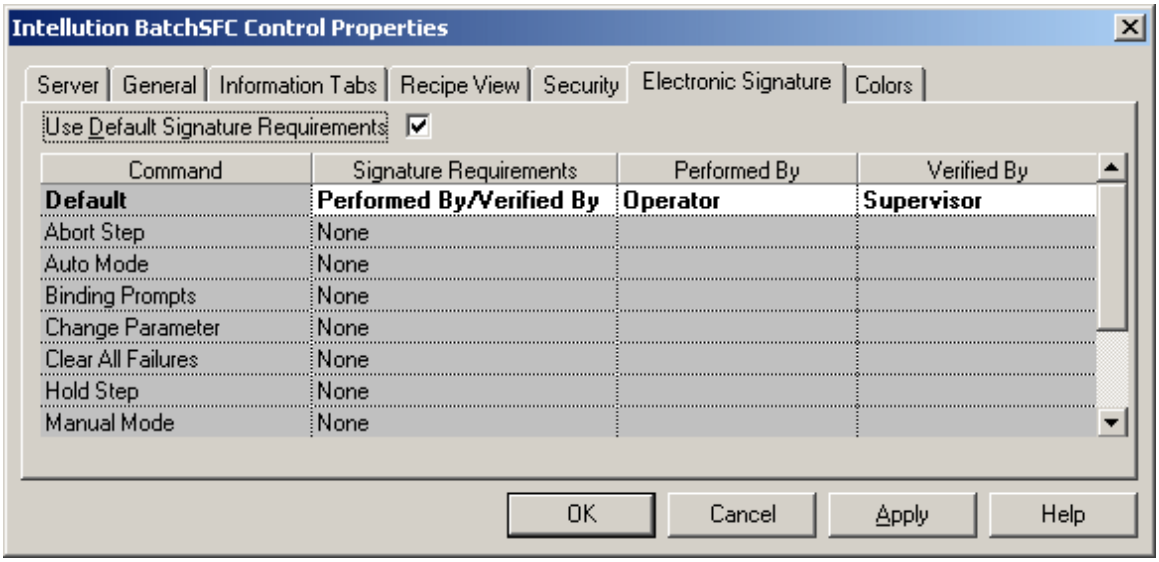
If you specified a Performed By or Performed By/Verified By signature type, Windows security groups are used to validate the user who performs the command. The user must be a member of the specified group to perform or verify the command. The same user *cannot* sign both the Performed By and Verified By signature requirements, even if that user resides in both the Performed By and Verified By groups. What that means is that the user performing a command cannot be the same user who verifies that command. Likewise, the user verifying a command cannot be the same user who performed that command.

To enter the groups from which the user is authorized, you specify the Performed By and Verified By groups in the Electronic Signature property page. This property page is enabled at design time only. The number of times the user can attempt to enter a failed signature is defined by your administrator through Windows security. Refer to account policy information in your Microsoft Windows Help system for more details.

Batch Execution supports both Local and Domain groups. Upon receiving a request to validate a signature, Batch Execution checks the local computer for the specified group. If the group is not found, it then checks for a Domain level group to verify the user name and password. You can choose to configure your security groups and user names on the Batch Execution Server computer or as part of your plant's overall Domain Security Configuration.

When the ActiveX control captures an electronic signature, the signature is recorded in the BATCH_CMD_SIGNATURE_SUCCESS table, along with the computer name and time stamp of the signature, as well as other data. If a signature fails, the event is not logged, nor is the command performed. For more information on the fields in this table refer to the BATCH_CMD_SIGNATURE_SUCCESS Table section.

The following figure shows the Electronic Signature property page for the BatchSFC control.



BatchSFC Electronic Signature Property Page

The following table lists the items that are available on the Electronic Signature property page.

Electronic Signature Property Page		
Item	**Description**	**Associated Property or Method**
Use Default Signature Requirements check box	When selected the user applies a single signature requirement for all methods on a control.	UseDefaultSignatureRequirements
Signature Requirements Grid Control	Contains the signature requirements for each command or the default.	GetSignatureRequirements SetSignatureRequirements

Configuring Electronic Signature Properties

The steps that follow explain how to configure the electronic signature properties for the BatchSFC ActiveX control.

NOTE: *You can only configure the electronic signature properties in design mode. You cannot configure these properties in run mode.*

To configure the electronic signature properties for BatchSFC:

1. Open the control in any design-time OLE container.
2. Right-click the control and select Properties from the pop-up menu. The control's property pages appear.
3. Select the Electronic Signature property page.
4. Select the Use Default Signature Requirements check box if you want to use a default signature.
5. Select the signature type for the default, or for each command listed (if default is not selected):
    - Performed By
    - Performed By / Verified By
    - None
6. Select the Windows security group for each Performed By and Verified By signature specified.

    NOTE: *Right-click on the edit box and select the Browse option to open the Windows Security Groups dialog box. Make sure that the cursor is not displaying in the text box when you right-click on it. Select a group from the Windows Security Groups dialog box and click OK. You can only browse local security groups, but you can manually enter a domain group.*

7. Click OK to save your changes.

Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchSFC control.

BatchSFC Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements If this property is True, it indicates that the default signature settings will be used for all commands for the ActiveX control. If this property is False, then the signature setting of each command is used. The default value of UseDefaultSignatureRequirements is True. The default signature requirements are None. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchSFC:GetUseDefaultSignatureRequirements();**void** CBatchSFC::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

BatchSFC Control Methods

The BatchSFC control supports the following methods:

- AbortStep Method
- AboutBox Method
- AutoStep Method
- ClearAllFailures Method
- ConnectToServer Method
- DisconnectFromServer Method
- GetIVBIS Method
- GetRecipeZoom Method
- HoldStep Method
- ManualStep Method
- RestartStep Method
- SetIVBISPointer Method
- SetCurrentBatch Method
- SetCurrentRecipeStep Method

- SetRecipeZoom Method
- StartStep Method
- StopStep Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method
- ZoomFull Method
- ZoomIn Method
- ZoomNormal Method
- ZoomOut Method

AbortStep Method

Description

Issues the Abort command on the currently selected batch.

C++ Syntax

```
void CBatchSFC::AbortStep();
```

Visual Basic Syntax

```
[form.]SFC1.AbortStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.AbortStep();
```

Visual Basic Example

```
SFC1.AbortStep
```

AboutBox Method

Description

Displays the About box.

C++ Syntax

```
void CBatchSFC::AboutBox()
```

Visual Basic Syntax

```
[form.]SFC1.AboutBox()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.AboutBox();
```

Visual Basic Example

```
SFC1.AboutBox
```

AutoStep Method

Description

Issues the Auto command on the currently selected recipe step.

C++ Syntax

```
void CBatchSFC::AutoStep()
```

Visual Basic Syntax

```
[form.]SFC1.AutoStep()
```

Return Type

Void.

C++ Example

```
SFC1.AutoStep();
```

Visual Basic Example

```
SFC1.AutoStep
```

ClearAllFailures Method

Description

Occurs when the operator issues a ClearAllFailures command.

C++ Syntax

```
void CBatchSFC::ClearAllFailures();
```

Visual Basic Syntax

```
[form.]SFC1.ClearAllFailures()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ClearAllFailures();
```

Visual Basic Example

```
SFC1.ClearAllFailures
```

ConnectToServer Method

Description

Establishes the connection to the VBIS Server.

C++ Syntax

```
BOOL CBatchSFC::ConnectToServer();
```

Visual Basic Syntax

```
[form.]SFC1.ConnectToServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if a connection is made.
- 0 if a connection is not made.

C++ Example

```
BOOL result = pSFC->ConnectToServer();
```

Visual Basic Example

```
SFC1.ConnectToServer
```

DisconnectFromServer Method

Description

Disconnects from the currently connected VBIS Server.

C++ Syntax

```
BOOL CBatchSFC::DisconnectFromServer();
```

Visual Basic Syntax

```
[form.]SFC1.DisconnectFromServer() As Boolean
```

Parameters

None.

Return Type

Boolean.

- 1 if the VBIS Server is successfully disconnected.
- 0 if the VBIS Server is not successfully disconnected.

C++ Example

```
BOOL result = pSFC->DisconnectFromServer();
```

Visual Basic Example

```
SFC1.DisconnectFromServer
```

GetIVBIS Method

Description

Returns the IVBIS pointer to which the control is connected. If the control is not connected, it returns NULL. Call Release () on the pointer when done with it.

C++ Syntax

```
LPDISPATCH CBatchSFC::GetIVBIS();
```

Visual Basic Syntax

```
[form.]SFC1.GetIVBIS() As Objects
```

C++ Example

```
IVBIS8* pIVBIS = pSFC->GetIVBIS();
...///Release it when done.
pIVBIS->Release();
```

Visual Basic Example

```
Set VBISP = SFC1.GetIVBIS
...
Set VBISP = Nothing
```

GetRecipeZoom Method

Description

Gets the current zoom values for the recipe displayed in the SFC screen. Typically, you would call the GetRecipeZoom method, save the values, and later call the SetRecipeZoom method with those saved values.

NOTE: *The zoom values are based on the dimensions of the displayed recipe. If you try to use the same values to display a different recipe, the result could be different.*

C++ Syntax

```
VARIANT_BOOL GetRecipeZoom(long* x, long* y);
```

Visual Basic Syntax

```
[form.]Control.GetRecipeZoom(x As Long, y As Long) As Boolean
```

Parameters

Parameter	Description
x	A number that represents the horizontal zoom value. This value is greater than 48.
y	A number that represents the vertical zoom value. This value is greater than 32.

Return Type

Boolean.

- 0 = zoom values not in use
- 1 = zoom values retrieved

C++ Example

```
BOOL result = pBatchSFC->GetRecipeZoom(long* x, long* y);
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchSFC1.GetRecipeZoom(long* x, long* y)
```

HoldStep Method

Description

Issues the Hold command on the currently selected batch.

C++ Syntax

```
void CBatchSFC::HoldStep();
```

Visual Basic Syntax

```
[form.]SFC1.HoldStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.HoldStep();
```

Visual Basic Example

```
SFC1.HoldStep
```

ManualStep Method

Description

Issues the Manual command on the currently selected batch.

C++ Syntax

```
void CBatchSFC::ManualStep();
```

Visual Basic Syntax

```
[form.]SFC1.ManualStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ManualStep();
```

Visual Basic Example

```
SFC1.ManualStep
```

RestartStep Method

Description

Issues the Restart command on the currently selected batch.

C++ Syntax

```
void CBatchSFC::RestartStep();
```

Visual Basic Syntax

```
[form.]SFC1.RestartStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.RestartStep();
```

Visual Basic Example

```
SFC1.RestartStep
```

SetIVBISPointer Method

Description

Sets the internal VBIS pointer of the control to the given VBIS pointer. VBIS Server name is also passed, but only for display information. Make sure that the VBIS Server name is correct.

C++ Syntax

```
BOOL CBatchSFC::SetIVBISPointer(IDispatch* pIVBIS, LPCTSTR lpstrServerName);
```

Visual Basic Syntax

```
[form.]SFC1.SetIVBISPointer(pIVBIS As Object, lpstrServerName As String) As Boolean
```

Parameters

Parameter	Description
pIVBIS	The pointer to the VBIS Server.
lpstrServerName	The name of the VBIS Server.

Return Type

Boolean.

- 1 if successful.
- 0 if not successful.

C++ Example

```
CString VBISServerName = pSFC->GetVBISServerName();
// get pIVBIS
IVBIS8* pIVBIS=pSFC->GetIVBIS();
pSFC->SetIVBISPointer(pIVBISP, VBISServerName);
pIVBIS->Release();
```

Visual Basic Example

```
Dim VBISServerName as String
VBISServerName = SFC1.VBISServerName
Set VBISP=SFC1.GetIVBIS
' get VBISP
SFC1.SetIVBISPointer VBISP, VBISServerName
Set VBISP = Nothing
```

NOTE: *These examples set the IVBIS pointer from one control and use it to set the IVBIS pointer of the second control.*

SetCurrentBatch Method

Description

Given the batch serial number, displays the specified batch on the SFC screen.

C++ Syntax

```
VARIANT_BOOL SetCurrentBatch(long lBatchSerialNumber);
```

Visual Basic Syntax

```
[form.]Control.SetCurrentBatch(lBatchSerialNumber As Long)
```

Parameters

Parameter	Description
lBatchSerialNumber	Batch serial number associated with this batch, generated internally by the server. This unique identification number is assigned to each batch by Batch Execution.

Return Type

Boolean.

- 0 = unsuccessful at setting the current batch
- 1 = successful at setting the current batch

C++ Example

```
BOOL result = pBatchSFC->SetCurrentBatch(123);
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchSFCt1.SetCurrentBatch(123)
```

SetCurrentRecipeStep Method

Description

Passes in a tab-delimited, fully qualified recipe step and displays it on the SFC screen. For example, a tab-delimited, fully qualified recipe step might be:

```
BASE:1	MAKE_BASE:1	ADD_INGS:1
```

where the space between the entries is a tab character.

C++ Syntax

```
VARIANT_BOOL SetCurrentRecipeStep(BSTR bstrFullRecipeStepName);
```

Visual Basic Syntax

```
[form.]Control.SetCurrentRecipeStep(bstrFullRecipeStepName As String) As Boolean
```

Parameters

Parameter	Description
bstrFullRecipeStepName	The fully qualified, recipe step name, in tab-delimited format.

Return Type

Boolean.

- 0 = unsuccessful at setting the current recipe
- 1 = successful at setting the current recipe

C++ Example

```
BOOL result = pBatchSFC->SetCurrentRecipeStep("BASE:1     MAKE_BASE:1     ADD_INGS:1");
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchSFCt1.SetCurrentRecipeStep("BASE:1     MAKE_BASE:1     ADD_INGS:1")
```

NOTE: *The spaces between BASE:1, MAKE_BASE:1, and ADD_INGS:1 represents tab characters.*

SetRecipeZoom Method

Description

Sets current zoom values for the recipe displayed in the SFC screen. Typically, you would call the GetRecipeZoom method, save the values, and later call the SetRecipeZoom method with those saved values.

NOTE: *The zoom values are based on the dimensions of the displayed recipe. If you try to use the same values to display a different recipe, the result could be different.*

C++ Syntax

```
VARIANT_BOOL SetRecipeZoom(long* x, long* y);
```

Visual Basic Syntax

```
[form.]Control.SetRecipeZoom(x As Long, y As Long) As Boolean
```

Parameters

Parameter	Description
x	A number that represents the horizontal zoom value. This value is greater than 48.
y	A number that represents the vertical zoom value. This value is greater than 32.

Return Type

Boolean.

- 0 = zoom values not set
- 1 = zoom values successfully set

C++ Example

```
BOOL result = pBatchSFC->SetRecipeZoom(50, 50);
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchSFC1.SetRecipeZoom(50, 50)
```

StartStep Method

Description

Issues the Start command on the currently selected batch.

C++ Syntax

```
void CSFC::StartStep();
```

Visual Basic Syntax

```
[form.]SFC1.StartStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.StartStep();
```

Visual Basic Example

```
SFC1.StartStep
```

StopStep Method

Description

Issues the Stop command on the currently selected batch.

C++ Syntax

```
void CSFC::StopStep();
```

Visual Basic Syntax

```
[form.]SFC1.StopStep()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.StopStep();
```

Visual Basic Example

```
SFC1.StopStep
```

GetCommandSignatureRequirements Method

Description

Gets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchSFC::GetCommandSignatureRequirements(long Command, long*
SignatureRequirements, BSTR* PerformedBy, BSTR* VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.GetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchSFC the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortStep = 1 • bcAutoMode = 2 • bcBindingPrompts = 3 • bcChangeParameter = 4 • bcClearAllFailures = 5 • bcHoldStep = 6 • bcManualMode = 7 • bcOperatorPrompts = 8 • bcRestartStep = 9 • bcStartStep = 10 • bcStopStep = 11
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

typedef enum _tagCommandID{
     bcDefault = 0,
     bcAbortStep = 1,
     bcAutoMode = 2,
     bcBindingPrompts = 3,
     bcChangeParameter = 4,
     bcClearAllFailures = 5,
     bcHoldStep = 6,
     bcManualMode = 7,
     bcOperatorPrompts = 8,
     bcRestartStep = 9,
     bcStartStep = 10,
     bcStopStep = 11,
} COMMANDID;

     BSTR bstrPerformedBy;
     BSTR bstrVerifiedBy;
     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     // Example of reading the signature requirements for the
     // Start Step Command.
     // Note, that if the UseDefaultSignatureRequirements is TRUE
     // then the setting for the Default command is used,
     // else the setting for the start batch command is used.

     if (m_pBatchSFC->GetUseDefaultSignatureRequirements ())
     {
          lCommand = bcDefault;
     }
     else
     {
          lCommand = bcStartStep;
     }

     m_pBatchSFC->GetCommandSignatureRequirements ( lCommand, &lSignatureType,
&bstrPerformedBy, &bstrVerifiedBy);

     strPerformedBy = bstrPerformedBy;
     strVerifiedBy = bstrVerifiedBy;

     ::SysFreeString(bstrPerformedBy);
     ::SysFreeString(bstrVerifiedBy);

     if (lSignatureType == stNone)
```

```
     {
     AfxMessageBox ("No Signature requirements for the Start Step command");
     }
     else if (lSignatureType == stPerformedBy)
     {
     AfxMessageBox ("Signature requirements for the Start Step command are Perform By: "
+ strPerformedBy);
     }
     else if (lSignatureType == stPerformedByVerifiedBy)
     {
     AfxMessageBox ("Signature requirements for the Start Step command are Perform By: "
+ strPerformedBy + " Verify By: " + strVerifiedBy);
     }
```

Visual Basic Example

```
Private Sub Command3_Click()
Dim Command As SFCLib.COMMANDID
Dim SignatureType As SFCLib.SignatureType
Dim bRetVal As Boolean
Dim strPerformedBy As String
Dim strVerifiedBy As String

' example of reading the signature requirements for the Start step Command
' note that if the UseDefaultSignatureRequirements is TRUE then the setting for the
Default command is used,
' else the setting for the start batch command is used.

If BatchList1.UseDefaultSignatureRequirements Then
    Command = bcDefault
Else
    Command = bcStartStep
End If

SFC1.GetCommandSignatureRequirements Command, SignatureType, strPerformedBy,
strVerifiedBy

If SignatureType = stNone Then
    MsgBox ("No Signature requirements for the Start Step Command")
ElseIf SignatureType = stPerformedBy Then
    MsgBox ("Signature requirements for the Start Step Command are Perform By: " +
strPerformedBy)
ElseIf SignatureType = stVerifiedBy Then
    MsgBox ("Signature requirements for the Start Step Command are Perform By: " +
strPerformedBy + " Verified By: " + strVerifiedBy)
End If
```

SetCommandSignatureRequirements Method

Description

Sets the signature requirements for the command. The signature can be defined with no signature required, only a "Performed By" signature required, or both "Performed By" and "Verified By" signatures required.

