GE VERNOVA

PROFICY® SOFTWARE & SERVICES

PROFICY BATCH EXECUTION 5.6

VBIS Automation Reference

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 VBIS Automation Reference is intended for integrators and programmers who want to develop applications that access and manipulate information within the Proficy Batch Execution environment through a set of automation interfaces. This help file assumes the reader is proficient in the Microsoft® Visual Basic® or Visual C++™ programming languages.

The following sections provide more details on what VBIS is and how to use the interfaces, properties, methods, and safe array values associated with it:

- Overview
- Using VBIS with Visual Basic
- Using VBIS with C++
- VBIS Language Quick Reference

Reference Documents

For related information on VBIS, refer to the following document:

Custom Applications manual

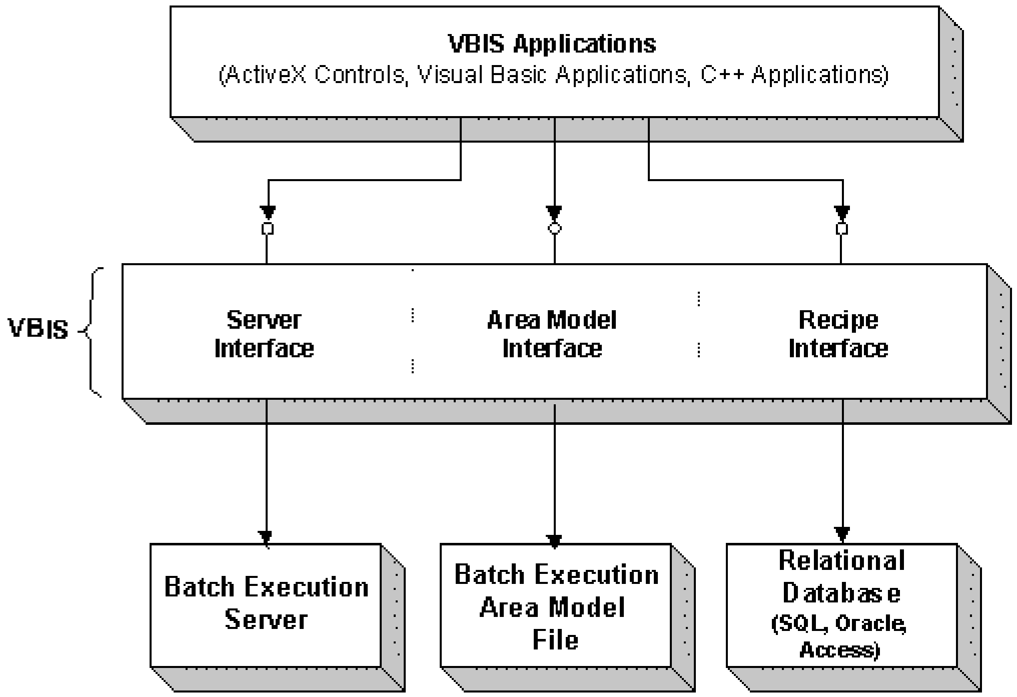
Overview

VBIS is a set of automation interfaces that provide a mechanism for third party applications (Visual Basic and Visual C++) to access and manipulate information within the Proficy Batch Execution environment. Using these object interfaces, you can write applications such as:

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

Note: *Before you can begin to use VBIS, you must install the software protection key. VBIS will not work without a key.*

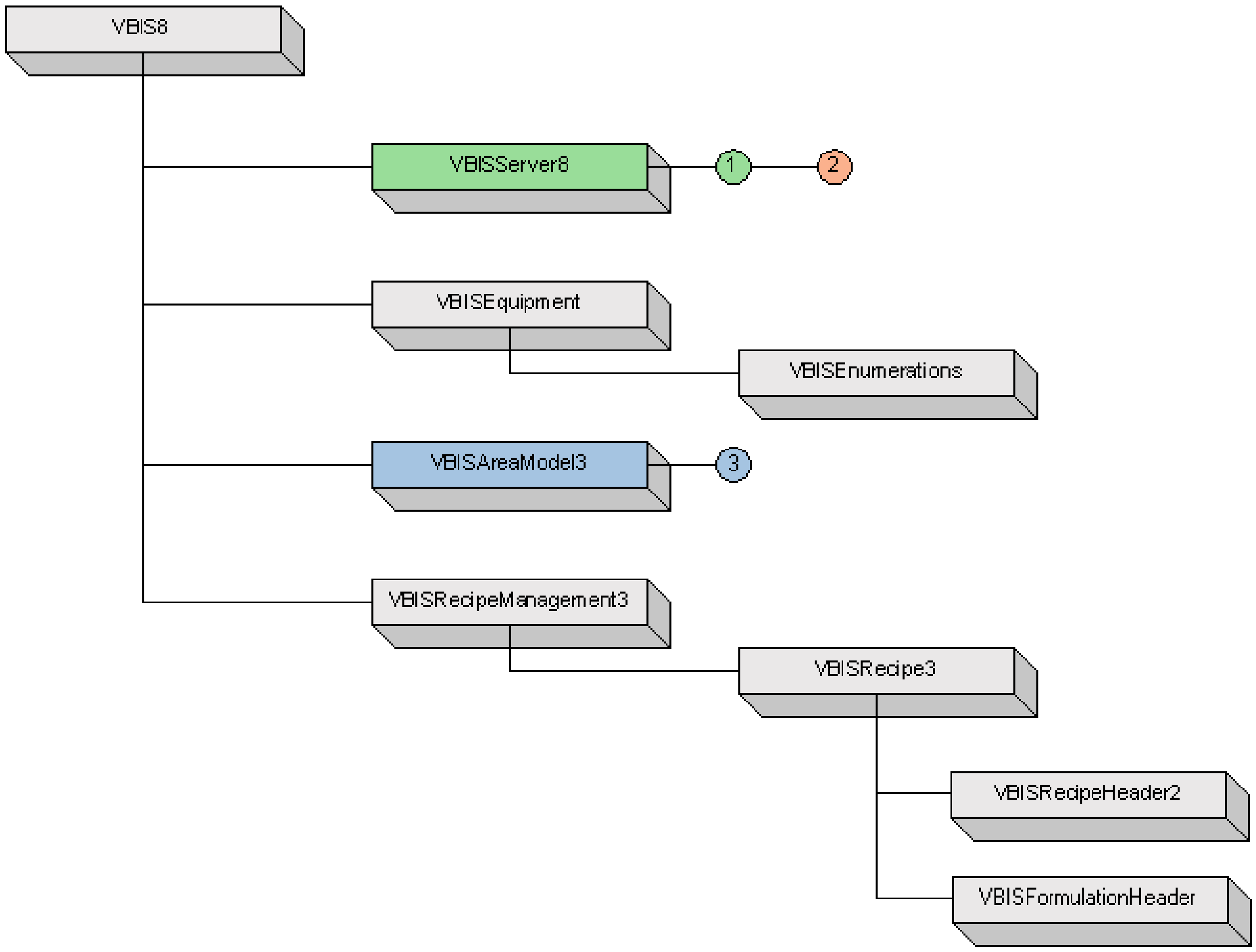
The following figure summarizes the interaction among an application, VBIS, and other Batch Execution components.