C++ Syntax

```
BOOL CBatchSFC::SetCommandSignatureRequirements(Long Command, Long SignatureRequirements,
CString PerformedBy, CString VerifiedBy);
```

Visual Basic Syntax

```
[form.]Control.SetCommandSignatureRequirements(Command As COMMANDID,
SignatureRequirements As SIGNATURETYPE, PerformedBy As String, VerifiedBy As String) As
Boolean
```

Parameters

Parameter	Description
Command	The command that executes from the ActiveX control. For BatchSFC the enumerated values (of type COMMANDID) for these commands are: • bcDefault = 0 • bcAbortStep = 1 • bcAutoMode = 2 • bcBindingPrompts = 3 • bcChangeParameter = 4 • bcClearAllFailures = 5 • bcHoldStep = 6 • bcManualMode = 7 • bcOperatorPrompts = 8 • bcRestartStep = 9 • bcStartStep = 10 • bcStopStep = 11
SignatureRequirements	The enumerated value (of type SIGNATURETYPE) of the signature required: • stNone = 0 • stPerformedBy = 1 • stPerformedByVerifiedBy = 2
PerformedBy	The group from which the user must be a member in order to enter the Performed By signature. The data type is String.
VerifiedBy	The group from which the user must be a member in order to enter the Verified By signature. The data type is String.

Return Type

Boolean.

- TRUE if the function succeeds.
- FALSE if the function fails. For example, if the programmer passes an invalid command in the first parameter, the function will fail.

C++ Example

```
// these are the constants that the methods use for signature
// type and command.
     typedef enum _tagSignatureType{
          stNone = 0,
          stPerformedBy = 1,
          stPerformedByVerifiedBy = 2,

     } SIGNATURETYPE;

typedef enum _tagCommandID{
     bcDefault = 0,
     bcAbortStep = 1,
     bcAutoMode = 2,
     bcBindingPrompts = 3,
     bcChangeParameter = 4,
     bcClearAllFailures = 5,
     bcHoldStep = 6,
     bcManualMode = 7,
     bcOperatorPrompts = 8,
     bcRestartStep = 9,
     bcStartStep = 10,
     bcStopStep = 11,
} COMMANDID;

     CString strPerformedBy;
     CString strVerifiedBy;
     long lCommand=bcDefault;
     long lSignatureType=stNone;

     strPerformedBy = _T("Operator");
     strVerifiedBy = _T("Supervisor");

     lCommand = bcStartStep;
     lSignatureType = stPerformedByVerifiedBy;
     m_pBatchSFC->SetCommandSignatureRequirements( lCommand, lSignatureType,
strPerformedBy, strVerifiedBy );
```

Visual Basic Example

```
' example to set a specific command signature requirement

Dim Command As SFCLib.COMMANDID
Dim SignatureType As SFCLib.SignatureType
Dim bRetVal As Boolean
Dim strPerformedBy As String
Dim strVerifiedBy As String
```

```
Command = bcStopStep
strPerformedBy = "Operator"
strVerifiedBy = "Supervisor"
SFC1.SetCommandSignatureRequirements Command, stPerformedByVerifiedBy, strPerformedBy,
strVerifiedBy
End Sub
```

ZoomFull Method

Description

Zooms the contents to fit the SFC display screen.

C++ Syntax

```
void CBatchSFC::ZoomFull()
```

Visual Basic Syntax

```
[form.]SFC1.ZoomFull()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ZoomFull();
```

Visual Basic Example

```
SFC1.ZoomFull
```

ZoomIn Method

Description

Zooms in the SFC display screen by 10%.

C++ Syntax

```
void CBatchSFC::ZoomIn()
```

Visual Basic Syntax

```
[form.]SFC1.ZoomIn()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ZoomIn();
```

Visual Basic Example

```
SFC1.ZoomIn
```

ZoomNormal Method

Description

Zooms to display the SFC in a legible size.

C++ Syntax

```
void CBatchSFC::ZoomNormal()
```

Visual Basic Syntax

```
[form.]SFC1.ZoomNormal()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ZoomNormal();
```

Visual Basic Example

```
SFC1.ZoomNormal
```

ZoomOut Method

Description

Zooms out the SFC display screen by 10%.

C++ Syntax

```
void CBatchSFC::ZoomOut()
```

Visual Basic Syntax

```
[form.]SFC1.ZoomOut()
```

Parameters

None.

Return Type

Void.

C++ Example

```
SFC1.ZoomOut();
```

Visual Basic Example

```
SFC1.ZoomOut
```

BatchSFC Control Events

The BatchSFC control generates the following events:

- AbortPressed Event
- AutoPressed Event
- ClearAllFailuresPressed Event
- ConnectedToServer Event
- DisconnectedFromServer Event
- HoldPressed Event
- ManualPressed Event
- RestartPressed Event
- ServerChanged Event
- StartPressed Event
- StopPressed Event

AbortPressed Event

Description

Occurs when the operator issues an AbortPressed command.

Event ID

7

C++ Syntax

```
void AbortPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event AbortPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

AutoPressed Event

Description

Occurs when the operator issues an AutoPressed command.

Event ID

6

C++ Syntax

```
void AutoPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event AutoPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

ClearAllFailuresPressed Event

Description

Occurs when the operator issues a ClearAllFailures command.

Event ID

8

C++ Syntax

```
void ClearAllFailuresPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event ClearAllFailuresPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

ConnectedToServer Event

Description

Occurs when the control has connected to the VBIS Server.

Event ID

10

C++ Syntax

```
void ConnectedToServer();
```

Visual Basic Syntax

```
Event ConnectedToSever()
```

DisconnectedFromServer Event

Description

Occurs when the control has disconnected from the VBIS Server.

Event ID

11

C++ Syntax

```
void DisconnectedFromServer();
```

Visual Basic Syntax

```
Event DisconnectedFromServer()
```

HoldPressed Event

Description

Occurs when the operator issues a Hold command.

Event ID

5

C++ Syntax

```
void HoldPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event HoldPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

ManualPressed Event

Description

Occurs when the operator issues a Manual command.

Event ID

4

C++ Syntax

```
void ManualPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event ManualPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

RestartPressed Event

Description

Occurs when the operator issues a Restart command.

Event ID

3

C++ Syntax

```
void RestartPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event RestartPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

ServerChanged Event

Description

Occurs when the control has switched VBIS Servers.

Event ID

9

C++ Syntax

```
void ServerChanged();
```

Visual Basic Syntax

```
Event ServerChanged()
```

StartPressed Event

Description

Occurs when the operator issues a Start command.

Event ID

1

C++ Syntax

```
void StartPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event StartPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

StopPressed Event

Description

Occurs when the operator issues a Stop command.

Event ID

2

C++ Syntax

```
void StopPressed(LPCTSTR bstrRecipe);
```

Visual Basic Syntax

```
Event StopPressed(bstrRecipe As String)
```

Parameters

Parameter	Description
bstrRecipe	The step in the SFC that is currently selected in the Steps drop-down list.

C++ Event Sink Map

The following shows an example of an event sink map.