VBIS Applications
(ActiveX Controls, Visual Basic Applications, C++ Applications)
VBIS
Server Interface
Area Model Interface
Recipe Interface
Batch Execution Server
Batch Execution Area Model File
Relational Database (SQL, Oracle, Access)

Note: *If you create an application that replaces the Batch Execution Client, you must configure the VBVIEW32.INI file on the client computer to point to the computer where the remote Batch Execution Server resides (if remote server connection is desired).*

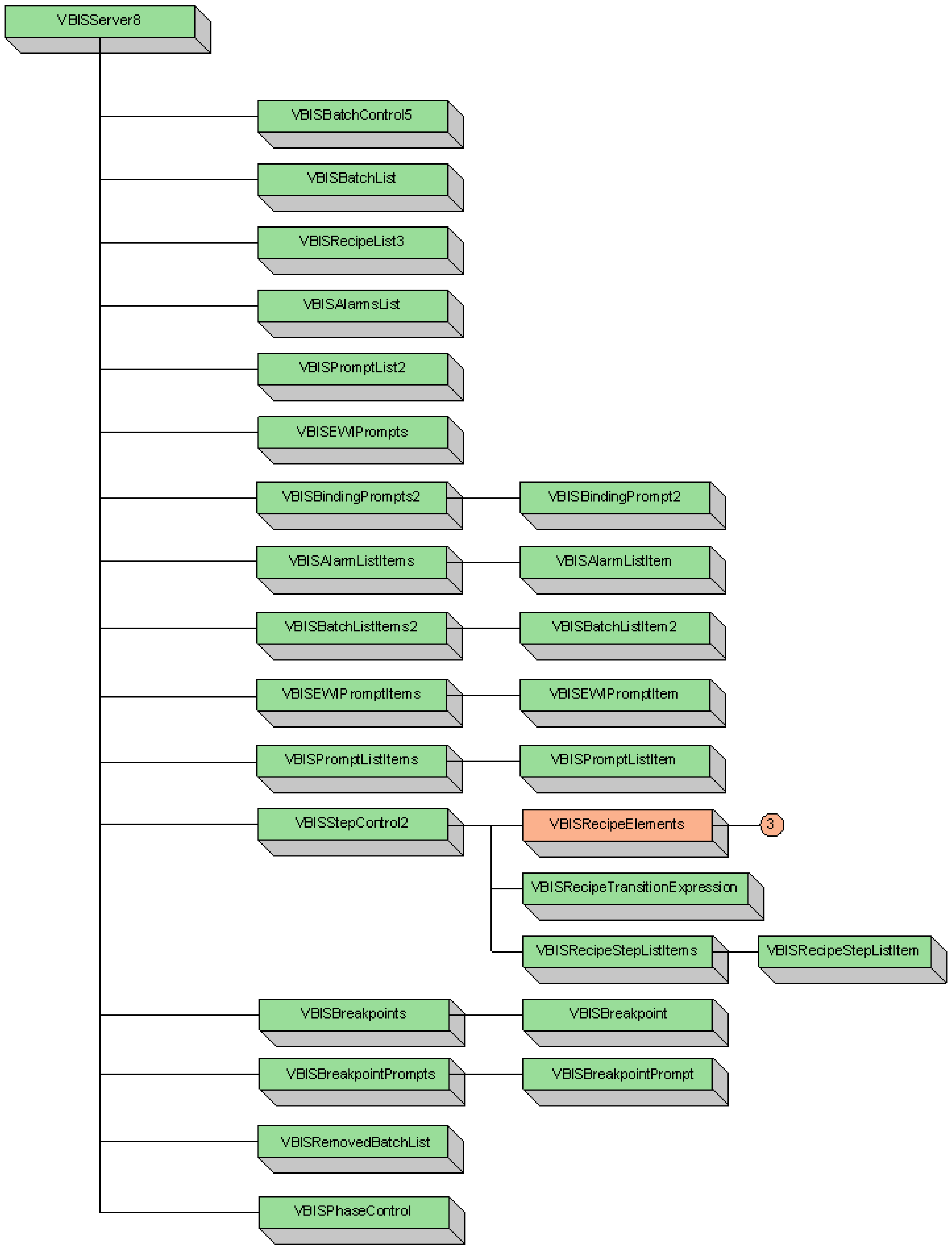
Understanding the VBIS8 Automation Interface Hierarchy

The graphic below shows the **VBIS8** automation interface hierarchy. To get more information on each object, click the object name in the graphic.