```
BEGIN_EVENTSINK_MAP(CSFCDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(CSFCDlg)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 1 /* StartPressed */, OnStartPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 2 /* StopPressed */, OnStopPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 3 /* RestartPressed */, OnRestartPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 4 /* ManualPressed */, OnManualPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 5 /* HoldPressed */, OnHoldPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 6 /* AutoPressed */, OnAutoPressedSfcctrl1,
VTS_BSTR)
```

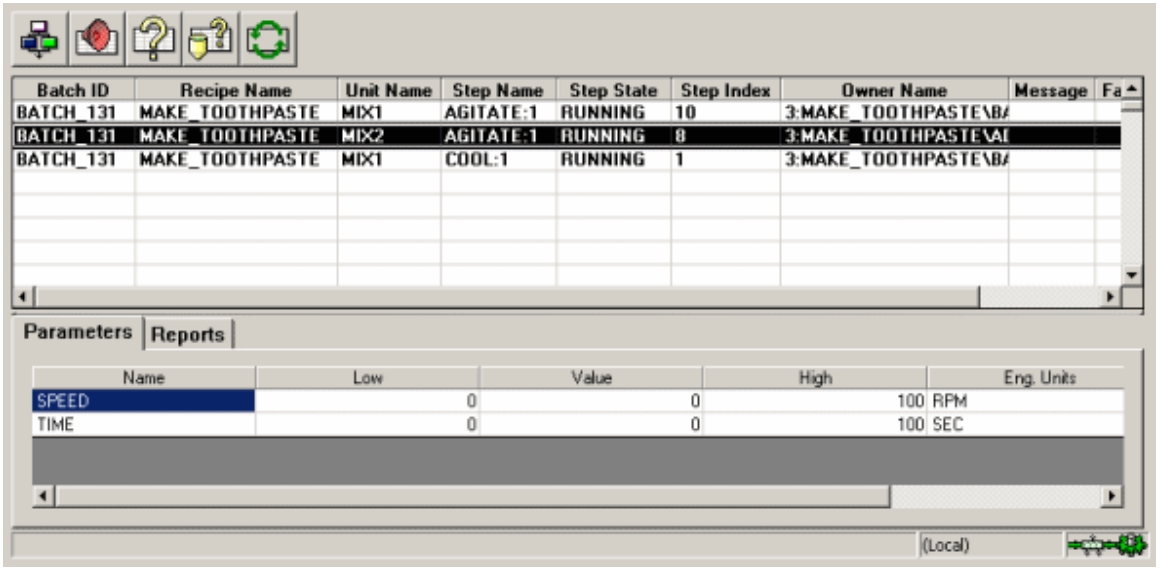
```
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 7 /* AbortPressed */, OnAbortPressedSfcctrl1,
VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 8 /* ClearAllFailuresPressed */,
OnClearAllFailuresPressedSfcctrl1, VTS_BSTR)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 9 /* ServerChanged */, OnServerChangedSfcctrl1,
VTS_NONE)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 10 /* ConnectedToServer */,
OnConnectedToServerSfcctrl1, VTS_NONE)
     ON_EVENT(CSFCDlg, IDC_SFCCTRL1, 11 /* DisconnectedFromServer */,
OnDisconnectedFromServerSfcctrl1, VTS_NONE)
     //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()
```

BatchActivePhaseList ActiveX Control

The "Intellution BatchActivePhaseList Control" provides functionality that's similar to the Phase Summary screen in the Batch Execution Client. The BatchActivePhaseList displays information about all of the active phases being executed by the Batch Server. You can select a batch and view the SFC screen, alarms, operator prompts, or binding prompts.

Designers can configure the BatchActivePhaseList control's GUI run-time appearance and functionality using the control's property pages. For example, the designer can disable command buttons, the right-click menu, or specific columns that the operator should not have access to. Refer to the Configuring the Batch Execution ActiveX Controls section for more information.

The following figure shows the BatchActivePhaseList control.



GE BatchActivePhaseList ActiveX Control

Developers can access the control programmatically through Visual Basic or Visual C++ using the control's properties and methods. The properties, methods, and events for the BatchActivePhaseList control are described in the following sections.

BatchActivePhaseList Control Properties

The following sections describe each property for the BatchActivePhaseList control. The properties are grouped into the following categories:

- Column properties
- Server properties
- Command Button properties
- Miscellaneous properties
- Security properties
- Electronic Signature properties
- Color properties
- Font properties

BatchActivePhaseList Control Columns Properties

The following table lists the properties that control the display of the columns in the BatchActivePhaseList control.

BatchActivePhaseList Column Properties	
Property	**Syntax**
BatchIDVisible Sets whether or not to display the Batch ID column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetBatchIDVisible(); **void** CBatchActivePhaseList::SetBatchIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDVisible[= *boolvalue*]
BatchIDFilter Sets the filter for the Batch ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetBatchIDFilter (); **void** CBatchActivePhaseList::SetBatchIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDFilter [= *text$*]
BatchIDHeaderText Specifies the header text for the Batch ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetBatchIDHeaderText (); **void** CBatchActivePhaseList::SetBatchIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDHeaderText [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
BatchIDWidth Sets the width of the Batch ID column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetBatchIDWidth(); **void** CBatchActivePhaseList::SetBatchIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchIDWidth[= *value!*]
BatchSerialNumberVisible Sets whether or not to display the Batch Serial Number column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetSerialNumberVisible(); **void** CBatchActivePhaseList::SetSerialNumberVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchSerialNumberVisible[= *boolvalue*]
BatchSerialNumberFilter Sets the filter for the Batch Serial Number column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetBatchSerialNumberFilter (); **void** CBatchActivePhaseList::SetBatchSerialNumberFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchSerialNumberFilter [= *text$*]
BatchSerialNumberHeaderText Specifies the header text for the Batch Serial Number column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetBatchSerialNumberHeaderText (); **void** CBatchActivePhaseList::SetBatchSerialNumberHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchSerialNumberHeaderText [= *text$*]
BatchSerialNumberWidth Sets the width of the Batch Serial Number column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetBatchSerialNumberWidth(); **void** CBatchActivePhaseList::SetBatchSerialNumberWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*BatchSerialNumberWidth[= *value!*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
RecipeNameVisible Sets whether or not to display the Recipe column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetRecipeNameVisible(); **void** CBatchActivePhaseList::SetRecipeNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeNameVisible[= *boolvalue*]
RecipeNameFilter Sets the filter for the Recipe Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetRecipeNameFilter (); **void** CBatchActivePhaseList::SetRecipeNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeNameFilter [= *text$*]
RecipeNameHeaderText Specifies the columns header text for the Recipe Name Header column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetRecipeNameHeaderText (); **void** CBatchActivePhaseList::SetRecipeNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeHeaderText [= *text$*]
RecipeNameWidth Sets the width of the Recipe Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetRecipeNameWidth(); **void** CBatchActivePhaseList::SetRecipeNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RecipeNameWidth[= *value!*]
UnitNameVisible Sets whether or not to display the Unit Name column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetUnitNameVisible(); **void** CBatchActivePhaseList::SetUnitNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameVisible[= *boolvalue*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
UnitNameFilter Sets the filter for the Unit Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetUnitNameFilter (); **void** CBatchActivePhaseList::SetUnitNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameFilter [= *text$*]
UnitNameHeaderText Specifies the header text for the Unit Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetUnitNameHeaderText (); **void** CBatchActivePhaseList::SetUnitNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameHeaderText [= *text$*]
UnitNameWidth Sets the width of the Unit Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetUnitNameWidth(); **void** CBatchActivePhaseList::SetUnitNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*UnitNameWidth[= *value!*]
ScheduledUnitNameVisible Sets whether or not to display the Scheduled Unit Name column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetScheduledUnitNameVisible(); **void** CBatchActivePhaseList::SetScheduledUnitNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ScheduledUnitNameVisible[= *boolvalue*]
ScheduledUnitNameFilter Sets the filter for the Scheduled Unit Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetScheduledUnitNameFilter (); **void** CBatchActivePhaseList::SetScheduledUnitNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ScheduledUnitNameFilter [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
ScheduledUnitNameHeaderText Specifies the header text for the Scheduled Unit Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetScheduledUnitNameHeaderText (); **void** CBatchActivePhaseList::SetScheduledUnitNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ScheduledUnitNameHeaderText [= *text$*]
ScheduledUnitNameWidth Sets the width of the Scheduled Unit Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetScheduledUnitNameWidth(); **void** CBatchActivePhaseList::SetScheduledUnitNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ScheduledUnitNameWidth[= *value!*]
StepNameVisible Sets whether or not to display the Step Name column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStepNameVisible(); **void** CBatchActivePhaseList::SetStepNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StepNameVisible[= *boolvalue*]
StepNameFilter Sets the filter for the Step Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepNameFilter (); **void** CBatchActivePhaseList::SetStepNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepNameFilter [= *text$*]
StepNameHeaderText Specifies the header text for the Step Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepNameHeaderText (); **void** CBatchActivePhaseList::SetStepNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepNameHeaderText [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
StepNameWidth Sets the width of the Step Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetStepNameWidth(); **void** CBatchActivePhaseList::SetStepNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StepNameWidth[= *value!*]
StepStateVisible Sets whether or not to display the Step State column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStepStateVisible(); **void** CBatchActivePhaseList::SetStepStateVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StepStateVisible[= *boolvalue*]
StepStateFilter Sets the filter for the Step State column. ***NOTE:*** *This value can be set, but it is ignored. Batch uses the SetStateFilterFlags Method instead.*	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepStateFilter (); **void** CBatchActivePhaseList::SetStepStateFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepStateFilter [= *text$*]
StepStateHeaderText Specifies the header text for the Step State column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepStateHeaderText (); **void** CBatchActivePhaseList::SetStepStateHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepStateHeaderText [= *text$*]
StepStateWidth Sets the width of the Step State column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetStepStateWidth(); **void** CBatchActivePhaseList::SetStepStateWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StepStateWidth[= *value!*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
StepModeVisible Sets whether or not to display the Step Mode column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStepModeVisible(); **void** CBatchActivePhaseList::SetStepModeVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StepModeVisible[= *boolvalue*]
StepModeFilter Sets the filter for the Step Mode column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepModeFilter (); **void** CBatchActivePhaseList::SetStepModeFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepModeFilter [= *text$*]
StepModeHeaderText Specifies the header text for the Step Mode column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepModeHeaderText (); **void** CBatchActivePhaseList::SetStepModeHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepModeHeaderText [= *text$*]
StepModeWidth Sets the width of the Step Mode column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetStepModeWidth(); **void** CBatchActivePhaseList::SetStepModeWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StepModeWidth[= *value!*]
StepIndexVisible Sets whether or not to display the Step Index column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStepIndexVisible(); **void** CBatchActivePhaseList::SetStepIndexVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StepIndexVisible[= *boolvalue*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
StepIndexFilter Sets the filter for the Step Index column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepIndexFilter (); **void** CBatchActivePhaseList::SetStepIndexFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepIndexFilter [= *text$*]
StepIndexHeaderText Specifies the header text for the Step Index column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetStepIndexHeaderText (); **void** CBatchActivePhaseList::SetStepIndexHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*StepIndexHeaderText [= *text$*]
StepIndexWidth Sets the width of the Step Index column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetStepIndexWidth(); **void** CBatchActivePhaseList::SetStepIndexWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*StepIndexWidth[= *value!*]
ElementIDVisible Sets whether or not to display the Element ID column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetElementIDVisible(); **void** CBatchActivePhaseList::SetElementIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ElementIDVisible[= *boolvalue*]
ElementIDFilter Sets the filter for the Element ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetElementIDFilter (); **void** CBatchActivePhaseList::SetElementIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ElementIDFilter [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
ElementIDHeaderText Specifies the header text for the Element ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetElementIDHeaderText (); **void** CBatchActivePhaseList::SetElementIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*ElementIDHeaderText [= *text$*]
ElementIDWidth Sets the width of the Element ID column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetElementIDWidth(); **void** CBatchActivePhaseList::SetElementIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*ElementIDWidth[= *value!*]
OwnerVisible Sets whether or not to display the Owner column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetOwnerVisible(); **void** CBatchActivePhaseList::SetOwnerVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerVisible[= *boolvalue*]
OwnerFilter Sets the filter for the Owner column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerFilter (); **void** CBatchActivePhaseList::SetOwnerFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerFilter [= *text$*]
OwnerHeaderText Specifies the header text for the Owner column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerHeaderText (); **void** CBatchActivePhaseList::SetOwnerHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerHeaderText [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
OwnerWidth Sets the width of the Owner column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetOwnerWidth(); **void** CBatchActivePhaseList::SetOwnerWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerWidth[= *value!*]
OwnerIDVisible Sets whether or not to display the Owner ID column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetOwnerIDVisible(); **void** CBatchActivePhaseList::SetOwnerIDVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerIDVisible[= *boolvalue*]
OwnerIDFilter Sets the filter for the Owner ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerIDFilter (); **void** CBatchActivePhaseList::SetOwnerIDFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerIDFilter [= *text$*]
OwnerIDHeaderText Specifies the header text for the Owner ID column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerIDHeaderText (); **void** CBatchActivePhaseList::SetOwnerIDHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerIDHeaderText [= *text$*]
OwnerIDWidth Sets the width of the Owner ID column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetOwnerIDWidth(); **void** CBatchActivePhaseList::SetOwnerIDWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerIDWidth[= *value!*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
OwnerNameVisible Sets whether or not to display the Owner Name column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetOwnerNameVisible(); **void** CBatchActivePhaseList::SetOwnerNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerNameVisible[= *boolvalue*]
OwnerNameFilter Sets the filter for the Owner Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerNameFilter (); **void** CBatchActivePhaseList::SetOwnerNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerNameFilter [= *text$*]
OwnerNameHeaderText Specifies the header text for the Owner Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetOwnerNameHeaderText (); **void** CBatchActivePhaseList::SetOwnerNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerNameHeaderText [= *text$*]
OwnerNameWidth Sets the width of the Owner Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetOwnerNameWidth(); **void** CBatchActivePhaseList::SetOwnerNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*OwnerNameWidth[= *value!*]
RequestRegisterVisible Sets whether or not to display the Request Register column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetRequestRegisterVisible(); **void** CBatchActivePhaseList::SetRequestRegisterVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RequestRegisterVisible[= *boolvalue*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
RequestRegisterFilter Sets the filter for the Request Register column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetRequestRegisterFilter (); **void** CBatchActivePhaseList::SetRequestRegisterFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RequestRegisterFilter [= *text$*]
RequestRegisterHeaderText Specifies the header text for the Request Register column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetRequestRegisterHeaderText (); **void** CBatchActivePhaseList::SetRequestRegisterHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*RequestRegisterHeaderText [= *text$*]
RequestRegisterWidth Sets the width of the Request Register column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetRequestRegisterWidth(); **void** CBatchActivePhaseList::SetRequestRegisterWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*RequestRegisterWidth[= *value!*]
KeyParameterNameVisible Sets whether or not to display the Key Parameter Name column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetKeyParameterNameVisible(); **void** CBatchActivePhaseList::SetKeyParameterNameVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterNameVisible[= *boolvalue*]
KeyParameterNameFilter Sets the filter for the Key Parameter Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetKeyParameterNameFilter (); **void** CBatchActivePhaseList::SetKeyParameterNameFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterNameFilter [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
KeyParameterNameHeaderText Specifies the header text for the Key Parameter Name column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetKeyParameterNameHeaderText (); **void** CBatchActivePhaseList::SetKeyParameterNameHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterNameHeaderText [= *text$*]
KeyParameterNameWidth Sets the width of the Key Parameter Name column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetKeyParameterNameWidth(); **void** CBatchActivePhaseList::SetKeyParameterNameWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterNameWidth[= *value!*]
KeyParameterValueEUVisible Sets whether or not to display the Key Parameter Engineering Units (EU) column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetKeyParameterValueEUVisible(); **void** CBatchActivePhaseList::SetKeyParameterValueEUVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterValueEUVisible[= *boolvalue*]
KeyParameterValueEUFilter Sets the filter for the Key Parameter Engineering Units (EU) column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetKeyParameterValueEUFilter (); **void** CBatchActivePhaseList::SetKeyParameterValueEUFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterValueEUFilter [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
KeyParameterValueEUHeaderText Specifies the header text for the Key Parameter Engineering Units (EU) column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetKeyParameterValueEUHeaderText (); **void** CBatchActivePhaseList::SetKeyParameterValueEUHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterValueEUHeaderText [= *text$*]
KeyParameterValueEUWidth Sets the width of the Key Parameter Engineering Units (EU) column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetKeyParameterValueEUWidth(); **void** CBatchActivePhaseList::SetKeyParameterValueEUWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*KeyParameterValueEUWidth[= *value!*]
MessageVisible Sets whether or not to display the Message column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetMessageVisible(); **void** CBatchActivePhaseList::SetMessageVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*MessageVisible[= *boolvalue*]
MessageFilter Sets the filter for the Message column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetMessageFilter (); **void** CBatchActivePhaseList::SetMessageFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*MessageFilter [= *text$*]
MessageHeaderText Specifies the header text for the Message column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetMessageHeaderText (); **void** CBatchActivePhaseList::SetMessageHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*MessageHeaderText [= *text$*]

BatchActivePhaseList Column Properties	
Property	**Syntax**
MessageWidth Sets the width of the Message column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetMessageWidth(); **void** CBatchActivePhaseList::SetMessageWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*MessageWidth[= *value!*]
FailureVisible Sets whether or not to display the Failure column.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetFailureVisible(); **void** CBatchActivePhaseList::SetFailureVisible(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureVisible[= *boolvalue*]
FailureFilter Sets the filter for the Failure column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetFailureFilter (); **void** CBatchActivePhaseList::SetFailureFilter (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureFilter [= *text$*]
FailureHeaderText Specifies the header text for the Failure column.	**C++ Syntax:** **CString** CBatchActivePhaseList::GetFailureHeaderText (); **void** CBatchActivePhaseList::SetFailureHeaderText (**LPCTSTR** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureHeaderText [= *text$*]
FailureWidth Sets the width of the Failure column.	**C++ Syntax:** **double** CBatchActivePhaseList::GetFailureWidth(); **void** CBatchActivePhaseList::SetFailureWidth(**double** value); **Visual Basic Syntax:** [*form.*]*Control.*FailureWidth[= *value!*]

BatchActivePhaseList Control Server Properties

The following table lists the VBIS Server properties for the BatchActivePhaseList control.

BatchActivePhaseList Control VBIS Server Properties

Property	Syntax
VBISServerName Sets the name of the remote VBIS Server to which the control should connect. An empty string indicates to use the local VBIS Server.	**C++ Syntax:** **Cstring** CBatchActivePhaseList::GetVBISServerName(); **void** CBatchActivePhaseList::SetVBISServerName(**LPCTSTR** *value*); **Visual Basic Syntax:** [*form.*]*Control.*VBISServerName[= *text$*]
ConnectAtStartup Sets whether or not to connect to the VBIS Server when the control is instantiated.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetConnectAtStartup(); **void** CBatchActivePhaseList::SetConnectAtStartup(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ConnectAtStartup[= *boolvalue*]
RefreshRate Sets the rate, in seconds, that the data in the control is updated. A value of 0 indicates manual refresh only. The default value for the Refresh Rate on the BatchActivePhaseList ActiveX control is 5.	**C++ Syntax:** **short** CBatchActivePhaseList::GetRefreshRate(); **void** CBatchActivePhaseList::SetRefreshRate(**short** *value*); **Visual Basic Syntax:** [*form.*]*Control.*RefreshRate[= *value%*]

BatchActivePhaseList Control Command Buttons Properties

The following table lists the properties that control the display of command buttons in the BatchActivePhaseList control.

BatchActivePhaseList Control Command Buttons Properties	
Property	**Syntax**
SFCButton Sets whether or not to display the SFC button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetSFCButton(); **void** CBatchActivePhaseList::SetSFCButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*SFCButton[= *boolvalue*]

BatchActivePhaseList Control Command Buttons Properties	
Property	**Syntax**
AlarmsButton Sets whether or not to display the Alarm List button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetAlarmsButton(); **void** CBatchActivePhaseList::SetAlarmsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.AlarmsButton[= *boolvalue*]
OperatorPromptsButton Sets whether or not to display the Operator Prompts button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetOperatorPromptsButton(); **void** CBatchActivePhaseList::SetOperatorPromptsButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.OperatorPromptsButton[= *boolvalue*]
BindingPromptsButton Sets whether or not to display the Binding Prompts button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetBindingPromptsButton (); **void** CBatchActivePhaseList::SetBindingPromptsButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.BindingPromptsButton[= *boolvalue*]
StartButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Start Batch button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStartButton(); **void** CBatchActivePhaseList::SetStartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.StartButton[= *boolvalue*]
HoldButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Hold Batch button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetHoldButton(); **void** CBatchActivePhaseList::SetHoldButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control*.HoldButton[= *boolvalue*]

BatchActivePhaseList Control Command Buttons Properties	
Property	**Syntax**
AbortButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Abort Batch button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetAbortButton (); **void** CBatchActivePhaseList::SetAbortButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AbortButton[= *boolvalue*]
AddCommentButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Add Comment button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetAddCommentButton (); **void** CBatchActivePhaseList::SetAddCommentButton (**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AddCommentButton[= *boolvalue*]
ClearAllFailuresButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Clear All Failures button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetClearAllFailuresButton(); **void** CBatchActivePhaseList::SetClearAllFailuresButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ClearAllFailuresButton[= *boolvalue*]
StopButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Stop Batch button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStopButton(); **void** CBatchActivePhaseList::SetStopButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*StopButton[= *boolvalue*]
RestartButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Restart Batch button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetRestartButton(); **void** CBatchActivePhaseList::SetRestartButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*RestartButton[= *boolvalue*]

BatchActivePhaseList Control Command Buttons Properties	
Property	**Syntax**
AutoButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Automatic Mode button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetAutoButton(); **void** CBatchActivePhaseList::SetAutoButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*AutoButton[= *boolvalue*]
ManualButton If you allow access to the SFC from the BatchActivePhaseList control, this command configures whether the SFC control displays the Manual Mode button.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetManualButton(); **void** CBatchActivePhaseList::SetManualButton(**BOOL** value); **Visual Basic Syntax:** [*form.*]*Control.*ManualButton[= *boolvalue*]

BatchActivePhaseList Control Miscellaneous Properties

The following table lists the miscellaneous properties for the BatchActivePhaseList control.

BatchActivePhaseList Control Miscellaneous Properties	
Property	**Syntax**
ToolBarVisible Sets whether or not to display the toolbar.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetToolBarVisible(); **void** CBatchActivePhaseList::SetToolBarVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToolBarVisible[= *boolvalue*]
StatusBarVisible Sets whether or not to display the status bar.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetStatusBarVisible(); **void** CBatchActivePhaseList::SetStatusBarVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*StatusBarVisible[= *boolvalue*]

BatchActivePhaseList Control Miscellaneous Properties	
Property	**Syntax**
ParametersTabVisible Sets whether or not to display the Parameters tab in run mode.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetParametersTabVisible(); **void** CBatchActivePhaseList::SetParametersTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ParametersTabVisible[= *boolvalue*]
VerifyCommandActions Sets whether or not to prompt the operator for confirmation when the operator executes a command.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetVerifyCommandActions(); **void** CBatchActivePhaseList::SetVerifyCommandActions(**BOOL** *value*); **Visual Basic Syntax:** [form.]Control.VerifyCommandActions[= boolvalue]
EnableRightMouseMenu Sets whether or not an operator can view the right-click menu at run-time.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetEnableRightMouseMenu(); **void** CBatchActivePhaseList::SetEnableRightMouseMenu(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*EnableRightMouseMenu[= *boolvalue*]
MouseDblClickedEnabled Sets whether or not to enable operators to double-click a recipe to create a batch.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetMouseDblClickedEnabled(); **void** CBatchActivePhaseList::SetMouseDblClickedEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*MouseDblClickedEnabled[= *boolvalue*]

BatchActivePhaseList Control Miscellaneous Properties	
Property	**Syntax**
InformationTabsVisible Sets whether or not to display the information area, which includes the Parameters and Reports tab, in run mode.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetInformationTabsVisible(); **void** CBatchActivePhaseList::SetInformationTabsVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.InformationTabsVisible[= *boolvalue*]
ReportsTabVisible Sets whether or not to display the Reports tab in run mode.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetReportsTabVisible(); **void** CBatchActivePhaseList::SetReportsTabVisible(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ReportsTabVisible[= *boolvalue*]

BatchActivePhaseList Control Security Properties

The following table lists the security properties for the BatchActivePhaseList control.

BatchActivePhaseList Control Security Properties	
Property	**Syntax**
ColumnEditEnabled Sets whether or not the operator can edit the properties on the Column property page.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetColumnEditEnabled(); **void** CBatchActivePhaseList::SetColumnEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ColumnEditEnabled[= *boolvalue*]
SortOrderEditEnabled Sets whether or not the operator can edit the sort order of the columns on the Sort Order property page.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetSortOrderEditEnabled(); **void** CBatchActivePhaseList::SetSortOrderEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.SortOrderEditEnabled[= *boolvalue*]

BatchActivePhaseList Control Security Properties	
Property	**Syntax**
ServerEditEnabled Sets whether or not the operator can edit the VBIS Server for the control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetServerEditEnabled(); **void** CBatchActivePhaseList::SetServerEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.ServerEditEnabled [= *boolvalue*]
RefreshRateEditEnabled Sets whether or not the operator can edit the refresh rate for the control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetRefreshRateEditEnabled(); **void** CBatchActivePhaseList::SetRefreshRateEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.RefreshRateEditEnabled[= *boolvalue*]
MiscEditEnabled Sets whether or not the operator can edit the properties on the Miscellaneous property page.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetMiscEditEnabled(); **void** CBatchActivePhaseList::SetMiscEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.MiscEditEnabled[= *boolvalue*]
EnableIFIXSecurity Sets whether the ActiveX control uses iFIX security to check if the current user is authorized to execute a command. When set to True, iFIX Security is enabled. When set to False, the default, there is no change to the current behavior. ***IMPORTANT:*** *Electronic signatures configured in Batch Execution override the Enable iFIX Security setting.*	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetEnableIFIXSecurity(); **void** CBatchActivePhaseList::SetEnableIFIXSecurity(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EnableIFIXSecurity[= *boolvalue*]

BatchActivePhaseList Control Security Properties	
Property	**Syntax**
ToggleConnectionEnabled Sets whether or not the operator can toggle the VBIS Server connection.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetToggleConnectionEnabled(); **void** CBatchActivePhaseList::SetToggleConnectionEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ToggleConnectionEnabled[= *boolvalue*]
ParamReportEditEnabled Sets whether or not the operator can edit SFC parameters, if the user is allowed to access the SFC from the BatchActivePhaseList control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetParamReportEditEnabled(); **void** CBatchActivePhaseList::SetParamReportEditEnabled(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ParamReportEditEnabled[= *boolvalue*]
ViewSFC Sets whether the operator is allowed to view the SFC screen from the BatchActivePhaseList control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetViewSFC(); **void** CBatchActivePhaseList::SetViewSFC(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewSFC[= *boolvalue*]
ViewAlarms Sets whether the operator is allowed to view an alarm list screen from the BatchActivePhaseList control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetViewAlarms(); **void** CBatchActivePhaseList::SetViewAlarms(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewAlarms[= *boolvalue*]
ViewOperatorPrompts Sets whether the operator is allowed to view an operator prompts screen from the BatchActivePhaseList control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetViewOperatorPrompts(); **void** CBatchActivePhaseList::SetViewOperatorPrompts(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewOperatorPrompts[= *boolvalue*]

BatchActivePhaseList Control Security Properties	
Property	**Syntax**
ViewBindingPrompts Sets whether the operator is allowed to view a screen of binding prompts from the BatchActivePhaseList control.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetViewBindingPrompts(); **void** CBatchActivePhaseList::SetViewBindingPrompts(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*ViewBindingPrompts[= *boolvalue*]

.

BatchActivePhaseList Control Electronic Signature Properties

The following table lists the properties that control the electronic signature settings for the BatchActivePhaseList control.

BatchActivePhaseList Control Electronic Signature Properties	
Property	**Syntax**
UseDefaultSignatureRequirements Sets a default signature type (None, Performed By, Performed By/Verified By) to all of the commands for this ActiveX control. If you do not use the UseDefaultSignatureRequirements property, you must specify the signature type for each command individually. Otherwise, the signature type is set to NONE.	**C++ Syntax:** **BOOL** CBatchActivePhaseList::GetUseDefaultSignatureRequirements();**void** CBatchActivePhaseList::SetUseDefaultSignatureRequirements(**BOOL** *value*); **Visual Basic Syntax:** [*form.*]*Control.*UseDefaultSignatureRequirements[= *boolvalue*]

BatchActivePhaseList Control Color Properties

The following table lists the color properties for the BatchActivePhaseList control.

BatchActivePhaseList Control Color Properties	
Property	**Syntax**
BackColor Sets the background color (the border around the edge) of the control.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetBackColor(); **void** CBatchActivePhaseList::SetBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.BackColor[= *color*]
EvenRowBackColor Sets the background color of the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetEvenRowBackColor(); **void** CBatchActivePhaseList::SetEvenRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowBackColor[= *color%*]
EvenRowTextColor Sets the color of the text in the even rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetEvenRowTextColor(); **void** CBatchActivePhaseList::SetEvenRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.EvenRowTextColor[= *color%*]
GridColor Sets the color of the grid lines in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetGridColor(); **void** CBatchActivePhaseList::SetGridColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.GridColor[= *color%*]
HeaderBackColor Sets the background color for the column headers.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetHeaderBackColor(); **void** CBatchActivePhaseList::SetHeaderBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderBackColor[= *color*]

BatchActivePhaseList Control Color Properties	
Property	**Syntax**
HeaderTextColor Sets the color for the header text.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetHeaderTextColor(); **void** CBatchActivePhaseList::SetHeaderTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderTextColor[= *color%*]
OddRowBackColor Sets the background color for the odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetOddRowBackColor(); **void** CBatchActivePhaseList::SetOddRowBackColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowBackColor[= *color%*]
OddRowTextColor Sets the text color for odd rows in the data list.	**C++ Syntax:** **OLE_COLOR** CBatchActivePhaseList::GetOddRowTextColor(); **void** CBatchActivePhaseList::SetOddRowTextColor(**OLE_COLOR** *value*); **Visual Basic Syntax:** [*form.*]*Control*.OddRowTextColor[= *color%*]

NOTE: *For examples on setting color properties, refer to the BatchList ActiveX Control section.*

BatchActivePhaseList Control Font Properties

The following table lists the properties that control the fonts in the BatchActivePhaseList control.

BatchActivePhaseList Control Font Properties	
Property	**Syntax**
HeaderFont Sets the font of the text in the headers.	**C++ Syntax:** **COleFont** CBatchActivePhaseList::GetHeaderFont(); **void** CBatchActivePhaseList::SetHeaderFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control*.HeaderFont[= *stdfontvariable*]

BatchActivePhaseList Control Font Properties	
Property	**Syntax**
TextFont Sets the font of the text in the data list.	**C++ Syntax:** **COleFont** CBatchActivePhaseList::GetTextFont(); **void** CBatchActivePhaseList::SetTextFont(**LPDISPATCH** *value*); **Visual Basic Syntax:** [*form.*]*Control.*TextFont[= *stdfontvariable*]

BatchActivePhaseList Control Methods

The BatchActivePhaseList control supports the following methods:

- ConnectToServer Method
- DisconnectFromServer Method
- SetSortKeys Method
- SetStateFilterFlags Method
- Refresh Method
- GetCommandSignatureRequirements Method
- SetCommandSignatureRequirements Method

The SetStateFilterFlags method is described in the following section. The remaining events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

SetStateFilterFlags Method

Description

Describes what active states display in the BatchActivePhaseList screen.

C++ Syntax