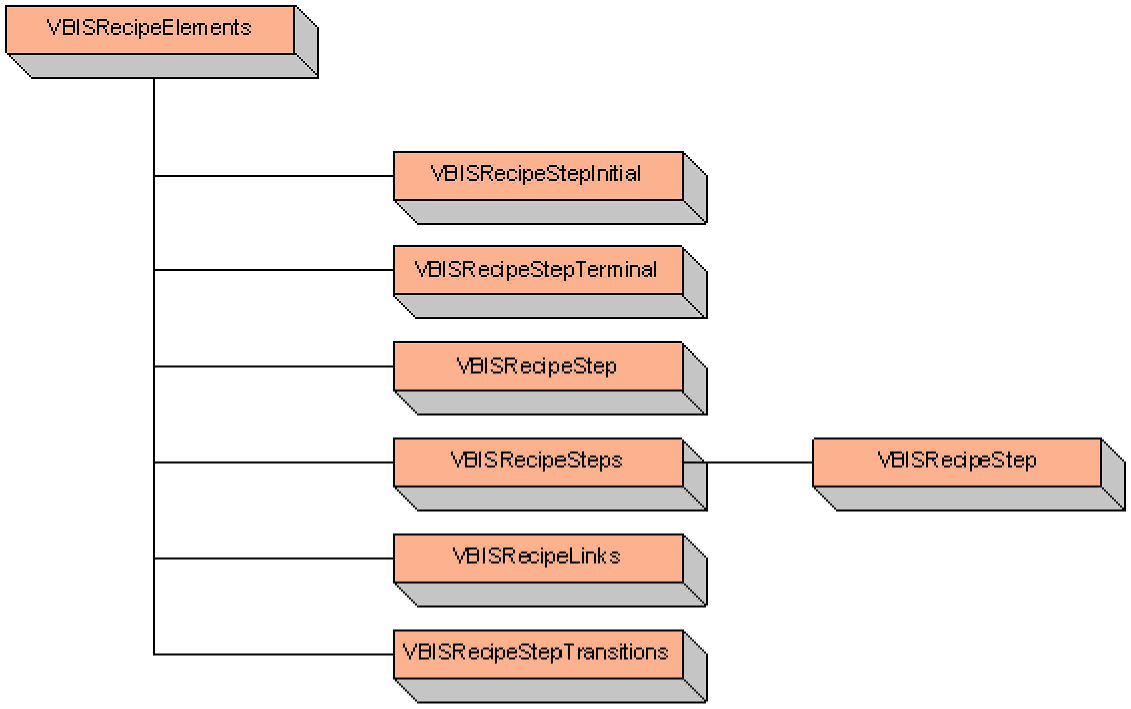
Understanding the VBISServer8 Hierarchy

The graphic below shows the **VBISServer8** hierarchy. To get more information on each object, click the object name in the graphic.


VBISServer8
VBISBatchControl5
VBISBatchList
VBISRecipeList3
VBISAlarmsList
VBISPromptList2
VBISEWIPrompts
VBISBindingPrompts2
VBISBindingPrompt2
VBISAlarmListItems
VBISAlarmListItem
VBISBatchListItems2
VBISBatchListItem2
VBISEWIPromptItems
VBISEWIPromptItem
VBISPromptListItems
VBISPromptListItem
VBISStepControl2
VBISRecipeElements
3
VBISRecipeTransitionExpression
VBISRecipeStepListItems
VBISRecipeStepListItem
VBISBreakpoints
VBISBreakpoint
VBISBreakpointPrompts
VBISBreakpointPrompt
VBISRemovedBatchList
VBISPhaseControl

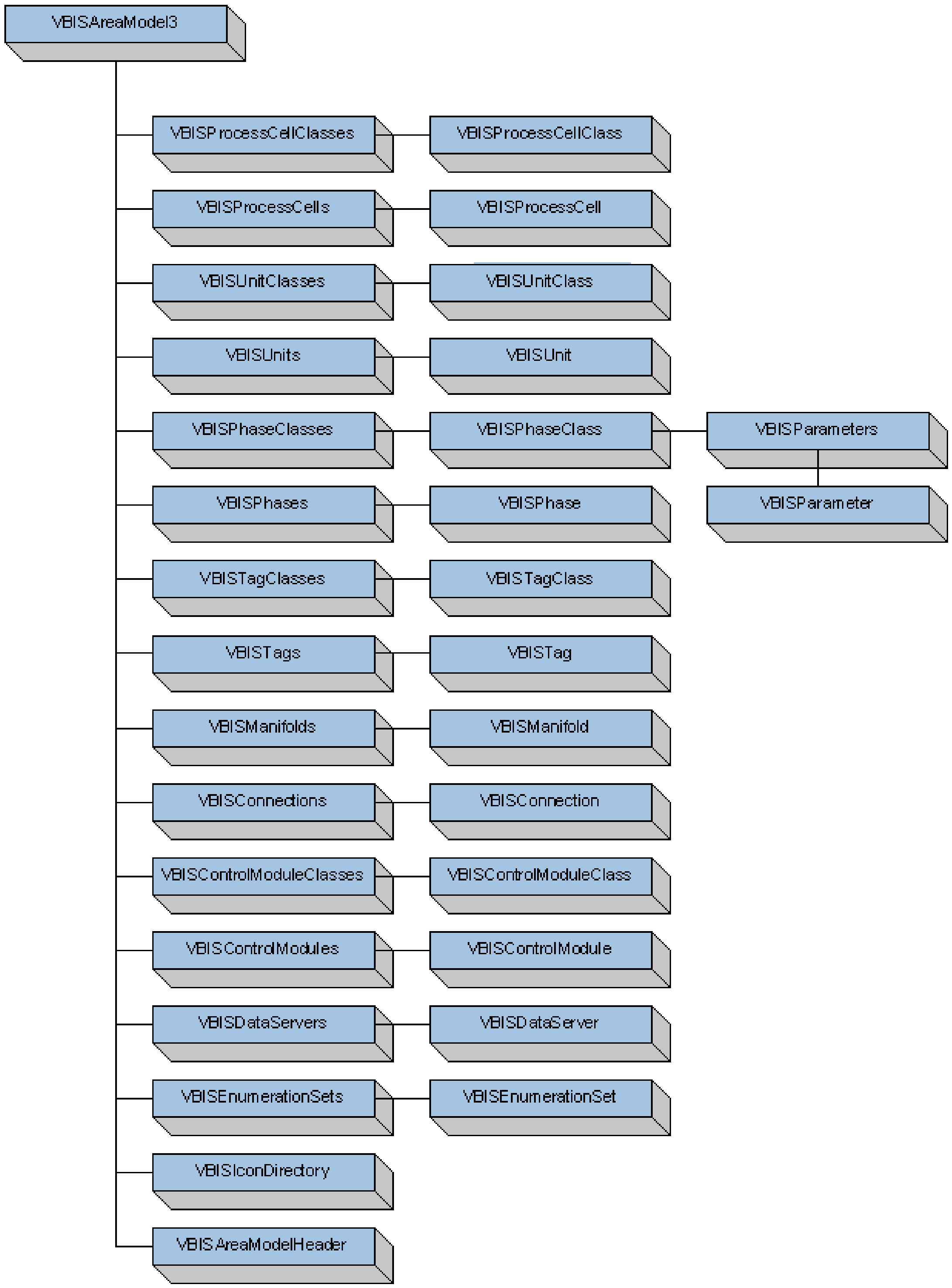
Understanding the VBISRecipeElements Hierarchy

The graphic below shows the **VBISRecipeElements** hierarchy. To get more information on each object, click the object name in the graphic.



Understanding the VBISAreaModel3 Hierarchy

The graphic below shows the **VBISAreaModel3** hierarchy. To get more information on each object, click the object name in the graphic.



Using VBIS with 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.

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

To obtain help on any dialog field or control, press F1 or right-click the mouse.

Accessing the VBIS Type Library from Visual Basic

To work with VBIS objects within your Visual Basic program, you must first include it as a reference into your Visual Basic program. In Visual Basic v6.0, do the following to reference VBIS:

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.