```
BOOL CBatchActivePhaseList::SetStateFilterFlags (BOOL RunState, BOOL HoldState, BOOL
StopState, BOOL AbortState);
```

Visual Basic Syntax

```
[form.]Control.SetStateFilterFlags (RunState As Boolean, HoldState As Boolean, StopState
As Boolean, AbortState As Boolean) As Boolean
```

Parameters

Parameter	Description
RunState	If set to true, any batch in the Run or Restarting state is shown in the BatchActivePhaseList control list.
HoldState	If set to true, any batch in the Holding or Held state is shown in the BatchActivePhaseList control list.
StopState	If set to true, any batch in the Stopping or Stopped state is shown in the BatchActivePhaseList control list.
AbortState	If set to true, any batch in the Aborting or Aborted state is shown in the BatchActivePhaseList control list.

Return Type

Boolean.

- 0 = unsuccessful
- 1 = success

C++ Example

```
BOOL result = pBatchActivePhaseList->SetStateFilterFlags (TRUE, TRUE, TRUE, TRUE)
// displays all states in the ActivePhaseList screen
```

Visual Basic Example

```
Dim bValue As Boolean
bValue = BatchActivePhaseList1.SetStateFilterFlags (TRUE, TRUE, TRUE, TRUE)
' displays all states in the ActivePhaseList screen
```

BatchActivePhaseList Control Events

The BatchActivePhaseList control generates the following events:

- ConnectedToServer Event
- DisconnectedFromServer Event
- ServerChanged Event

These events are the same for each control. Refer to the BatchList ActiveX Control section for a description of these common events.

Understanding Windows Security and the Batch Execution ActiveX Controls

Running software in a distributed environment adds complexity to system configuration and maintenance. This is because a distributed application requires extra attention to security context details.

The most frequent issue you may encounter when deploying Batch Execution in a distributed environment is with the Batch Execution ActiveX control on a client computer not connecting to the VBIS Server on a remote computer. While you may assume that the problem resides on the server, it is important to understand that the problem could exist on the client, server, or both computers.

The sections that follow describe an ideal configuration to avoid this connection issue. It also describes some troubleshooting tips if you have deployed your system.

The Ideal Configuration

Avoiding connection problems in the first place is preferable to troubleshooting them. The following suggestions list ideal settings for the domain controller, the server, and the client.

The Domain Controller

Create Windows security groups on your domain controller that model the various roles in your organization. For example, you might capture the responsibilities of the plant floor operators in a group called Operators and their immediate managers may be organized into a group called Supervisors. As people come and go, or as people are reassigned throughout the organization, modify the members of the Operators and Supervisors groups to keep them current.

The Server

Make sure that you do the following on the server computer:

- Confirm that you enabled the Distributed Component Object Model (DCOM) protocol. For steps, refer to the Enabling Distributed COM section in the System Configuration Manual.
- Confirm that you set the authentication level to Connect for the Batch Integration Services application. For steps on how to check the authentication level, refer to the Verifying the DCOM Authentication Level on the Server section in the System Configuration Manual (in these steps, replace "GE Intelligent Platforms EIB Server" with "GE Intelligent Platforms Batch Integration Services").
- Confirm that you configured custom access and launch permissions for the Batch Integration Services application. For steps on how to check this, refer to the Verifying Access and Launch Permissions section in the System Configuration Manual (in these steps, replace "GE Intelligent Platforms EIB Server" with "GE Intelligent Platforms Batch Integration Services"). Do not remove any already listed groups.

The Client

Make sure that you do the following on the client computer:

- Confirm that you enabled the Distributed Component Object Model (DCOM) protocol. For steps, see the Enabling Distributed COM section in the System Configuration Manual.
- Confirm that you set the authentication level of Batch Execution ActiveX control container to Connect. For steps on how to check the authentication level, refer to the Verifying the DCOM Authentication Level on the Client section in the System Configuration Manual.

Troubleshooting Tips

This section lists some practical measures you can take when troubleshooting connection issues between the ActiveX controls and the VBIS Server.

To troubleshoot connection issues:

1. Eliminate networking and hardware issues by pinging the server from a DOS prompt. If the ping fails, you must resolve that issue before proceeding.
2. Enable Windows auditing for logon and logoff events. Refer to the online Help for Windows for instructions on how to do this. With this option enabled, you can use the Microsoft® Event Viewer to examine the reasons for any failures.
3. Use the Microsoft Event Viewer to examine the security log. Typically, you'll observe something like this on the server computer:

   ```
   Logon Failure:

   Reason: Unknown user name or bad password

   User Name: Operator1

   Domain: PLANTFLOORPC

   Logon Type: 3

   Logon Process: NtLmSsp

   Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0

   Workstation Name: LINE1PC
   ```

   This error occurs when the account which you're logged in to on the client is unknown to the server and its trusted domain controllers. One way to resolve this issue is by setting up your environment to allow you to log in to the client and the server using the same account.

   NOTE: *If you want to use local accounts on the client and server computers (as opposed to a single domain account), the user names and passwords must be an exact match between computers.*

4. Verify that you enabled the DCOM protocol on both client and server computers. For steps, see the Enabling Distributed COM section in the System Configuration Manual.
5. Verify that the password for the account you created during the install is correct. If you did not specify an account, the default user account, BatchExecutive, with a password of

batchrules is used. In Batch Execution 4.6 the password on the BatchExecutive account changed from "batch" to "batchrules." Be sure to check the password in the User Manager, as well as on the GE Intelligent Platforms Batch Integration Services in the Windows services applet. Keep in mind that passwords are case-sensitive.

6. On the server computer, verify that the Batch Execution Integration Services access and launch permissions list the Everyone and Domain\Everyone (where domain is the name of your domain) Windows groups.

   IMPORTANT: *After you perform this step, anyone who can view the computer on the network can also launch VBIS.*

7. Verify that the client and server have the same default DCOM authentication level. One way to troubleshoot the authentication level is to set the default DCOM authentication level to None on both the client and server.

   IMPORTANT: *After you perform this step, you essentially turn off DCOM security. That means, anyone who can access the computer on the network can also write code to access and launch applications on the computer.*

Appendix A: VBIS Recipe Data Model

The VBIS recipe data model defines the structures and rules that represent the storage of recipes in a relational database. The data model:

- Is not specific to a particular vendor's implementation.
- Represents the tables, table attributes, constraints, and relationships among the tables.

Creating the Data Model

You can create the tables shown in the Entity Relationship Diagram section by running the setup script for your relational database. For instructions on running a setup script, refer to the Batch Execution System Configuration Manual.

Upgrading

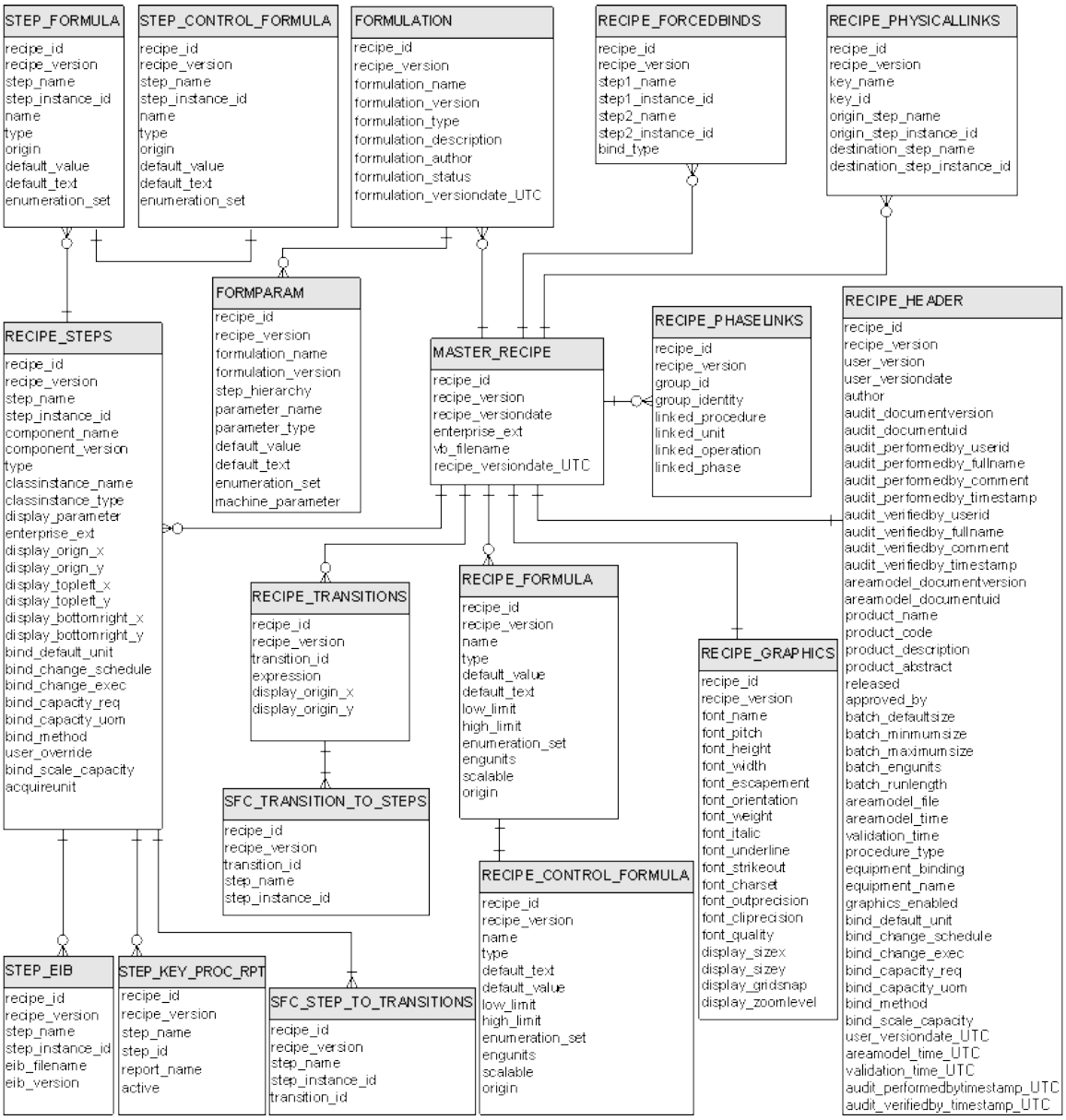
Customers who are upgrading from previous versions of Batch Execution need to run an upgrade script to upgrade their existing tables. Refer to the Upgrade Guide for instructions on running this upgrade script.

Entity Relationship Diagram

After you run the setup script for your relational database, the data model, shown in the following figure, is created. This data model includes the relational database tables you need to create master and control recipes.

NOTE: *The Batch Execution user interfaces were updated to use the term parameter instead of*

formula in version 4.5, however the table names did not change to reflect this naming convention. For example, STEP_FORMULA refers to the step parameters and RECIPE_FORMULA refers to the recipe parameters.



VBIS Recipe Data Model

Creating a Recipe

To add a recipe to the tables in the VBIS data model, you can:

- Use the Recipe Editor to create a recipe and save it to the relational database. Refer to the Recipe Development Manual for more information on creating a recipe with the Recipe Editor.

- Populate the tables with a third-party tool.

Attribute Domain

An *attribute* is a fact or non-decomposable piece of information describing the content of a table. Each table in the VBIS data model is comprised of one or more attributes required by Batch Execution master recipes. An *attribute domain* defines a set of valid values for an attribute. Domain characteristics define both generic characteristics and entity related characteristics. The generic characteristics describe the attributes' data type, length, allowable values, and meaning. The entity (table) characteristics define the attribute's uniqueness, null support, default value, and additional constraints.

The following are the characteristics associated with each attribute in the model:

Attribute Domain	**Value/Description**
Data Type	Integer, Real, String, Text, Date.
Length	The maximum number of characters or digits supported by the data type.
Allowable Values	The range of values supported.
Uniqueness	Unique or non-unique. By default, attributes are non-unique.
Null Support	Allowed or not allowed.
Key	Indicates whether the attribute is part of a primary, alternate, or foreign key.
Default Value	The attribute value if no value is supplied.
Description	Text describing the function of the attribute.
Constraints	Rules that enforce data integrity between and within tables.

The following sections describe each table in the VBIS data model.

VBIS_DCT Table

Some allowable values in the VBIS Data Model are represented by a single digit. For example, in the RECIPE_FORMULA table, the scalable attribute has allowable values of 0 (not scalable) or 1 (scalable). While the data model represents these values internally with a numeric code, you may want to replace each number with text when you generate a report.

To replace the numeric values with text, issue an SQL JOIN command to the table VBIS_DCT. This table defines the text associated with the numeric values for the allowable values domain. The following table describes the attributes for the VBIS_DCT table.

VBIS_DCT Attributes	
Attribute	**Attribute Domain**
NAME	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: Internal usage. Do not modify. Description: Language-dependent string name.
ID	Data Type: Integer. Length: 1 Null Support: Nulls not allowed. Allowable Values: Internal usage. Do not modify. Description: Numerical ID associated with Name attribute.
TYPE	Data Type: Character. Length: 1 Null Support: Nulls not allowed. Description: Classifies the name into a logical grouping as the following tables show.

VBIS_DCT Recipe Values		
Name	**ID**	**Type**
NONE	0	R (recipe)
CLASS	1	R
INSTANCE	2	R

VBIS_DCT Formula Values		
Name	**ID**	**Type**
REAL	1	F (step parameter or recipe parameter)

LONG	2	F
STRING	3	F
ENUM	4	F

VBIS_DCT Formula Origin Values

Name	ID	Type
VALUE	0	O (parameter origin)
DEFER	1	O
OPERATOR	2	O

VBIS_DCT Procedure Type Values

Name	ID	Type
PROCEDURE	1	P (procedure type)
UNIT	2	P
OPERATION	3	P

VBIS_DCT Boolean Values

Name	ID	Type
YES	1	B (Boolean)
NO	0	B

VBIS_DCT Step Values

Name	ID	Type
INITIAL	0	S (step)
REGULAR	1	S
TERMINAL	2	S

Master Recipe Tables

The VBIS data model contains the following master recipe tables:

- MASTER_RECIPE
- RECIPE_GRAPHICS
- RECIPE_HEADER
- RECIPE_FORMULA
- RECIPE_PHASELINKS
- RECIPE_STEPS
- STEP_EIB
- RECIPE_TRANSITIONS
- STEP_FORMULA
- SFC_STEP_TO_TRANSITIONS
- SFC_TRANSITION_TO_STEPS
- RECIPE_FORCEDBINDS
- RECIPE_PHYSICALLINKS
- STEP_KEY_PROC_RPT Table
- FORMPARAM Table
- FORMULATION Table

For detailed information on the master recipe tables, refer to the following subsections. For information on the control recipe tables in the VBIS data model, refer to the Control Recipe Tables section.

MASTER_RECIPE Table

The MASTER_RECIPE table stores the basic master recipe information required by Batch Execution.

MASTER_RECIPE Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Uniquely identifies a recipe.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
RECIPE_VERSIONDATE	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: The current date and time. Description: Batch Execution internal date and time. Do not modify.

MASTER_RECIPE Table	
Attribute	**Attribute Domain**
ENTERPRISE_EXT	Data Type: String. Length: 30 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Indicates that the recipe is integrated into a higher level business system. User defined.
VB_FILENAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any valid file name. Default Value: None. Description: The recipe file name to use during batch production. The name does not include the full path, just the recipe file name.
RECIPE_VERSIONDATE_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Time stamp date in UTC format. Default Value: The current date and time, in UTC format. Description: Batch Execution internal date and time. Do not modify.

RECIPE_GRAPHICS Table

The RECIPE_GRAPHICS table stores Batch Execution Recipe Editor graphical information.

RECIPE_GRAPHICS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal version number. Do not modify.
FONT_NAME	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: None. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_PITCH	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_GRAPHICS Table	
Attribute	**Attribute Domain**
FONT_HEIGHT	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_WIDTH	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_ESCAPEMENT	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_ORIENTATION	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_GRAPHICS Table	
Attribute	**Attribute Domain**
FONT_WEIGHT	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_ITALIC	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_UNDERLINE	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_STRIKEOUT	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_GRAPHICS Table	
Attribute	**Attribute Domain**
FONT_CHARSET	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_OUTPRECISION	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_CLIPPRECISION	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
FONT_QUALITY	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_GRAPHICS Table	
Attribute	**Attribute Domain**
DISPLAY_SIZEX	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_SIZEY	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_GRIDSNAP	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_ZOOMLEVEL	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_HEADER Table

The RECIPE_HEADER table stores administrative information for each recipe.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal version number. Do not modify. The recipe ID and the user version number uniquely identify a recipe.
USER_VERSION	Data Type: String. Length: 4 characters. Key: Alternate key (AK). Null Support: Nulls not allowed. Allowable Values: 0-9999. Default Value: 1 Description: User defined recipe version number.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
USER_VERSIONDATE	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: The current date and time. Description: The date and time when the recipe was last modified.
AUTHOR	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: 1 Description: The creator of the recipe.
AUDIT_DOCUMENTVERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Description: The audit version number of the recipe file. The audit version number increases by one each time the recipe is saved in the Batch Execution Recipe Editor. However, for recipe files which include sub-recipes, the audit version number of the sub-recipe file does not increment unless there is a change in the sub-recipe itself or in the area model.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
AUDIT_DOCUMENTUID	Data Type: String. Length: 32 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: A unique, system-generated identifier for the file. Batch Execution generates the globally unique identifier (also known as a GUID) when you save a file for the first time, or use the Save As action to resave a file in the Batch Execution Recipe Editor.
AUDIT_PERFORMEDBY_USERID	Data Type: String. Length: 20 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Windows user name for the person who performed the action. This field is blank if a Performed By signature is not required for this action.
AUDIT_PERFORMEDBY_FULLNAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Full name of the person who performed the action. Batch Execution does not accept a signature without a full user name defined in Windows security. This field is blank if a Performed By signature is not required for this action.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
AUDIT_PERFORMEDBY_COMMENT	Data Type: String. Length: 2048 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Comment entered by the supervisor who entered the Performed By electronic signature. This field is blank if the user did not enter a comment or if a Performed By signature is not required for this action.
AUDIT_PERFORMEDBY_TIMESTAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: The current date and time. Description: The date and time that the user entered the Performed By signature. This field is blank if a Performed By signature is not required for this action.
AUDIT_VERIFIEDBY_USERID	Data Type: String. Length: 20 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Windows user name for the person who verified the action. This field is blank if a Verified By signature is not required for this action.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
AUDIT_VERIFIEDBY_FULLNAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Full name of the person who verified the action. Batch Execution does not accept a signature without a full user name defined in Windows security. This field is blank if a Verified By signature is not required for this action.
AUDIT_VERIFIEDBY_COMMENT	Data Type: String. Length: 2048 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: Comment entered by the supervisor who entered the Verified By electronic signature. This field is blank if the user did not enter a comment or if a Verified By signature is not required for this action.
AUDIT_VERIFIEDBY_TIMESTAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: The current date and time. Description: The date and time that the user entered the Verified By signature. This field is blank if a Verified By signature is not required for this action.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
AREAMODEL_DOCUMENTVERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Description: The audit version number of the area model file.
AREAMODEL_DOCUMENTUID	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: The unique, system-generated identifier for the area model. Batch Execution generates the GUID when you save a new area model for the first time, or use the Save As command to resave an area model.
PRODUCT_NAME	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: The commercial name of the product.
PRODUCT_CODE	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: The product code of the material produced by this recipe.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
PRODUCT_DESCRIPTION	Data Type: String. Length: 132 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: Text describing what this recipe makes.
PRODUCT_ABSTRACT	Data Type: Text. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: Abstract of the recipe describing what it does and how it does it.
RELEASED	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (released), 0 (not released). Default Value: 0 Description: Set to 1 if the recipe has been released to production. Once released, Batch Execution adds the recipe to the batch list.
APPROVED_BY	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: Name of the individual who approved the recipe for release to production.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
BATCH_DEFAULTSIZE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: 0-9 and a decimal point. Default Value: 0 Description: The default amount manufactured in one batch by this recipe.
BATCH_MINIMUMSIZE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: 0-9 and a decimal point. Default Value: 0 Description: The minimum amount that can be manufactured with the recipe.
BATCH_MAXIMUMSIZE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: 0-9 and a decimal point. Default Value: 100 Description: The maximum amount that can be manufactured with the recipe.
BATCH_ENGUNITS	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: Engineering units for the batch size, minimum, and maximum values.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
BATCH_RUNLENGTH	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: Estimated duration of the batch.
AREAMODEL_FILE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any valid path. Default Value: The path of the equipment database. Description: The path of the area model file used to build the recipe. This must be the same as the area model in the VBEXEC.INI file for the recipe to execute on the Batch Execution Server.
AREAMODEL_TIME	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: None. Description: The date and time that the area model was last modified.
VALIDATION_TIME	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Any valid date. Default Value: None. Description: The date and time that the recipe was validated against its area model.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
PROCEDURE_TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (procedure), 2 (unit procedure), 3 (operation). Default Value: None. Description: The type of procedural element (ISA-S88.01).
EQUIPMENT_BINDING	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (none), 1 (class-based), 2 (instance-based). Default Value: 0 Description: Indicates whether the recipe is class- or instance-based. Valid only for unit procedures and operations. For procedures, this attribute is always 0.
EQUIPMENT_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Name of the unit class or unit instance upon which the procedure has been built.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
GRAPHICS_ENABLED	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (graphics enabled), 0 (graphics disabled). Default Value: 0 Description: Indicates whether the recipe has graphical information, such as a font or coordinates of steps and transitions. This information is stored with the recipe and is used by the Recipe Editor and the Batch Execution Client application.
BIND_METHOD	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (None), 1 (Batch Creation), 2 (Automatic), 3 (Operator Prompt). Default Value: 0 Description: Bind type for the unit procedure. Batch procedures and unit operations are always 0 (None).
BIND_DEFAULT_UNIT	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Description: Default unit to bind to during batch execution. Assigned by the recipe developer. The operator may be able to change this at batch creation time.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
BIND_CHANGE_SCHEDULE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (allow operators to modify bind type at batch creation), 0 (prevent operators from modifying binding at batch creation). Default Value: 1 Description: Controls operator interaction during batch creation.
BIND_CHANGE_EXEC	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (allow operators to modify the bind type during batch execution, that is during a rebind), 0 (prevent operators from rebinding a unit procedure to a unit). Default Value: 1 Description: Controls operator interaction during batch execution.
BIND_CAPACITY_REQ	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: 0-9, decimal point. Default Value: NULL. Description: Minimum required unit capacity on which this procedure may execute. Checked against the unit capacity defined for the unit in the area model.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
BIND_CAPACITY_UOM	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Alphanumeric. Default Value: None. Description: Unit of measure for the required unit capacity amount.
BIND_SCALE_CAPACITY	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (do not scale), 1 (scale according to the batch scale). Default Value: 1 Description: If set, the capacity requirement is scaled according to the batch scaling factor that is specified when the batch is created.
USER_VERSIONDATE_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: The current date and time, in UTC format. Description: The date and time, in UTC format, when the recipe was last modified.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
AUDIT_PERFORMEDBY_TIMESTAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: The current date and time, in UTC format. Description: The date and time, in UTC format, that the user entered the Performed By signature. This field is blank if a Performed By signature is not required for this action.
AUDIT_VERIFIEDBY_TIMESTAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: The current date and time, in UTC format. Description: The date and time, in UTC format, that the user entered the Verified By signature. This field is blank if a Verified By signature is not required for this action.
AREAMODEL_TIME_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: None. Description: The date and time, in UTC format, that the area model was last modified.

RECIPE_HEADER Table	
Attribute	**Attribute Domain**
VALIDATION_TIME_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: None. Description: The date and time, in UTC format, that the recipe was validated against its area model.

RECIPE_FORMULA Table

The RECIPE_FORMULA table stores master recipe parameter process inputs and parameters.

RECIPE_FORMULA Table	
Attribute	**Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Master recipe version number. Do not modify.

RECIPE_FORMULA Table	
Attribute	**Domain**
NAME	Data Type: String. Length: 60 characters. Key: Primary key (PK3). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the recipe parameter.
TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (real), 2 (long), 3 (string), 4 (enumeration). Default Value: 1 Description: The type of data that the recipe represents.
DEFAULT_TEXT	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: Depends on the type of recipe parameter. Default Value: Depends on the type of recipe parameter. Description: The default value for the parameter. Valid for strings and enumerations only.
DEFAULT_VALUE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls allowed. Allowable Values: Depends on the type of recipe parameter. Default Value: Depends on the type of recipe parameter. Description: The default value for the parameter. Valid for reals and longs only.

RECIPE_FORMULA Table	
Attribute	**Domain**
LOW_LIMIT	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: Depends on the type of recipe parameter. Default Value: Depends on the type of recipe parameter. Description: The default value for the parameter. Valid for reals and longs only.
HIGH_LIMIT	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: Depends on the type of recipe parameter. Default Value: Depends on the type of parameter. Description: The default value for the parameter. Valid for reals and longs only.
ENUMERATION_SET	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the enumeration that contains the recipe parameter value. Only valid when the type is an enumeration.
ENGUNITS	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The engineering units associated with the recipe parameter.

RECIPE_FORMULA Table	
Attribute	**Domain**
SCALABLE	Data Type: Integer. Length: 1 digit. Null Support: Nulls allowed. Allowable Values: 1 (scalable), 0 (not scalable). Default Value: 0 Description: Indicates if the parameter is to be scaled according to the scale of the batch.
ORIGIN	Data Type: Integer. Length: 99 digits. Null Support: Nulls not allowed. Allowable Values: 0 (value), 1 (defer), 2 (operator; operations only), 3 (defer – legacy), 4 (defer to unit procedure), 5 (defer to procedure). Default Value: 99 Description: Defines the level of the recipe that supplies the value for the step.

RECIPE_PHASELINKS Table

The RECIPE_PHASELINKS table stores the synchronized phases among procedural elements.

RECIPE_PHASELINKS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.

RECIPE_PHASELINKS Table	
Attribute	**Attribute Domain**
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
GROUP_ID	Data Type: Integer. Length: 2 characters. Key: Primary key (PK3). Null Support: Null not allowed. Allowable Values: 1-50. Default Value: None. Description: Phase link group ID.
GROUP_IDENTITY	Data Type: Integer. Length: Auto-number. Key: Primary key (PK4). Null Support: Null not allowed. Default Value: None. Description: Auto-number for phase link group identity.
LINKED_PROCEDURE	Data Type: String. Length: 255 characters. Null Support: Null allowed; dependent upon recipe type. Allowable Values: a-z, A-Z, 0-9, the underscore character, and the colon character. Default Value: None. Description: ID of procedure recipe containing a linked phase.

RECIPE_PHASELINKS Table	
Attribute	**Attribute Domain**
LINKED_UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, the underscore character, and the colon character. Default Value: None. Description: Unit procedure step ID for the current procedure.
LINKED_OPERATION	Data Type: String. Length: 255. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, the underscore character, and the colon character. Default Value: None. Description: Operation step ID for the current unit procedure.
LINKED_PHASE	Data Type: String. Length: 55. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, the underscore character, and the colon character. Default Value: None. Description: Phase ID for the current operation.

RECIPE_STEPS Table

The RECIPE_STEPS table represents the sequential function chart steps for each recipe.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK3). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: None. Description: Recipe Editor step name. For operations, this name is the phase name within the equipment module. For procedures and unit procedures, this is the name of an existing unit procedure or operation. Constraint: If the step is within a procedure or unit procedure, a recipe in the MASTER_RECIPE table must exist with the corresponding step_name for the current procedure level.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK4). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.
TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (initial step), 1 (step), 2 (terminal step). Default Value: None. Description: The type of sequential function chart step.
COMPONENT_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: None. Description: Name of a file or SQL-based recipe that this step represents. In the case of SQL-based recipes, it is the same as the step name. This attribute is used by the Recipe Editor. Do not modify.
COMPONENT_VERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
CLASSINSTANCE_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, the underscore character, and "CLASS". Default Value: None. Description: For procedures, the name of the unit class or instance the step requires. For unit procedures, this name is defined as "CLASS". For operations, this name is a null value.
CLASSINSTANCE_TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (class-based), 2 (instance-based). Default Value: 1 Description: Indicates the type of recipe the current step represents when the step is a procedure or unit procedure. If the step is an operation, then it defaults to 1.
DISPLAY_PARAMETER	Data Type: String. Length: 60 characters. Key: Parameter name in the STEP_FORMULA table (foreign key). Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: The step parameter name displayed in the sequential function chart for the current step. Only one parameter name is displayed.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
ENTERPRISE_EXT	Data Type: String. Length: 30 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: None. Description: Indicates that the current step is integrated into a higher level business system. User defined.
DISPLAY_ORIGIN_X	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_ORIGIN_Y	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_TOPLEFT_X	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
DISPLAY_TOPLEFT_Y	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_BOTTOMRIGHT_X	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_BOTTOMRIGHT_Y	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
USER_OVERRIDE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (not applicable), 1 (override unit procedure's equipment requirements). Default Value: 0 Description: For batch procedure steps, you can override the equipment requirements of the unit procedure at the batch procedure level.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
BIND_METHOD	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (None), 1 (Batch Creation), 2 (Automatic), 3 (Operator Prompt). Default Value: 0 Description: The bind type for the unit procedure. Batch procedures and unit operations will always be 0 (None).
BIND_DEFAULT_UNIT	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Description: Default unit to bind during batch execution. Assigned by the recipe developer. The operator may be able to change this at batch creation time.
BIND_CHANGE_SCHEDULE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (allow modification of binding type at batch creation), 0 (prevent operator from modifying binding at batch creation). Default Value: 1 Description: Controls operator interaction when a batch is created.
BIND_CHANGE_EXEC	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (allow operators to modify the bind type during batch execution, that is, during a rebind), 0 (prevent operators from rebinding unit procedure). Default Value: 1 Description: Controls operator during batch execution.

RECIPE_STEPS Table	
Attribute	**Attribute Domain**
BIND_CAPACITY_REQ	Data Type: Real. Length: 7 digits, including decimal point. Null Support: Nulls not allowed. Allowable Values: 0-9 and decimal point. Default Value: NULL. Description: The minimum required unit capacity on which this procedure can execute. Checked against the unit capacity defined for a unit in the area model.
BIND_CAPACITY_UOM	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Alphanumeric. Default Value: None. Description: Unit of measure for the required unit capacity amount.
BIND_SCALE_CAPACITY	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (do not scale), 1 (scale according to batch scaling factor). Default Value: 1 Description: If set, the capacity requirement is scaled according to the batch scaling factor that is specified when the batch is created.
ACQUIREUNIT	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 0 (do not acquire), 1 (acquire unit). Default Value: 1 Description: If set, acquire the unit.

STEP_EIB Table

The STEP_EIB table represents the sequential function chart steps for each EIB.

STEP_EIB Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: None. Description: Recipe Editor step name. For operations, this name is the phase name within the equipment module. For procedures and unit procedures, this is the name of an existing unit procedure or operation. Constraint: If the step is within a procedure or unit procedure, a recipe in the MASTER_RECIPE table must exist with the corresponding step_name for the current procedure level.

STEP_EIB Table	
Attribute	**Attribute Domain**
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.
EIB_FILENAME	Data Type: String. Length: 65 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and the underscore character. Default Value: None. Description: EIB file name.
EIB_VERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls allowed. Allowable Values: Any number, 0 or greater. 0 means use the most current version of the EIB. The greatest allowable value cannot be greater than the highest version for that EIB in the production store. You can view the latest version in the WorkInstruction Document Management dialog box in the WorkInstruction Editor. Default Value: 0 Description: Batch Execution internal EIB version.

RECIPE_TRANSITIONS Table

The RECIPE_TRANSITIONS table represents the sequential function chart transitions for each recipe.

RECIPE_TRANSITIONS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
TRANSITION_ID	Data Type: Integer. Length: 6 digits. Key: Primary Key (PK3). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: 0-9 Default Value: None. Description: Unique identifier used to distinguish between each transition within a recipe.

RECIPE_TRANSITIONS Table	
Attribute	**Attribute Domain**
EXPRESSION	Data Type: String. Length: 1024 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: TRUE. Description: Transition expression.
DISPLAY_ORIGIN_X	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.
DISPLAY_ORIGIN_Y	Data Type: Integer. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Read only. Default Value: None. Description: Internally set by the Recipe Editor.

STEP_FORMULA Table

The STEP_FORMULA table represents process inputs and parameters of the master recipe step.

STEP_FORMULA Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1), Foreign key (FK1) to RECIPE_STEPS. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2), Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK3), Foreign key (FK3). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Recipe Editor step name. For operations, this is the name of a phase within the equipment module. For procedures and unit procedures, this is the name of an existing unit procedure or operation.

STEP_FORMULA Table	
Attribute	**Attribute Domain**
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK4), Foreign key (FK4). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.
NAME	Data Type: String. Length: 60 characters. Key: Primary key (PK5). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the step parameter.
TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (real), 2 (long), 3 (string), 4 (enumeration). Default Value: 1 Description: The type of data the step parameter represents.
ORIGIN	Data Type: Integer. Length: 99 digits. Null Support: Nulls not allowed. Allowable Values: 0 (value), 1 (defer), 2 (operator; operations only), 3 (defer – legacy), 4 (defer to unit procedure), 5 (defer to procedure). Default Value: 99 Description: Defines the level of the recipe that supplies the value for the step.

STEP_FORMULA Table	
Attribute	**Attribute Domain**
DEFAULT_VALUE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls allowed. Allowable Values: Depends on the type of step parameter. Default Value: Low Limit for reals and longs. Description: The default value for the step parameter. Constraints: The value must be within the limits of one of the following: The defined equipment parameters for operation steps. The limits defined in the procedure or unit procedure parameter that this step represents.
DEFAULT_TEXT	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: Depends on the type of step parameter. Default Value: Depends on the type of step parameter. Description: The default text for the step parameter. Valid when the step type is string or enumeration. Also valid when the origin is deferred. ***NOTE:*** *The value for deferred steps is equal to the name of a recipe parameter from the identified recipe.*
ENUMERATION_SET	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the enumeration that contains the step parameter value. Only valid when the type is an enumeration.

SFC_STEP_TO_TRANSITIONS Table

The SFC_STEP_TO_TRANSITIONS table represents the links from steps to transitions.

SFC_STEP_TO_TRANSITIONS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK3). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Recipe Editor step name. For operations, this name is the phase name within the equipment module. For procedure and unit procedures this is the name of an existing unit procedure or operation.

SFC_STEP_TO_TRANSITIONS Table	
Attribute	**Attribute Domain**
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK4). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.
TRANSITION_ID	Data Type: Integer. Length: 6 digits. Key: Primary key (PK5). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: None. Description: Unique integer identifier used to distinguish between each transition within a recipe.

SFC_TRANSITION_TO_STEPS Table

The SFC_TRANSITION_TO_STEPS table represents the links from transitions to steps.

SFC_TRANSITION_TO_STEPS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.
TRANSITION_ID	Data Type: Integer. Length: 6 digits. Key: Primary key (PK3). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: None. Description: Unique integer identifier used to distinguish between each transition within a recipe.

SFC_TRANSITION_TO_STEPS Table	
Attribute	**Attribute Domain**
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK4). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Recipe Editor step name. For operations, this name is the phase name within the equipment module. For procedures and unit procedures, this is the name of an existing unit procedure or operation.
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK5). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.

RECIPE_FORCEDBINDS Table

The RECIPE_FORCEDBINDS table contains the forced binding information for the recipe.

RECIPE_FORCEDBINDS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2), Foreign key, (FK1). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.
STEP1_NAME	Data Type: Text. Length: 255 characters. Key: Primary key (PK3). Null Support: Nulls not allowed. Default Value: None. Description: The name of the first of two unit procedures using forced bindings.
STEP1_INSTANCE_ID	Data Type: Text. Length: 6 characters. Key: Primary key (PK4). Null Support: Nulls not allowed. Default Value: None. Description: The ID of the first unit procedure using forced binding.

RECIPE_FORCEDBINDS Table	
Attribute	**Attribute Domain**
STEP2_NAME	Data Type: Text. Length: 255 characters. Key: Primary key (PK5). Null Support: Nulls not allowed. Default Value: None. Description: The name of the second unit procedure using forced bindings.
STEP2_INSTANCE_ID	Data Type: Text. Length: 6 characters. Key: Primary key (PK6). Null Support: Nulls not allowed. Default Value: None. Description: The ID of the second unit procedure using forced bindings.
BIND_TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable values: 0 (use the same unit), or 1 (use different units). Default Value: None. Description: Specifies whether the forced binding pair of unit procedures must run on the same or different units.

RECIPE_PHYSICALLINKS Table

The RECIPE_PHYSICALLINKS table contains the Jacobson link information for the recipe.

RECIPE_PHYSICALLINKS Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Foreign key (FK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version. Do not modify.
KEY_NAME	For future use. Nulls allowed.
KEY_ID	For future use. Nulls allowed.
ORIGIN_STEP_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Default Value: None. Description: The name of the origin unit procedure in the Jacobson Link connection.

RECIPE_PHYSICALLINKS Table	
Attribute	**Attribute Domain**
ORIGIN_STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Null Support: Nulls not allowed. Default Value: None. Description: The name of the origin unit procedure ID in the Jacobson Link connection.
DESTINATION_STEP_NAME	Data Type: Text. Length: 255 characters. Null Support: Nulls not allowed. Default Value: None. Description: The name of the destination unit procedure in the Jacobson Link connection.
DESTINATION_STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Null Support: Nulls not allowed. Default Value: None. Description: The ID of the destination unit procedure in the Jacobson Link connection.

STEP_KEY_PROC_RPT Table

The STEP_KEY_PROC_RPT table represents information on key process report parameters.

STEP_KEY_PROC_RPT Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1), Foreign key (FK1) to RECIPE_STEPS. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2), Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK3), Foreign key (FK3). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Recipe Editor step name. For operations, this is the name of a phase within the equipment module. For procedures and unit procedures, this is the name of an existing unit procedure or operation.

STEP_KEY_PROC_RPT Table	
Attribute	**Attribute Domain**
STEP_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK4), Foreign key (FK4). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.
REPORT_NAME	Data Type: String. Length: 65 characters. Key: Primary key (PK5). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the key process report.
ACTIVE	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Indicates whether the key process report is active.

FORMPARAM Table

The FORMPARAM table contains a list of parameters associated with each formulation.

FORMPARAM Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1), Foreign key (FK1) to FORMULATION. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
FORMULATION_NAME	Data Type: String. Length: 40 characters. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: The formulation name.

FORMPARAM Table	
Attribute	**Attribute Domain**
FORMULATION_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK3). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: The formulation version number.
STEP_HIERARCHY	Data Type: String. Length: 255 characters. Key: Primary key (PK4). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: Step hierarchy.
PARAMETER_NAME	Data Type: String. Length: 60 characters. Null Support: Nulls not allowed. Allowable Values: ASCII string. Default Value: None. Description: The name of the parameter.
PARAMETER_TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (real), 2 (long), 3 (string), 4 (enumeration). Default Value: 1 Description: The type of data the step parameter represents.

FORMPARAM Table	
Attribute	**Attribute Domain**
DEFAULT_VALUE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls allowed. Allowable Values: Depends on the type of step parameter. Default Value: Low Limit for reals and longs. Description: The default value for the step parameter. Constraints: The value must be within the limits of one of the following: • The defined equipment parameters for operation steps. • The limits defined in the procedure or unit procedure parameter that this step represents.
DEFAULT_TEXT	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: Depends on the type of step parameter. Default Value: Depends on the type of step parameter. Description: The default text for the step parameter. Valid when the step type is string or enumeration. Also valid when the origin is deferred. ***NOTE:*** *The value for deferred steps is equal to the name of a recipe parameter from the identified recipe.*
ENUMERATION_SET	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the enumeration that contains the step parameter value. Only valid when the type is an enumeration.

FORMPARAM Table	
Attribute	**Attribute Domain**
MACHINE_PARAMETER	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: Default Value: 1 Description: The machine parameter.

FORMULATION Table

The FORMULATION table is used by formulations.

FORMULATION Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1), Foreign key (FK1) to MASTER_RECIPE. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.

FORMULATION Table	
Attribute	**Attribute Domain**
FORMULATION_NAME	Data Type: String. Length: 40 characters. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: The formulation name.
FORMULATION_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK3). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: The formulation version number.
FORMULATION_TYPE	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: The formulation type.
FORMULATION_DESCRIPTION	Data Type: String. Length: 132 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: Description of the formulation.

FORMULATION Table	
Attribute	**Attribute Domain**
FORMULATION_AUTHOR	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: Author of the formulation.
FORMULATION_PRODUCT_CODE	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: The user-defined product code of the material produced by this formulation.
FORMULATION_BATCH_SIZE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls allowed. Allowable Values: 0-9 and a decimal point. Description: The amount manufactured in one batch by this formulation. The units for this value are derived from the master recipe.
FORMULATION_VALID	Data Type: Integer. Length: 6 digits. Null Support: Nulls allowed. Allowable Values: 0,1 Default Value: 1 Description: Describes whether the formulation was in a valid state when saved. 0 is not valid. 1 is valid.

FORMULATION Table	
Attribute	**Attribute Domain**
MASTER_RECIPE_AUDIT_VERSION	Data Type: Integer. Length: 6 digits. Null Support: Nulls allowed. Allowable Values: 1 Default Value: 1 Description: The version of the master recipe against which the formulation was validated.
FORMULATION_STATUS	Data Type: Integer. Length: 6 digits. Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: The formulation status: • 0 is Draft • 1 is Ready • 2 is Approved • -1 is Withdrawn
FORMULATION_VERSIONDATE_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: A time stamp in UTC format. Default Value: None. Description: The date and time, in UTC format, that the formulation was validated.

Control Recipe Tables

During batch production, the Batch Execution Server uses the control recipe tables to allow process values to change without affecting the associated master recipes. You can update these tables with the Recipe Editor by creating and saving recipes to a relational database. Once updated, the control recipe tables are exact duplicates of the RECIPE_FORMULA and STEP_FORMULA tables for the saved recipe.

When a recipe runs, the Batch Execution Server reads the process inputs and parameters from the RECIPE_CONTROL_FORMULA and STEP_CONTROL_FORMULA tables. You can change the control parameters prior to batch execution without changing:

- The master recipe parameters (RECIPE_FORMULA table) and step parameters (STEP_FORMULA table).
- The way in which parameters are deferred to higher levels.

The following subsections describe the control recipe tables.

RECIPE_CONTROL_FORMULA Table

The RECIPE_CONTROL_FORMULA table represents the control recipe parameter process inputs and parameters.

RECIPE_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.

RECIPE_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
NAME	Data Type: String. Length: 60 characters. Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the recipe parameter.
TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (real), 2 (long), 3 (string), 4 (enumeration). Default Value: None. Description: The type of data the recipe parameter represents.
DEFAULT_TEXT	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: Depends on the type of parameter. Default Value: Depends on the type of parameter. Description: The default value of the parameter, saved as text. Only strings and enumerations are valid.
DEFAULT_VALUE	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls allowed. Allowable Values: Depends on the type of parameter, reals and longs only. Default Value: Depends on the type of parameter. Description: The default value of the parameter.

RECIPE_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
LOW_LIMIT	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: Depends on the type of recipe parameter, reals and longs only. Default Value: Depends on the type of recipe parameter. Description: The default value for the parameter.