Parameter Syntax

In Visual Basic, method calls that return a value (FUNCTIONS) require parentheses around the group of parameters. Method calls that don’t return a value (PROCEDURES) do not require parentheses. For example:

Batch_State_Returned = BCObj.State(RecipeID) <- Functions require parentheses around parameters

BCObj.Command Recipe_ID, “STOP” <- Procedures do not use parentheses around parameters

Properties, like procedures, do not require parentheses.

Count = ENObj.CountEnumSet <- properties do not require parentheses.

Creating and Releasing VBIS Objects in Visual Basic

Visual Basic programs use an automation object. To create an automation object, use the following syntax:

```
Dim varObject As VBIS8
Set varObject = CreateObject("Intellution.VBIS.8")
```

To use a call to a remote computer, use the following syntax, but replace *Computer* with the actual name of the remote computer:

```
Dim varObject As VBIS8
Set varObject = CreateObject("Intellution.VBIS.8", "Computer")
```

When your application has made all required VBIS8 calls, you need to release the object. Typically, this is done prior to the application shutting down. To release an OLE object in a Visual Basic application, use the following syntax:

```
Set varObject = Nothing
```

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. The VBIS area model is made up of objects that represent S88.01 entities, such as process cells, units, and equipment phases. VBIS groups these objects together as collections based upon the class of the object. A class represents a collection; and the items of that class are the instances within the area model. 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.

Using VBIS with 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.*

Creating, Initializing, and Releasing VBIS Objects in C++

Before using any of the examples in this help system, you must initialize the COM library and create a VBIS8 object. The **CoInitialize** function initializes the Component Object Model(COM) library. You must initialize the library before you can call its functions. The **CoCreateInstance** function creates a single uninitialized object of the VBIS8 class. To initialize VBIS8, use the following code segment once at the beginning of your application:

```
HRESULT hr;
BOOL bReturn = TRUE; // Everything is fine so far

// Initialize the COM library
hr = CoInitialize (NULL);
if (SUCCEEDED (hr))
{
 // Create a single uninitialized object of the class IVBIS8
 m_pIVBIS8 = NULL;
 hr = ::CoCreateInstance (CLSID_VBIS8,
      NULL,
      CLSCTX_SERVER,
      IID_IVBIS8,
      (void**)&m_pIVBIS8);

 if (FAILED (hr))
 {
  bReturn = FALSE;
 }
}

else
{
 bReturn = FALSE;
}
```

Releasing a VBIS Object

Your application needs to release the VBIS8 object and un-initialize the COM library. To release the VBIS8 object, use the following code segment once at the end of your application:

```
// Do I have a VBIS8 object
if (m_pIVBIS8)
{
 // Release it
 m_pIVBIS8->Release ();
 // Un-Initialize the COM library
 CoUninitialize ();
}
```

VBIS Language Reference

Interfaces

The section below lists the interfaces for the VBIS8 Interface, in alphabetical order from VBISA-Z.

VBIS8 Interface

The **VBIS8** interface is the root object in the **VBIS8** automation interface hierarchy. The **VBIS8** interface provides access to the following lower-level interfaces:

- VBISServer8
- VBISEquipment
- VBISAreaModel3
- VBISRecipeManagement3

VBISActiveRecipeStepListItems Interface

The **VBISActiveRecipeStepListItems** interface provides access to a filtered collection of VBISRecipeStepListItems objects. The collection is filtered by the state of the recipe step.

Syntax

object.**VBISActiveRecipeStepListItems***(bsProcID, lActiveStepMask)*

The **VBISActiveRecipeStepListItems** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsProcID	BSTR (C++) String (Visual Basic)	The batch serial number.
lActiveStepMask	LONG	The mask for the desired active state(s) in the recipe step collection: 1 – Running or Restarting 2 – Held or Holding 4 – Stopped or Stopping 8 – Aborted or Aborting The values may be masked in a binary fashion to retrieve a collection containing more than one state. 3 – Held/Holding or Running/Restarting 5 – Stopped/Stopping or Running/Restarting 6 – Stopped/Stopping or Held/Holding 7 – Stopped/Stopping or Held/Holding or Running/Restarting 9 – Aborted/Aborting or Running/Restarting 10 – Aborted/Aborting or Held/Holding 11 – Aborted/Aborting or Held/Holding or Running/Restarting 12 – Aborted/Aborting or Stopped/Stopping 13 – Aborted/Aborting or Stopped/Stopping or Running/Restarting 14 – Aborted/Aborting or Stopped/Stopping or Held/Holding 15 – Aborted/Aborting or Stopped/Stopping or Held/Holding or Running/Restarting

NOTE: *This method only returns the PHASE level (Level 4) recipe steps.*

VBISAlarmListItem Interface

The **VBISAlarmListItem** interface provides access to alarm list data stored in the Batch Execution Server.

Properties

- PhaseID
- PhaseName
- PhaseState
- Mode
- ArbitrationSet
- UnitID
- UnitName
- Owner
- BatchID
- FailureMessage
- PhaseMessage
- ValidUnitList

VBISAlarmListItems Interface

The **VBISAlarmListItems** interface is a collection of **VBISAlarmListItem** objects. The **VBISAlarmListItems** interface provides access to the following lower-level interface:

- VBISAlarmListItem

Properties

- Count
- Item

VBISAlarmsList Interface

IMPORTANT: *VBISAlarmList is provided for backwards compatibility only. For new application development, use the VBISAlarmListItems Interface instead.*

The **VBISAlarmsList** interface provides access to alarm list data stored in the Batch Execution Server. You must instantiate **VBISAlarmsList** from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Alarms list records are stored in safe arrays.

VBISAreaModelHeader Interface

The **VBISAreaModelHeader** interface provides access to audit trail data collected for the current area model.

Properties

- AreaAuditVersion
- AreaAuditPerformedByUserID
- AreaAuditPerformedByName
- AreaAuditPerformedByTime
- AreaAuditPerformedByComment
- AreaAuditVerifiedByUserID
- AreaAuditVerifiedByName
- AreaAuditVerifiedByTime
- AreaAuditVerifiedByComment

VBISAreaModel3 Interface

The **VBISAreaModel3** interface provides access to the equipment defined in the Batch Execution area model. You must instantiate **VBISAreaModel3** from the **VBIS8** object interface.

Properties

- Name
- Revision
- VBISEnumerationSets
- VBISProcessCellClasses
- VBISProcessCells
- VBISUnitClasses
- VBISUnits
- VBISPhaseClasses
- VBISPhases
- VBISTagClasses
- VBISTags
- VBISManifolds
- VBISConnections
- VBISControlModuleClasses
- VBISControlModules
- VBISDataServers
- VBISIconDirectory
- ItemPositions
- ItemIconNames
- IconFromFilenames
- VBISAreaModelHeader

VBISBatchControl5 Interface

The **VBISBatchControl5** batch server control interface provides access 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. You must instantiate **VBISBatchControl5** from the **VBISServer8** object interface.

Properties

- GetParameters
- GetReportParameters
- UnitTags

Methods

- Add
- Bind
- State
- Command
- ReBind
- SetParameter
- AddEvent
- SetUnitTag
- EWIAddEvent
- SecurityAddEvent

VBISBatchList Interface

IMPORTANT: *VBISBatchList is provided for backwards compatibility only. For new application development, use the VBISBatchListItems2 Interface instead.*

The **VBISBatchList** interface provides access to batch list data stored in the Batch Execution Server. You must instantiate **VBISBatchList** from the **VBISServer8** object interface.

Properties

- Count
- Type
- Next

Method

- Query

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Batch list records are stored in safe arrays.

VBISBatchListItem2 Interface

The **VBISBatchListItem2** interface provides access to batch list data stored in the Batch Execution Server.

Properties

- BatchID
- RecipeName
- RecipeVersion
- BatchDescription
- Scale
- StartTime
- ElapsedTime
- Failures
- BatchState
- BatchMode
- CommandMask
- Type

- ParametersRequired
- UnitsRequired
- ParametersSupplied
- UnitsSupplied
- BatchBound
- DefaultBind
- OperatorBindParameters
- OperatorBindUnits
- OperatorInteraction
- ProcessCellList
- PhaseList
- UnitList
- BatchSerialNumber
- RecipeAuditVersion

VBISBatchListItems2 Interface

The **VBISBatchListItems2** interface is a collection of **VBISBatchListItem2** objects. The **VBISBatchListItems2** interface provides access to the following lower-level interface:

- VBISBatchListItem2

Properties

- Count
- Item

VBISBindingPrompt2 Interface

The **VBISBindingPrompt2** interface provides access to binding prompts.

Properties

- Time
- BatchID
- BatchSerialNumber
- Recipe
- Description
- Event
- Value
- EngineeringUnits
- Area
- ProcessCell
- Unit
- Phase
- EventID
- UnitClassName
- DefaultUnit
- StepName
- VBISBindingUnits

Method

- Acknowledge

VBISBindingPrompts2 Interface

The **VBISBindingPrompts2** interface is a collection of **VBISBindingPrompt2** objects. The **VBISBindingPrompts2** interface provides access to the following lower-level interface:

- VBISBindingPrompt2

Properties

- Count
- Item

VBISBindingUnit Interface

The **VBISBindingUnit** interface provides access to the binding unit.

Properties

- Name

VBISBindingUnits Interface

The **VBISBindingUnits** interface is a collection of **VBISBindingUnit** objects. The **VBISBindingUnits** interface provides access to the following lower-level interface:

- VBISBindingUnit

Properties

- Count
- Item

VBISBreakpoint Interface

The **VBISBreakpoint** interface provides access to the transition breakpoints.

Properties

- ID
- BatchID
- BatchSerialNumber
- TransitionID
- Expression
- Recipe

VBISBreakpoints Interface

The **VBISBreakpoints interface** is a collection of **VBISBreakpoint** objects. The **VBISBreakpoints interface** provides access to the following lower-level interface:

- VBISBreakpoint Interface

Properties

- Count
- Item

VBISBreakpointPrompt Interface

The **VBISBreakpointPrompt** interface provides access to the transition breakpoint prompt.

Properties

- ID
- BatchID
- BatchSerialNumber
- BreakpointID
- TransitionID
- Expression
- Recipe

Method

- Acknowledge

VBISBreakpointPrompts Interface

The **VBISBreakpointPrompts** interface is a collection of **VBISBreakpointPrompt** objects. The **VBISBreakpointPrompts** interface provides access to the following lower-level interface:

- VBISBreakpointPrompt Interface

Properties

- Count
- Item

VBISConnection Interface

The **VBISConnection** interface provides access to connections defined in the Batch Execution area model.

Properties

- Name
- Label
- VBISNeededEquipment
- MaxOwners
- EquipmentID
- Source
- SourceType
- Destination
- DestinationType

VBISConnections Interface

The **VBISConnections** interface is a collection of **VBISConnection** objects. The **VBISConnections** interface provides access to the following lower-level interface:

- VBISConnection

Properties

- Count
- Item

VBISControlModule Interface

The **VBISControlModule** interface provides access to control modules defined in the Batch Execution area model.

Properties

- Name
- VBISNeededEquipment

- MaxOwners
- EquipmentID
- ClassName

VBISControlModuleClass Interface

The **VBISControlModuleClass** interface provides access to control module classes defined in the Batch Execution area model.

Properties

- Name
- VBISControlModules

VBISControlModuleClasses Interface

The **VBISControlModuleClasses** interface is a collection of **VBISControlModuleClass** objects. The **VBISControlModuleClasses** interface provides access to the following lower-level interface:

- VBISControlModuleClass

Properties

- Count
- Item

VBISControlModules Interface

The **VBISControlModules** interface is a collection of **VBISControlModule** objects. The **VBISControlModules** interface provides access to the following lower-level interface:

- VBISControlModule

Properties

- Count
- Item

VBISDataServer Interface

The **VBISDataServer** interface provides access to the OPC data servers defined in the Batch Execution area model.

Properties

- Name
- Application
- Topic
- Watchdog
- BadValue
- Type
- AdviseForRequest
- RequestInitialValue
- DefaultServerFlag

VBISDataServers Interface

The **VBISDataServers** interface is a collection of **VBISDataServer** objects. The **VBISDataServers** interface provides access to the following lower-level interface:

- VBISDataServer

Properties

- Count
- Item

VBISEnumeration Interface

The **VBISEnumeration** interface provides access to enumerations defined in the Batch Execution area model.

Properties

- Name
- Ordinal

VBISEnumerations Interface

The **VBISEnumerations** interface provides the list (collection) of enumerations for the parameter (if applicable). You must instantiate **VBISEnumerations** from the **VBISEquipment** object interface.

Properties

- CountEnumSet
- NextEnumSet

Methods

- GetNextEnum
- GetCountEnum
- QueryEnumSet
- QueryEnum
- GetDefaultEnum

VBISEnumerations2 Interface

The **VBISEnumerations2** interface is a collection of **VBISEnumeration** objects. The **VBISEnumerations2** interface provides access to the following lower-level interface:

- VBISEnumeration

Properties

- Count
- Item

VBISEnumerationSet Interface

The **VBISEnumerationSet** interface provides access to enumeration sets defined in the Batch Execution area model.