HIGH_LIMIT	Data Type: Real. Length: 7 digits, including a decimal point. Null Support: Nulls not allowed. Allowable Values: Depends on the type of recipe parameter, real and longs only. Default Value: Depends on the type of recipe parameter. Description: The default value for the parameter.
ENUMERATION_SET	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the enumeration that contains the parameter value. Only valid when type is an enumeration.
ENGUNITS	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The engineering units associated with the parameter.

RECIPE_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
SCALABLE	Data Type: Integer. Length: 1 digit. Null Support: Nulls allowed. Allowable Values: 1 (scalable), 0 (not scalable). Default Value: 0 Description: Indicates if the parameter is to be scaled according to the scale of the batch.
ORIGIN	Data Type: Integer. Length: 99 digits. Null Support: Nulls not allowed. Allowable Values: 0 (value), 1 (defer), 2 (operator; operations only), 3 (defer – legacy), 4 (defer to unit procedure), 5 (defer to procedure). Default Value: 99 Description: Defines the level of the recipe that supplies the value for the step.

STEP_CONTROL_FORMULA Table

The STEP_CONTROL_FORMULA table represents the control step process inputs and parameters for the master recipe.

STEP_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
RECIPE_ID	Data Type: String. Length: 255 characters. Key: Primary key (PK1), Foreign key (FK1) to RECIPE_STEPS. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Depends on the procedure type (procedure, unit procedure, or operation). Description: Master recipe ID.

STEP_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
RECIPE_VERSION	Data Type: Integer. Length: 6 digits. Key: Primary key (PK2), Foreign key (FK2). Null Support: Nulls not allowed. Allowable Values: 1 Default Value: 1 Description: Batch Execution internal recipe version number. Do not modify.
STEP_NAME	Data Type: String. Length: 255 characters. Key: Primary key (PK3), Foreign key (FK3). Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: Recipe Editor step name. For operations, this name is the phase name within the equipment module. For procedure and unit procedure this is the name of an existing unit procedure or operation.
STEP_INSTANCE_ID	Data Type: String. Length: 6 characters. Key: Primary key (PK4), Foreign key (FK4). Null Support: Nulls not allowed. Allowable Values: Digits 0-9. Default Value: 1 Description: The Recipe Editor step instance identifier. A step may be included in more than one recipe. For each instance of the step, a unique instance ID is required. This is a number starting at 1.

STEP_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
NAME	Data Type: String. Length: 60 characters. Key: Primary key (PK5). Uniqueness: Unique. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the step parameter.
TYPE	Data Type: Integer. Length: 1 digit. Null Support: Nulls not allowed. Allowable Values: 1 (real), 2 (long), 3 (string), 4 (enumeration). Default Value: 1 Description: The type of data the step parameter represents.
ORIGIN	Data Type: Integer. Length: 99 digits. Null Support: Nulls not allowed. Allowable Values: 0 (value), 1 (defer), 2 (operator; operations only), 3 (defer – legacy), 4 (defer to unit procedure), 5 (defer to procedure). Default Value: 99 Description: Defines the level of the recipe that supplies the value for the step.

STEP_CONTROL_FORMULA Table	
Attribute	**Attribute Domain**
DEFAULT_VALUE	Data Type: Real. Length: 7 digits. Null Support: Nulls allowed. Allowable Values: Depends on the type of parameter. Default Value: Low limit for reals and longs. Description: The default step parameter value. Constraint: The value must be within the limits of one of the following: The defined equipment parameters for operations. The limits defined in the procedure or unit procedure parameter that this step represents. The recipe step name in the RECIPE_STEPS is equal to the recipe ID in the MASTER_RECIPE table for procedures and unit procedures. ***NOTE:*** *The value for deferred steps is equal to the name of a recipe parameter from the identified recipe.*
DEFAULT_TEXT	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: Depends on the type of parameter. Default Value: Depends on the type of parameter. Description: The default value of the step parameter, saved as text. Only strings and enumerations are valid.
ENUMERATION_SET	Data Type: String. Length: 60 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The name of the enumeration that contains the step parameter value. Only valid when type is an enumeration.

Appendix B: Batch Event Journal Data Model

The Batch Event Journal logical data model contains electronic batch data that is written to the relational database by the Batch Execution Archiver application. This data represents an electronic record for each batch and includes the following information:

- Control recipe states, transitions, and external events for the procedural elements.
- Process values that are sent to and from the process controller.
- Data representing the formula, procedural parameters, messages, and operator input.

In addition, the Event Journal data model:

- Is not specific to a particular vendor's implementation.
- Represents the tables, table attributes, constraints, and relationships among the tables.

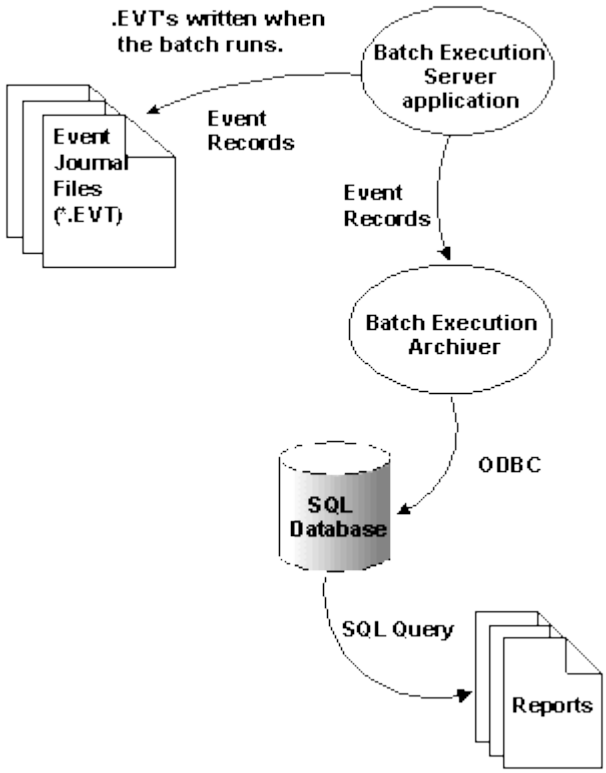
Creating the Data Model

You can create the tables shown in the Archiving Event Journal Data section by running the setup script for your relational database. There are also scripts for customers who are upgrading from previous versions of Batch Execution.

Existing and new customers should refer to the System Configuration Manual for information on running the script provided with Batch Execution to create the Event Journal data model in your relational database. For additional upgrade instructions, existing customers should refer to the Upgrade Guide.

Archiving Event Journal Data

After you run the setup script for your relational database, the Batch Execution data model is created. To satisfy all your reporting requirements, Batch Execution provides Active Journaling. Active Journaling is the process of recording event data in a relational database. The following figure illustrates the Active Journaling architecture.



Active Journaling Architecture

Attribute Domain

An *attribute* is a fact or non-decomposable piece of information describing the content of a table. Each table in the Event Journal data model is comprised of one or more attributes required by Batch Execution. An *attribute domain* defines a set of valid values for an attribute. Domain characteristics define both generic characteristics and entity-related characteristics. The generic characteristics describe the attribute's data type, length, allowable values, and meaning. The entities (table) characteristics define the attribute's uniqueness, null support, default value and additional constraints.

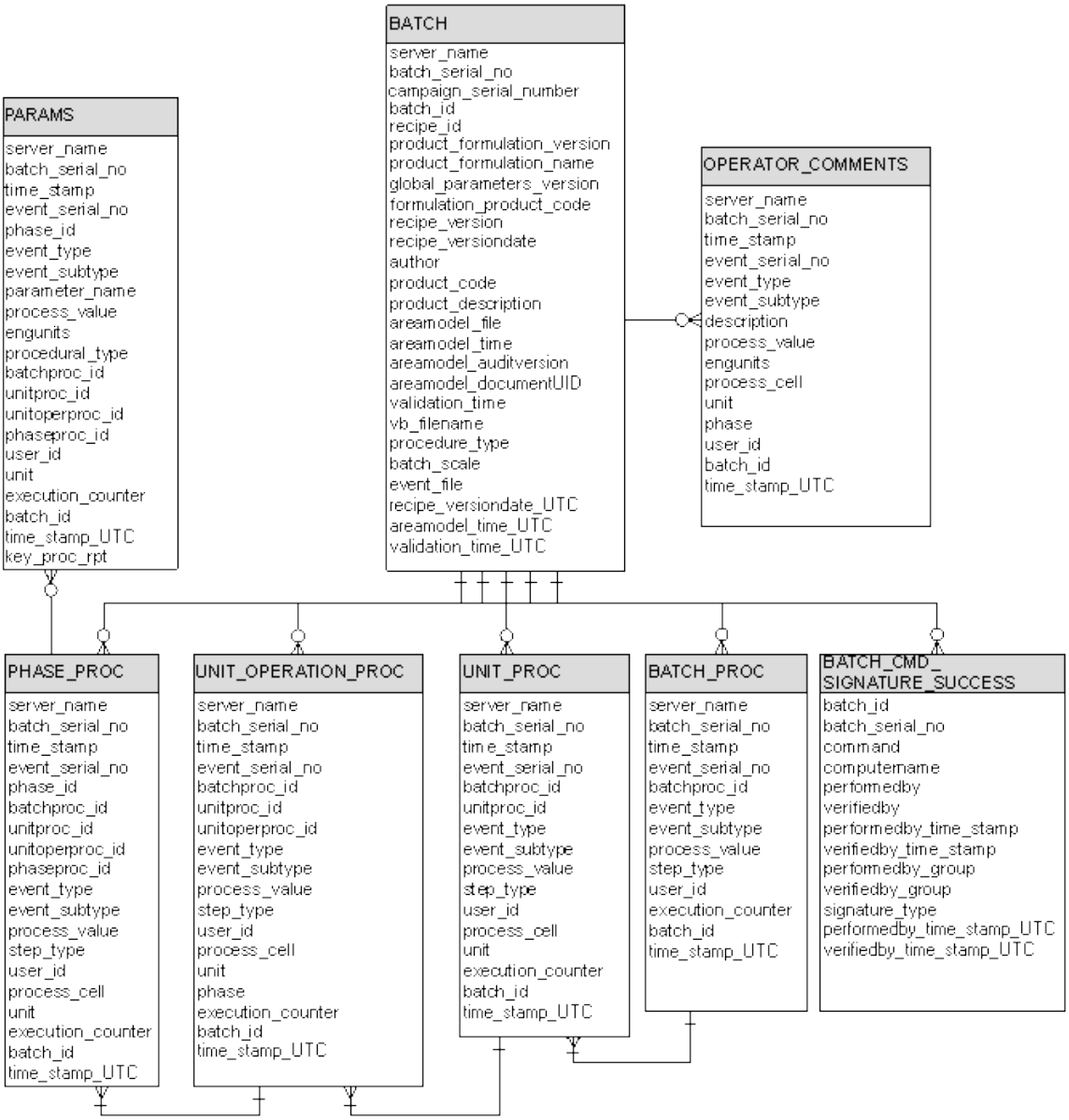
The following are the characteristics associated with each attribute in the model:

Attribute Domain	**Value/Description**
Data Type	Integer, Real, String, Text, Date.
Length	The maximum number of characters or digits supported by the data type.
Allowable Values	The range of values supported.

Attribute Domain	**Value/Description**
Uniqueness	Unique or non-unique. By default, attributes are non-unique.
Null Support	Allowed or not allowed.
Key	Indicates whether the attribute is part of a primary, alternate, or foreign key.
Default Value	The attribute value, if no value is supplied.
Description	Text describing the function of the attribute.
Constraints	Rules that enforce data integrity between and within tables.

Entity Relationship Diagram

The Event Journal entity relationship diagram, shown in the following figure, identifies the entities (tables) and the relationships between these entities that represent the collection of data related to one batch. The batch history for one batch is comprised of entries and events that are logged during the execution of the batch.



Batch Event Journal Entity Relationship Diagram

The following sections describe each table in the Event Journal data model.

The BATCH Table

The BATCH table stores the control recipe attributes of a batch. This includes the:

- Static information from the master recipe.
- Control parameters that are entered when a batch is scheduled.

Event Types

The following are the valid event types for the Batch table:

- Event File Name
- Recipe Header
- Batch Description
- Scale

The following table lists the attributes for the BATCH table.

BATCH Table	
Attribute	**Attribute Domain**
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Description: The server name.
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID that identifies an instance of a control recipe (a batch).

BATCH Table	
Attribute	**Attribute Domain**
CAMPAIGN_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Default value: 0 Description: A user-defined ID that identifies an instance of a campaign.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).
RECIPE_ID	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: A unique ID that identifies a master recipe.
PRODUCT_FORMULATION_VERSION	Data Type: String. Length: 4 characters. Null Support: Nulls allowed. Allowable Values: This value is incremented by Batch Execution. It can be any integer value, depending on how many times the formulation was saved. Default Value: 1. Description: If the batch is a formulation, this column contains the version number for the Formulation; otherwise, it is NULL.

BATCH Table	
Attribute	**Attribute Domain**
PRODUCT_FORMULATION_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: Description: If the batch is a formulation, this column contains the name of the formulation; otherwise, it is NULL.
GLOBAL_PARAMETERS_VERSION	Data Type: String. Length: 4 characters. Null Support: Nulls allowed. Allowable Values: This value is incremented by Batch Execution. It can be any integer value, depending on how many times the global parameters set was saved. Default Value: 1. Description: If the batch is a formulation and a global parameter set is defined, this column contains the version number of the Global Parameter set; otherwise, it is NULL.
FORMULATION_PRODUCT_CODE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any printable character. Default Value: None. Description: If the batch is a formulation, this column contains the user-defined product code of the material produced by the formulation; otherwise, it is NULL.

BATCH Table	
Attribute	**Attribute Domain**
RECIPE_VERSION	Data Type: String. Length: 4 characters. Null Support: Nulls allowed. Allowable Values: 1. Default Value: 1. Description: User defined version.
RECIPE_VERSIONDATE	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date. Default Value: None. Description: Date and time the recipe was last modified.
AUTHOR	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Alpha-Integer and special characters. Default Value: None. Description: Creator of recipe.
PRODUCT_CODE	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: Alpha-integer and special characters. Default Value: None. Description: Product code of the material produced by the recipe.

BATCH Table	
Attribute	**Attribute Domain**
PRODUCT_DESCRIPTION	Data Type: String. Length: 132 characters. Null Support: Nulls allowed. Allowable Values: Alpha-integer and special characters. Default Value: None. Description: Description of the product produced by the recipe.
AREAMODEL_FILE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Valid operating system path (c:\Program Files\Proficy\Proficy Batch Execution\Projects\DEMO\RECIPES\demo.cfg). Default Value: Set by the Batch Execution Recipe Editor to the area model directory and file name. Description: Windows drive, path name, and the name of the area model file that is used to build recipe.
AREAMODEL_TIME	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date. Default Value: None. Description: The date and time that the area model was last modified.

BATCH Table	
Attribute	**Attribute Domain**
AREAMODEL_AUDITVERSION	Data Type: Integer. Length: 10 digits. Null Support: Nulls allowed. Allowable Values: This number can range from 0 to 2147483647. Description: The audit version number of the area model file. The audit version number increases by one each time the area model is saved in the Batch Execution Equipment Editor.
AREAMODEL_DOCUMENTUID	Data Type: String. Length: 40 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Description: A unique, system-generated identifier for the file. Batch Execution generates the globally unique identifier (also known as a GUID) when you save a file for the first time, or use the Save As action to save a file in the Batch Execution Equipment Editor.
VALIDATION_TIME	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date. Default Value: None. Description: Time stamp when the recipe was validated against the specified area model.

BATCH Table	
Attribute	**Attribute Domain**
VB_FILENAME	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Valid file name. Default Value: None. Description: File name from which the Batch Execution Server reads the recipe during batch execution.
PROCEDURE_TYPE	Data Type: String. Length: 50 characters. Null Support: Nulls allowed. Allowable Values: 1 (batch procedure), 2 (unit procedure), or 3 (unit operation). Default Value: None. Description: Type of procedural element (based on ISA-S88.01).
BATCH_SCALE	Data Type: Integer. Length: 3. Null Support: Nulls allowed. Allowable Values: 0-999. Default Value: 100. Description: The scaling factor specified for the batch when the batch is created.
EVENT_FILE	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: Any. Default Value: None. Description: Full path and filename of event journal file (.EVT).

BATCH Table	
Attribute	**Attribute Domain**
RECIPE_VERSIONDATE_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date and time, in UTC format. Default Value: None. Description: Date and time, in UTC format, that the recipe was last modified.
AREAMODEL_TIME_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date and time, in UTC format. Default Value: None. Description: The date and time, in UTC format, that the area model was last modified.
VALIDATION_TIME_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid date and time, in UTC format. Default Value: None. Description: Time stamp, in UTC format, when the recipe was validated against the specified area model.

The BatchAnalysis Table

The BatchAnalysis table provides batch event data for use in Proficy Plant Applications Batch Analysis Reports. It includes all the event records received by the Archiver, with the exception of the records eliminated by the filters. The following table lists the attributes for the BatchAnalysis table.

The BatchAnalysis Table	
Attribute	**Attribute Domain**
LclTime	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Default Value: None. Description: The local date and time of the event.
campaign_serial_no	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Default Value: 0 Description: A user-defined ID that identifies an instance of a campaign.
BATCH_ID	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a batch.
RECIPE	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: A unique ID that identifies a master recipe.

The BatchAnalysis Table	
Attribute	**Attribute Domain**
DESCRIPT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: 1. Default Value: 1. Description: The description of the event. The description recorded in this field depends upon the event type.
EVENT	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: Type of event for the journal entry.
PVALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The process value. The value recorded in this field depends upon the event type.
EU	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The engineering units.

The BatchAnalysis Table	
Attribute	**Attribute Domain**
AREA	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The area model the batch executed from.
PROCCELL	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The process cell on which the batch executed.
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The unit on which the phase executed.
PHASE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The equipment phase instance name.

The BatchAnalysis Table	
Attribute	**Attribute Domain**
UNIQUEID	Data Type: String. Length: 12 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The unique serial number for the batch.
PHASEDESC	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The description of the phase.
USERID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled the Batch Execution event.
RECPTYPE	Data Type: String. Length: 50 characters. Null Support: Nulls not allowed. Allowable Values: Default Value: 1 Description: The type of recipe executed as part of the event.

The BatchAnalysis Table	
Attribute	**Attribute Domain**
SEQUENCE	Data Type: Integer. Length: 10 characters. Null Support: Nulls not allowed. Allowable Values: Default Value: 1 Description: A unique number for that event record.
SERVERNAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.

The BatchAnalysisLog Table

When event data is being archived to the Archiver3x table, if an error occurs, the information is inserted in the BatchAnalysisLog table, allowing the transaction to commit successfully. The data from the inserted row is saved into this table along with the SQL message information. The BatchAnalysisLog table provides diagnostic information for these unexpected error conditions that occur during the archival process.

The BatchAnalysisLog table is used with Proficy Plant Applications Batch Analysis Reports. The following table lists the attributes for the BatchAnalysisLog table.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
LclTime	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Default Value: None. Description: The local date and time of the event.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
campaign_serial_no	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Description: A user-defined ID that identifies an instance of a campaign.
BATCH_ID	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a batch.
RECIPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: A unique ID that identifies a master recipe.
DESCRIPT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: 1. Default Value: 1. Description: The description of the event. The description recorded in this field depends upon the event type.
EVENT	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: Type of event for the journal entry.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
PVALUE	Data Type: String. Length: 4000 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The process value. The value recorded in this field depends upon the event type.
EU	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The engineering units.
AREA	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The area model the batch executed from.
PROCCELL	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The process cell on which the batch executed.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
UNIT	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The unit on which the phase executed.
PHASE	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The equipment phase instance name.
UNIQUEID	Data Type: String. Length: 12 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The unique serial number for the batch.
PHASEDESC	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The description of the phase.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
USERID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled the Batch Execution event.
RECPTYPE	Data Type: String. Length: 50 characters. Null Support: Nulls allowed. Allowable Values: Default Value: 1 Description: The type of recipe executed as part of the event.
SEQUENCE	Data Type: Integer. Length: 10 characters. Null Support: Nulls allowed. Allowable Values: Default Value: 1 Description: A unique number for that event record.
SERVERNAME	Data Type: String. Length: 64 characters. Null Support: Nulls allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
TIMESTAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event occurred.
SQLError	Data Type: Integer. Length: 10 characters. Null Support: Nulls allowed. Default Value: 1 Description: The SQL error number. For more information on a specific error number, refer to the Microsoft Events and Errors Message Center: http://www.microsoft.com/technet/support/ee/ee_advanced.aspx.
SQLSeverity	Data Type: Integer. Length: 10 characters. Null Support: Nulls allowed. Default Value: None. Description: A number representing the severity of the error.
SQLdLevel	Data Type: Integer. Length: 10 characters. Null Support: Nulls not allowed. Default Value: 1 Description: A number representing the SQL level.
SQLDescription	Data Type: String. Length: 510 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: A description of the error.

The BatchAnalysisLog Table	
Attribute	**Attribute Domain**
LocalErrorMsg	Data Type: String. Length: 500 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The local error message.

The BatchAnalysisTrigger Table

The BatchAnalysisTrigger table is used along with the BatchAnalysis table and Proficy Plant Applications Batch Analysis Reports.

The BatchAnalysisTrigger table is used to determine when to start the copy of event data from the Batch Archiver3X table into the BatchAnalysis table. A row is inserted into the BatchAnalysisTrigger table when the first event indicating that a unit has been bound to the batch is inserted into the Archiver3X database. The existence of a row in this table for the batch indicates a unit has been bound to the batch and that the event data from Batch can be copied to the BatchAnalysis table. When the 'End of BATCH' event is inserted into the Archiver3X table, the row for the batch is removed from the BatchAnalysisTrigger table.

The following table lists the attributes for the BatchAnalysisTrigger table.

The BatchAnalysisTrigger Table	
Attribute	**Attribute Domain**
UNIQUEID	Data Type: String. Length: 12 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The unique serial number for the batch.

The BatchAnalysisTrigger Table	
Attribute	**Attribute Domain**
SERVERNAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIMESTAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event occurred.

The OPERATOR_COMMENTS Table

The Operator Comments (OPERATOR_COMMENTS) table contains the comments that the operator entered during the execution of the batch.

Event Types

The only valid event type for the Operator Comments table is User. The following table lists the attributes for the Operator Comments table.

OPERATOR_COMMENTS Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID that identifies an instance of a control recipe (a batch).
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time that the event occurred.
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server for each batch and for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Default Value: None. Description: The event type for journal entry.

OPERATOR_COMMENTS Table	
Attribute	**Attribute Domain**
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Default Value: None. Description: The classification of the event type.
DESCRIPTION	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any alphanumeric string. Default Value: None. Description: The description entered by the operator.
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Any alphanumeric string. Default Value: None. Description: The relevant value entered by the operator.
ENGUNITS	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The engineering units associated with relevant value.