Properties

- Name
- VBISEnumerations2

VBISEnumerationSets Interface

The **VBISEnumerationSets** interface is a collection of **VBISEnumerationSet** objects. The **VBISEnumerationSets** interface provides access to the following lower-level interface:

- VBISEnumerationSet

Properties

- Count
- Item

VBISEquipment Interface

The **VBISEquipment** interface provides access to the following lower-level object interface:

- VBISEnumerations

You must instantiate **VBISEquipment** from the **VBIS8** or **VBIS** object interface.

Properties

- VBISEnumerations

VBISEWIPromptItem Interface

The **VBISEWIPromptItem** interface provides access to EWI prompts stored in the Batch Execution Server.

Properties

- Time
- BatchID
- BatchSerialNumber
- Recipe
- FileName
- FileVersion
- AreaModel
- ProcessCell
- Unit
- Phase
- EventID

Method

- Acknowledge

VBISEWIPromptItems Interface

The **VBISEWIPromptItems** interface is a collection of **VBISEWIPromptItem** objects. The **VBISEWIPromptItems** interface provides access to the following lower-level interface:

- VBISEWIPromptItem

Properties

- Count
- Item

VBISEWIPrompts Interface

The **VBISEWIPrompts** interface provides access to EWI prompts stored in the Batch Execution Server. You must instantiate VBISEWIPrompts from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query
- Acknowledge

VBISFormulationHeader Interface

The **VBISFormulationHeader** interface provides access to the formulation header defined in the Batch Execution Formulation Editor.

Properties

- FormulationName
- FormulationVersion
- RecipeID
- RecipeVersion

- FormulationType
- FormulationDescription
- FormulationAuthor
- FormulationProductCode
- FormulationBatchSize
- FormulationValid
- MasterRecipeAuditVersion
- FormulationStatus
- FormulationVersionDateUTC
- FormulationVersionDateLocal

VBISIconDirectory Interface

The **VBISIconDirectory** interface provides access to the icon (bitmap) directories in the Batch Execution area model.

Properties

- ProcessCellClass
- UnitClass
- Phase
- Manifold

VBISManifold Interface

The **VBISManifold** interface provides access to manifold objects defined in the Batch Execution area model.

Property

- Name
- IconFilename
- VBISNeededEquipment
- MaxOwners

- EquipmentID
- VBISConnections
- VBISProcessCells

VBISManifolds Interface

The **VBISManifolds** interface is a collection of **VBISManifold** objects. The **VBISManifolds** interface provides access to the following lower-level interface:

- VBISManifold

Properties

- Count
- Item

VBISMessage Interface

The **VBISMessage** interface provides access to operator messages in the Batch Execution area model.

Properties

- Name
- ID
- Log
- External

VBISMessages Interface

The **VBISMessages** interface is a collection of **VBISMessage** objects. The **VBISMessages** interface provides access to the following lower-level interface:

VBISMessage

Properties

- Count
- Item

VBISNeededEquipment Interface

The **VBISNeededEquipment** interface provides accessed to the needed equipment defined in the Batch Execution area model.

Properties

- VBISProcessCells
- VBISUnits
- VBISPhases
- VBISConnections
- VBISControlModules
- VBISManifolds

VBISParameter Interface

The **VBISParameter** interface provides access to the equipment phase parameters defined in the Batch Execution area model.

Properties

- Name
- Type
- ID
- EngineeringUnits
- Scalable
- LowLimit
- HighLimit
- Value
- Enumerations

Remarks

Type can be 1 (real), 2 (long), 3 (string), or 5 (enumeration). If Type is 5 (enumeration), then EngineeringUnits holds the enumeration set.

VBISParameters Interface

The **VBISParameters** interface is a collection of **VBISParameter** objects for the recipe. The **VBISParameters** interface provides access to the following lower-level interface:

- VBISParameter

Properties

- Count
- Item

VBISPhase Interface

The **VBISPhase** interface provides access to equipment phases in the Batch Execution area model.

Properties

- Name
- VBISNeededEquipment
- MaxOwners
- EquipmentID
- ClassName
- VBISReportTags Accesses the VBISTags interface
- VBISParameterTags Accesses the VBISTags interface
- VBISRequestTags Accesses the VBISTags interface
- VBISPhaseReports Accesses the VBISReports interface
- VBISParameters
- VBISUnits
- UnitIDTagName
- CommandTagName
- StatusTagName
- RequestTagName

- FailureTagName
- OwnerTagName
- PauseTagName
- PausedTagName
- SingleStepTagName
- StepIndexTagName

VBISPhase2 Interface

The **VBISPhase2** interface provides access to equipment phases in the Batch Execution area model and run-time information.

Properties

- Name
- VBISNeededEquipment
- MaxOwners
- EquipmentID
- ClassName
- VBISReportTags Accesses the VBISTags interface
- VBISParameterTags Accesses the VBISTags interface
- VBISRequestTags Accesses the VBISTags interface
- VBISPhaseReports Accesses the VBISReports interface
- VBISParameters
- VBISUnits
- UnitIDTagName
- CommandTagName
- StatusTagName
- RequestTagName

- FailureTagName
- OwnerTagName
- PauseTagName
- PausedTagName
- SingleStepTagName
- StepIndexTagName
- State
- Pause
- Mode
- ArbMask
- CmdMask
- UnitID
- CurrentUnit
- Owner
- BatchID
- Failure
- Msg
- Step
- ValidUnits

Methods

- AcquirePhase
- ReleasePhase
- Command
- StartPhase

VBISPhaseClass Interface

The **VBISPhaseClass** interface provides access to equipment phases defined in the Batch Execution area model.

Properties

- Name
- IconFilename
- Type
- NumberOfRequestTags
- NumberOfReportTags
- NumberOfParameterTags
- NumberOfPartners
- VBISParameters
- VBISReports
- VBISMessages
- VBISPhases

VBISPhaseClasses Interface

The **VBISPhaseClasses** interface is a collection of **VBISPhaseClass** objects. The **VBISPhaseClasses interface** provides access to the following lower-level interface:

- VBISPhaseClass

Properties

- Count
- Item

VBISPhaseControl Interface

The **VBISPhaseControl** interface returns phase control interface object. You must instantiate VBISPhaseControl from the **VBISServer8** object interface.

Property

- VBISPhases2

Methods

- AcquirePhase
- ReleasePhase
- Command
- StartPhase

VBISPhases Interface

The **VBISPhases** interface is a collection of **VBISPhase** objects. The **VBISPhases** interface provides access to the following lower-level interface:

- VBISPhase

Properties

- Count
- Item

VBISPhases2 Interface

The **VBISPhases2** interface is a collection of **VBISPhase2** objects. The **VBISPhases2** interface provides access to the following lower-level interface:

- VBISPhase2

Properties

- Count
- Item

Method

FindPhaseFromID

VBISProcessCell Interface

The **VBISProcessCell** interface provides access to process cells in the Batch Execution area model.

Properties

- Name
- ClassName
- EquipmentID
- MaxOwners
- HMIPicture
- VBISNeededEquipment
- VBISUnits
- VBISManifolds
- VBISConnections

VBISProcessCellClass Interface

The **VBISProcessCellClass** interface provides access to process cell classes defined in the Batch Execution area model.

Property

- Name
- IconFilename
- VBISProcessCells

VBISProcessCellClasses Interface

The **VBISProcessCellClasses** interface is a collection of **VBISProcessCellClass** objects defined in the area model. The **VBISProcessCellClasses** interface provides access to the following lower-level interface:

- VBISProcessCellClass

Properties

- Count
- Item

VBISProcessCells Interface

The **VBISProcessCells** interface is the collection of **VBISProcessCell** objects. The **VBISProcessCells** interface provides access to the following lower-level interface:

- VBISProcessCell

Properties

- Count
- Item

VBISPromptList2 Interface

IMPORTANT: *VBISPromptList2 is provided for backwards compatibility only. For new application development, use the VBISPromptListItems Interface instead.*

The **VBISPromptList2** interface provides access to prompt list data stored in the Batch Execution Server. You must instantiate **VBISPromptList2** from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query
- Acknowledge

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Prompt list records are stored in safe arrays.

VBISPromptListItem Interface

The **VBISPromptListItem** interface provides access to operator prompt list data stored in the Batch Execution Server.

Properties

- Time
- BatchID
- Recipe
- Description
- EventType
- Value
- EngineeringUnits
- AreaModel
- ProcessCell
- Unit
- Phase
- EventID
- ResponseType
- High
- Low
- Default

Method

- Acknowledge

VBISPromptListItems Interface

The **VBISPromptListItems** interface is a collection of **VBISPromptListItem** objects. The **VBISPromptListItems** interface provides access to the following lower-level interface:

- VBISPromptListItem

Properties

- Count
- Item

VBISRecipe3 Interface

The **VBISRecipe3** interface provides access to the recipe data stored in the Batch Execution Server. You must instantiate **VBISRecipe3** from the **VBISRecipeManagement3** object interface.

Methods

- ResetControl
- UpdateMaster
- Verify
- RebuildRecipeDir
- AddRecipe
- GetRecipeHeader
- GetProductFormulationHeader
- GetGlobalFormulationHeader

Remarks

To use this interface, your Batch Execution recipes and formulations must be stored in the relational database, meaning the recipe and formulation file types must be set to SQL in the Batch Execution WorkSpace project.

VBISRecipeElements Interface

The **VBISRecipeElements** interface describes the elements of the sequential function chart (SFC).

Properties

- Abstract
- Description
- Identifier
- ProductCode
- VersionNum

- Author
- VersionDate
- Unit
- FontInfo
- RecipeStepInitial
- RecipeStepTerminal
- RecipeStepParent
- VBISRecipeSteps
- VBISRecipeLinks
- VBISRecipeStepTransitions

VBISRecipeHeader2 Interface

The **VBISRecipeHeader2** interface provides access to the recipe header defined in the Batch Execution Recipe Editor.

Properties

- RecipeID
- VersionNumber
- VersionDate
- Author
- ProductCode
- Description
- Abstract
- RecipeType
- Equipment
- Name
- AreaModelValidatedAgainst

- ReleasedToProduction
- ProductName
- BatchSizeMinimum
- BatchSizeMaximum
- BatchSizeUnits
- BatchRunLength
- BatchSizeDefault
- ApprovedBy
- AreaModelFilename
- ValidationTime
- Graphics
- StorageType
- HeaderVersionNumber
- DefaultUnit
- OperatorChangeBindCreate
- OperatorChangeBindExecute
- UnitCapacity
- UnitOfMeasure
- UnitBindMethod
- ScaleCapacity
- RecipeAuditVersion
- RecipeAuditPerformedByUserID
- RecipeAuditPerformedByName
- RecipeAuditPerformedByTime
- RecipeAuditPerformedByComment

- RecipeAuditVerifiedByUserID
- RecipeAuditVerifiedByName
- RecipeAuditVerifiedByTime
- RecipeAuditVerifiedByComment

VBISRecipeLink Interface

The **VBISRecipeLink** interface provides access to the links from steps to transitions, includes Jacobson links, transitions, Or/And Divergences and Convergences, and so on.

Properties

- Type
- Starting Nodes
- Ending Nodes

VBISRecipeLinks Interface

The **VBISRecipeLinks** interface is a collection of **VBISRecipeLink** objects. The VBISRecipeLinks interface provides access to the following lower-level interface:

- VBISRecipeLink

Properties

- Count
- Item

VBISRecipeList3 Interface

The **VBISRecipeList3** batch server recipe list interface provides access to recipe list data stored in the Batch Execution Server. You must instantiate **VBISRecipeList3** from the **VBISServer8** object interface.

Properties

- Count
- Next
- Parameters
- Steps

Methods

- Query
- RecipeCollection

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Recipe list records are stored in safe arrays.

VBISRecipeManagement3 Interface

The **VBISRecipeManagement3** interface allows you to create and maintain recipes. The **VBISRecipeManagement3** interface provides access to the following lower-level object interface:

- VBISRecipe3

You must instantiate **VBISRecipeManagement3** from the **VBIS8** object interface.

Remarks

To use this interface your Batch Execution recipes must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

VBISRecipeStep (Child) Interface

The **VBISRecipeStep (Child)** interface provides access to a child step in a recipe.

Properties

- ID
- XPos
- YPos
- X2Pos
- Y2Pos
- Name
- RecipePath
- Mode

- Control
- State
- Fail
- KeyParamValue
- Index
- CmdMask
- BindType
- UnitClass
- UnitName
- ActUnit
- ReBind
- AcknowledgeBind
- ChildRecipeElements
- RecipeParameterCount
- GetRecipeParameter
- RecipeParameterValueByIndex
- RecipeParameterValueByName
- RecipeParameterEnumerationValues
- RecipeReportCount
- GetRecipeReport

VBISRecipeStepInitial Interface

The **VBISRecipeStepInitial** interface provides access to the starting point of a recipe’s sequential function chart.

Properties

- ID
- XPos
- YPos
- State
- Fail

VBISRecipeStepListItem Interface

The **VBISRecipeStepListItem** interface provides access to the step information for the recipe.