PROCESS_CELL	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The relevant process cell.

OPERATOR_COMMENTS Table	
Attribute	**Attribute Domain**
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The relevant unit.
PHASE	Data Type: String. Length: 40 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The relevant phase.
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled batch execution.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).

OPERATOR_COMMENTS Table	
Attribute	**Attribute Domain**
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

The BATCH_PROC Table

The Batch Procedure (BATCH_PROC) table captures the execution activity of a batch procedure. A batch procedure is comprised of one or more Unit Procedures.

Event Types

The following are valid event types for the Batch Procedure Table:

- System Message
- Step Activity
- State Change
- Mode Change
- State Command
- Mode Command

The following table lists of attributes for the Batch Procedure table.

BATCH_PROC Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID that identifies an instance of a control recipe (a batch).
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: The date and time the event occurred.
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server for each batch and for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.
BATCHPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: Batch procedure recipe ID.

BATCH_PROC Table	
Attribute	**Attribute Domain**
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: Type of event for the journal entry.
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: Classification of event type.
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Dependent on the event subtype. Default Value: None. Description: The event process value, appropriate for the event type.
STEP_TYPE	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: NULL or BPC, UPC, UOP, or PHASE. Default Value: NULL. Description: Indicates if the step is a unit procedure, unit operation, or phase.

BATCH_PROC Table	
Attribute	**Attribute Domain**
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: ID of the user who controlled batch execution.
EXECUTION_COUNTER	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Indicates the number of times a recipe component has been executed. Applies to when a recipe contains a loop.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.

BATCH_PROC Table	
Attribute	**Attribute Domain**
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

The UNIT_PROC Table

The Unit Procedure (UNIT_PROC) table captures the execution activity of a unit procedure. This consists of state and mode changes, step activation, and deactivation.

Event Types

The following are the valid event types for the Unit Procedure table:

- System Message
- Step Activity
- State Change
- Mode Change
- State Command
- Mode Command
- Active Binding

The following table lists the attributes for the Unit Procedure table.

UNIT_PROC Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by Batch Execution Server, 0 to 2147483647. Default Value: None. Description: Unique ID used to identify an instance of a control recipe (a batch).
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event occurred.
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by Batch Execution Server per batch for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.
BATCHPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The batch procedure recipe ID or unit procedure recipe ID, if it is not part of a batch procedure recipe.

UNIT_PROC Table	
Attribute	**Attribute Domain**
UNITPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The unit procedure recipe ID.
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: The event type of the journal entry.
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The classification of the event type.
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Dependent on event subtype. Default Value: None. Description: The event process value, appropriate for the event type.

UNIT_PROC Table	
Attribute	**Attribute Domain**
STEP_TYPE	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: NULL or unit operation. Default Value: NULL. Description: Recorded as part of step activation.
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: ID of the user who controlled batch execution.
PROCESS_CELL	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The process cell on which the batch executed.
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The process cell on which the batch executed.

UNIT_PROC Table	
Attribute	**Attribute Domain**
EXECUTION_COUNTER	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Indicates the number of time a recipe component has been executed. Applies to when a recipe contains a loop.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

The UNIT_OPERATION_PROC Table

The Unit Operation Procedure (UNIT_OPERATION_PROC) table captures the execution activity of a unit operation procedure. This consists of state and mode changes, step activation, and step deactivation.

Event Types

The following are the valid event types for the Unit Operation Procedure table:

- System Message
- Step Activity
- State Change
- Mode Change
- State Command
- Mode Command

The following table lists the attributes for the Unit Operation Procedure table.

UNIT_OPERATION_PROC Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID that identifies an instance of a control recipe (a batch).
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: The date and time that the event occurred.

UNIT_OPERATION_PROC Table	
Attribute	**Attribute Domain**
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server for each batch and for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.
BATCHPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The batch procedure recipe ID or unit procedure recipe ID, if it is not part of a batch procedure recipe.
UNITPROC_ID	Data Type: String. Length: 255 characters. ***NOTE:*** *If you want to support BATCH IDs with a length of 128 characters, instead of 40, you need to change the length to 128 in your database software.* Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: Unit procedure recipe ID.
UNITOPERPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: Unit operation recipe ID.

UNIT_OPERATION_PROC Table	
Attribute	**Attribute Domain**
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: The event type of the journal entry.
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The classification of the event type.
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Dependent on the event subtype. Default Value: None. Description: The event process value, appropriate for the event type.
STEP_TYPE	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: NULL or Phase. Default Value: NULL. Description: Recorded as part of step activation.

UNIT_OPERATION_PROC Table	
Attribute	**Attribute Domain**
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled batch execution.
PROCESS_CELL	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The process cell on which the batch executed.
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The unit on which the unit procedure executed.
PHASE	Data Type: String. Length: 40 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The relevant phase.

UNIT_OPERATION_PROC Table	
Attribute	**Attribute Domain**
EXECUTION_COUNTER	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Indicates the number of times a recipe component has been executed. Applies to a looping recipe.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

The PHASE_PROC Table

The Phase (PHASE_PROC) table captures the execution activity of a phase. This consists of state and mode changes, step activation, and step deactivation.

Event Types

The following are the valid event types for the Phase table:

- System Message
- Step Activity
- State Change
- Mode Change
- State Command
- Mode Command
- Phase Logic Arbitration
- Permissive Sent
- Permissive Received
- Permissive Canceled

The following table lists the attributes for the Phase table.

PHASE_PROC Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID identifies an instance of a control recipe (a batch).

PHASE_PROC Table	
Attribute	**Attribute Domain**
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: The date and time the event occurred.
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server for each batch and for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.
PHASE_ID	Data Type: String. Length: 40 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The equipment phase instance name.
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: The event type of the journal entry.

PHASE_PROC Table	
Attribute	**Attribute Domain**
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The classification of the event type.
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: Dependent on the event subtype. Default Value: None. Description: The event process value, appropriate for the event type.
BATCHPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The batch procedure recipe ID, if the phase is part of a batch procedure recipe.
UNITPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The unit procedure recipe ID, if phase is part of a unit procedure recipe.

PHASE_PROC Table	
Attribute	**Attribute Domain**
UNITOPERPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The unit operation ID, if the phase is part of a unit operation.
PHASEPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The recipe phase name.
STEP_TYPE	Data Type: String. Length: 32 characters. Null Support: Nulls allowed. Allowable Values: NULL or BPC, UPC, UOP, or PHASE. Default Value: NULL. Description: Indicates if the step is a unit procedure, unit operation, or phase.
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled batch execution.

PHASE_PROC Table	
Attribute	**Attribute Domain**
PROCESS_CELL	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The process cell on which the batch executed.
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The unit on which the phase executed.
EXECUTION_COUNTER	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by Batch Execution Server (1 based). Default Value: 1. Description: Indicates the number of times a recipe component has been executed. Applies to when a recipe contains a loop.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).

PHASE_PROC Table	
Attribute	**Attribute Domain**
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

The PARAMS Table

The Parameters (PARAMS) table captures the recipe data for a procedural entity. This consists of:

- Recipe and step formula data that is defined in the master recipe.
- Modifications made to the recipe by the operator.
- Operator responses to parameter requests. Parameter values can be modified at all levels within a control recipe. Therefore, the data identifies the recipe component to which the parameter data corresponds.

Event Types

The following are the valid event types for the Parameters table:

- Report
- Prompt
- Prompt Response
- Message
- Recipe Value
- Recipe Value Change

- Comment
- Parameter Download Verified

The following table lists the attributes for the Parameters table.

PARAMS Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: Unique ID used to identify an instance of a control recipe (a batch).
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event occurred.
EVENT_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by Batch Execution Server per batch for each event. Default Value: 1. Description: Used to distinguish between multiple events occurring at the same time.

PARAMS Table	
Attribute	**Attribute Domain**
PHASE_ID	Data Type: String. Length: 40 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The equipment phase instance name.
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: The event type of the journal entry.
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The classification of the event type.
PARAMETER_NAME	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: ASCII string. Default Value: None. Description: The name of the parameter.

PARAMS Table	
Attribute	**Attribute Domain**
PROCESS_VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: Dependent on event subtype. Default Value: None. Description: The event process value, appropriate for the parameter.
ENGUNITS	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, and the underscore character. Default Value: None. Description: The engineering units associated with the parameter.
PROCEDURAL_TYPE	Data Type: String. Length: 50 characters. Null Support: Nulls not allowed. Allowable Values: 1 (procedure), 2 (unit procedure), 3 (unit operation), 4 (phase). Default Value: None. Description: The procedural element on which the event occurred (based on S88.01).
BATCHPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The batch procedure recipe ID, if the phase is part of a batch procedure recipe.

PARAMS Table	
Attribute	**Attribute Domain**
UNITPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The unit procedure recipe ID, if the phase is part of a unit procedure recipe.
UNITOPERPROC_ID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The unit operation ID, if the phase is part of a unit procedure recipe.
PHASEPROC_ID	Data Type: String. Length: 128 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscores (_). Default Value: None. Description: The recipe phase name.
USER_ID	Data Type: String. Length: 80 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled batch execution.

PARAMS Table	
Attribute	**Attribute Domain**
UNIT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: Process cell on which the batch executed.
EXECUTION_COUNTER	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Indicates the number of times a recipe component has been executed. Applies to when a recipe contains a loop.
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.

PARAMS Table	
Attribute	**Attribute Domain**
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.
Key_Proc_Rpt	Data Type: Integer. Length: 10 digits. Null Support: Nulls allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Key process report parameter.

The BATCH_CMD_SIGNATURE_SUCCESS Table

The BATCH_CMD_SIGNATURE_SUCCESS table captures the electronic signatures for the batch events. This table includes signature events for all ActiveX controls, except the EWI ActiveX control. EWI and EIB signatures are captured in other tables. The following table lists the attributes for the BATCH_CMD_SIGNATURE_SUCCESS table.

BATCH_CMD_SIGNATURE_SUCCESS Table	
Attribute	**Attribute Domain**
BATCH_ID	Data Type: String. Length: 128 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a control recipe (a batch).

BATCH_CMD_SIGNATURE_SUCCESS Table	
Attribute	**Attribute Domain**
BATCH_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Allowable Values: Sequential number generated by the Batch Execution Server. This number can range from 0 to 2147483647. Default Value: None. Description: A unique ID that identifies an instance of a control recipe (a batch).
COMMAND	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: The batch command that the ActiveX control performed.
COMPUTERNAME	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: The name of the Windows computer that the batch ActiveX control is running on.
PERFORMEDBY	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: Name of the person who performed the command. This field is blank if the Performed By signature is not configured for this command.

BATCH_CMD_SIGNATURE_SUCCESS Table	
Attribute	**Attribute Domain**
VERIFIEDBY	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: Name of the person who approved the command. This field is blank if the Verified By signature is not configured for this command.
PERFORMEDBY_TIME_STAMP	Data Type: Date/time. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event was performed and signed. If there is no Performed By signature required, then this field is Null.
VERIFIEDBY_TIME_STAMP	Data Type: Date/time. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: Date and time the event was verified and signed. If there is no Verified By signature required, then this field is Null.
PERFORMEDBY_GROUP	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: The Windows group to which the user who performed the action belongs.

BATCH_CMD_SIGNATURE_SUCCESS Table	
Attribute	**Attribute Domain**
VERIFIEDBY_GROUP	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: The Windows group to which the user who verified the action belongs.
SIGNATURE_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z, 0-9, and underscore (_). Default Value: None. Description: The type of signature: performed by, verified by, or approved by.
PERFORMEDBY_TIME_STAMP_UTC	Data Type: Date/time. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: Date and time, in UTC format, that the event was performed and signed. If there is no Performed By signature required, then this field is Null.
VERIFIEDBY_TIME_STAMP_UTC	Data Type: Date/time. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: Date and time, in UTC format, that the event was verified and signed. If there is no Verified By signature required, then this field is Null.

The ARCHIVER3X Table

The Archiver 3.x table (ARCHIVER3X) table replicates the format of the unique table in the previous Visual*Batch* version (3.x). It includes all the event records received by the Archiver, with the exception of the records eliminated by the filters. The following table lists the attributes for the Archiver3x table.

The Archiver3x Table	
Attribute	**Attribute Domain**
LclTime	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Default Value: None. Description: The local date and time of the event.
CAMPAIGN_SERIAL_NO	Data Type: Integer. Length: 10 digits. Null Support: Nulls not allowed. Description: A user-defined ID that identifies an instance of a campaign.
BATCH_ID	Data Type: String. Length: 125 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: Dependent upon user configurations. Description: A user-defined ID that identifies an instance of a batch.
RECIPE	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: A unique ID that identifies a master recipe.

The Archiver3x Table	
Attribute	**Attribute Domain**
DESCRIPT	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: 1. Default Value: 1. Description: The description of the event. The description recorded in this field depends upon the event type.
EVENT	Data Type: String. Length: 125 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: Type of event for the journal entry.
PVALUE	Data Type: String. Length: 4096 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The process value. The value recorded in this field depends upon the event type.
EU	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: None. Description: The engineering units.

The Archiver3x Table	
Attribute	**Attribute Domain**
AREA	Data Type: String. Length: 125 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The area model the batch executed from.
PROCCELL	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The process cell on which the batch executed.
UNIT	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_), and colon (:). Default Value: None. Description: The unit on which the phase executed.
PHASE	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, and the underscore character. Default Value: None. Description: The equipment phase instance name.

The Archiver3x Table	
Attribute	**Attribute Domain**
UNIQUEID	Data Type: String. Length: 12 characters. Null Support: Nulls not allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The unique serial number for the batch.
PHASEDESC	Data Type: String. Length: 125 characters. Null Support: Nulls allowed. Allowable Values: a-z, A-Z , 0-9, underscores (_) and colon (:). Default Value: None. Description: The description of the phase.
USERID	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: NULL or user name. Default Value: NULL. Description: The ID of the user who controlled the Batch Execution event.
RECPTYPE	Data Type: String. Length: 50 characters. Null Support: Nulls not allowed. Allowable Values: Default Value: 1 Description: The type of recipe executed as part of the event.

The Archiver3x Table	
Attribute	**Attribute Domain**
SEQUENCE	Data Type: Integer. Length: 10 characters. Null Support: Nulls not allowed. Allowable Values: Default Value: 1 Description: A unique number for that event record.
SERVERNAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
LclTime_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Default Value: None. Description: The local date and time, in UTC format.
Key_Proc_Rpt	Data Type: Integer. Length: 10 digits. Null Support: Nulls allowed. Allowable Values: Sequential number generated by the Batch Execution Server (1 based). Default Value: 1. Description: Key process report parameter.

The BATCH_SYSTEM_STATUS Table

The BATCH_SYSTEM_STATUS table captures the heartbeat event messages, when a heartbeat is enabled in the Batch Execution Workspace in the Server tab of the Batch Execution Configuration dialog box. You can configure the time interval, in minutes, between heartbeat events. The shorter the interval, the more likely you will need to maintain this table in your database software to prevent it

from growing too rapidly.

You can use this event message in external applications to detect communication failure from the Batch Execution Server and the database.

The following table lists the attributes for the BATCH_SYSTEM_STATUS table.

BATCH_SYSTEM_STATUS Table	
Attribute	**Attribute Description**
SERVER_NAME	Data Type: String. Length: 64 characters. Null Support: Nulls not allowed. Allowable Values: Windows computer name. Default Value: None. Description: The computer name on which the Batch Execution Server resides.
TIME_STAMP	Data Type: Date. Length: Not applicable. Null Support: Nulls not allowed. Allowable Values: Valid time stamp. Default Value: Current date and time. Description: The date and time the event occurred.
EVENT_TYPE	Data Type: String. Length: 40 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: Batch System Status. Description: The event type of the journal entry.
EVENT_SUBTYPE	Data Type: String. Length: 255 characters. Null Support: Nulls allowed. Allowable Values: System-defined. Default Value: Heartbeat. Description: The classification of the event type.

BATCH_SYSTEM_STATUS Table	
Attribute	**Attribute Description**
VALUE	Data Type: String. Length: 255 characters. Null Support: Nulls not allowed. Allowable Values: System-defined. Default Value: None. Description: The value of the heartbeat event. For normal operations, if the heartbeat exists, the value is BatchServer.
TIME_STAMP_UTC	Data Type: Date. Length: Not applicable. Null Support: Nulls allowed. Allowable Values: Valid time stamp, in UTC format. Default Value: Current date and time. Description: The date and time, in UTC format, that the event occurred.

Event File Mapping Example

The following table shows examples of event data that is mapped to the tables in Event Journal data model.

Event File Mapping Example		
Field Name	**Column Name**	**Example**
Unique ID	Batch Serial Number	19
LclTime	Time Stamp	10:37:29 PM
Recipe	Batch Procedure Recipe ID	MAKE_BASE
Event	Event Type	State Command
Description	Event Subtype	State Commanded
Pvalue	Value	Start

Event File Mapping Example		
Field Name	**Column Name**	**Example**
EU	Step Type	NULL
UserID	User ID	JBANGS
Procedure Type	Recipe Type	UPC
Serial Number	EvtSerialNo.	12

The Recipe ID stored in the event file consists of three concatenated elements:

- Batch serial number
- Recipe ID
- Recipe execution counter

For example, 19:MAKE_BASE-1.

For the Event type "Step Activity", the EU field denotes the type of step. Valid steps for a batch procedure step are initial, unit procedure, and terminal.

Index

C

D

E