Properties

- ElementID
- S88Type
- StepIndex
- RequestRegister
- OwnerID
- CommandMask
- StepName
- KeyParameterName
- KeyParameterValueEU
- State
- Mode
- UnitName
- ScheduledUnitName
- Owner
- Message

- Failure
- OwnerName

VBISRecipeStepListItems Interface

The **VBISRecipeStepListItems** interface is a collection of **VBISRecipeStepListItem** objects. The **VBISRecipeStepListItems** interface provides access to the following lower-level interface:

- VBISRecipeStepListItem

Properties

- Count
- Item

VBISRecipeStepNode Interface

The **VBISRecipeStepNode** interface provides access to the nodes in a recipe step.

Properties

- ID
- XPos
- YPos

VBISRecipeStep (Parent) Interface

The **VBISRecipeStep (Parent)** interface provides access to a parent step in a recipe. A parent step contains derivative steps in a recipe.

Properties

- ID
- XPos
- YPos
- X2Pos
- Y2Pos
- Name
- RecipePath

- Mode
- Control
- State
- Fail
- KeyParamValue
- Index
- CmdMask
- BindType
- UnitClass
- UnitName
- ActUnit
- ReBind
- AcknowledgeBind
- ChildRecipeElements
- RecipeParameterCount
- GetRecipeParameter
- RecipeParameterValueByIndex
- RecipeParameterValueByName
- RecipeParameterEnumerationValues
- RecipeReportCount
- GetRecipeReport

VBISRecipeSteps Interface

The **VBISRecipeSteps** interface provides access to a collection of children steps that describe the logic of the recipe. The VBISRecipeSteps interface provides access to the following lower-level interface:

VBISRecipeStep

Properties

- Count
- Item
- StepFromID

VBISRecipeStepTerminal Interface

The **VBISRecipeStepTerminal** interface provides access to the ending point of a recipe’s sequential function chart.

Properties

- ID
- XPos
- YPos
- State

VBISRecipeStepTransition Interface

The **VBISRecipeStepTransition** defines when a recipe moves from one step to another in a sequential function chart.

Properties

- ID
- XPos
- YPos
- State
- Fail
- Acquiring
- Condition

VBISRecipeStepTransitions Interface

The **VBISRecipeStepTransitions** interface is a collection of **VBISRecipeStepTransition** objects. The VBISRecipeStepTransitions interface provides access to the following lower-level interface:

- VBISRecipeStepTransition

Properties

- Count
- Item
- FindTransitionFromID

VBISRecipeTransitionExpression Interface

The **VBISRecipeTransitionExpression** interface provides access to the transitions in a recipe.

Property

- RowCount

Method

- GetRowData

VBISRemovedBatchList Interface

The **VBISRemovedBatchList** interface provides the final state of batches that have been removed from the Batch Server.

Properties

- Count
- Item
- Next

Methods

- Query

VBISRemovedBatchListItem Interface

The **VBISRemovedBatchListIem** interface provides access to the items removed from the batch.

Properties

- ID
- State

VBISReport Interface

The **VBISReport** interface provides access to phase reports in the Batch Execution area model.

Properties

- Name
- Type
- ID
- EngineeringUnits
- Log
- External
- Operator

VBISReports Interface

The **VBISReports** interface is a collection of **VBISReport** objects. The **VBISReports** interface provides access to the following lower-level interface:

- VBISReport

Properties

- Count
- Item

VBISServer8 Interface

The **VBISServer8** interface is used to communicate with the Batch Execution server. The **VBISServer8** interface provides access to the following lower-level interfaces:

- VBISBatchControl5
- VBISBatchList
- VBISRecipeList3
- VBISAlarmsList
- VBISPromptList2
- VBISBindingPrompts2
- VBISEWIPromptItems
- VBISBatchListItems2
- VBISAlarmListItems
- VBISPromptListItems
- VBISStepControl2
- VBISPhaseControl
- VBISEWIPrompts
- VBISBreakpoints Interface
- VBISBreakpointPrompts Interface
- VBISRemovedBatchList Interface

You must instantiate **VBISServer8** from the **VBIS8** object interface.

Property

- Status

Methods

- ReConnect
- AuthenticateUser

- SetBreakpoint
- ClearBreakpoint

VBISStep Interface

The **VBISStep** interface provides access to the steps in a Batch Execution recipe.

Properties

- Name
- DefaultUnitName
- VBISUnitClass

VBISStepControl2 Interface

The **VBISStepControl2** interface provides manual phase control to phases. You must instantiate VBISStepControl2 from the **VBISServer8** object interface. The VBISStepControl2 interface provides access to the following lower-level interfaces:

- VBISRecipeElements
- VBISRecipeTransitionExpression
- VBISRecipeStepListItems

Methods

- Command
- StartStep
- HoldStep
- RestartStep
- AbortStep
- StopStep
- ManualStep
- AutoStep
- ClearAllFailures
- VBISActiveRecipeStepListItems

VBISSteps Interface

The **VBISSteps** interface is a collection of **VBISStep** objects. The **VBISSteps** interface provides access to the following lower-level interface:

- VBISStep

Properties

- Count
- Item

VBISTag Interface

The **VBISTag** interface provides access to equipment phase tags defined in the Batch Execution area model.

Properties

- Name
- ClassName
- DataType
- ItemName
- TagType
- VBISDataServer

VBISTagClass Interface

The **VBISTagClass** interface provides access to the equipment phase tags defined in the Batch Execution area model.

Properties

- Name
- DataType
- VBISTags

VBISTagClasses Interface

The **VBISTagClasses** interface is a collection of **VBISTagClass** objects. The **VBISTagClasses** interface provides access to the following lower-level interface:

- VBISTagClass

Properties

- Count
- Item

VBISTags Interface

The **VBISTags** interface is a collection of **VBISTag** objects. The **VBISTags** interface provides access to the following lower-level interface:

- VBISTag

Properties

- Count
- Item

VBISUnit Interface

The **VBISUnit** interface provides access to units defined in the Batch Execution area model.

Properties

- Name
- ClassName
- HMIPicture
- Capacity
- UOM
- ReadyFlag
- DefaultPriority
- EquipmentID
- MaxOwners

- VBISPhases
- VBISTags
- VBISTagClasses
- VBISNeededEquipment

VBISUnitClass Interface

The **VBISUnitClass** interface provides access to unit classes in the Batch Execution area model.

Properties

- Name
- IconFilename
- VBISUnits
- VBISTagClasses
- VBISPhaseClasses
- VBISPhases

VBISUnitClasses Interface

The **VBISUnitClasses** interface is a collection of **VBISUnitClass** objects defined in the area model. The **VBISUnitClasses** interface provides access to the following lower-level interface:

- VBISUnitClass

Properties

- Count
- Item

VBISUnits Interface

The **VBISUnits** interface is a collection of **VBISUnit** objects. The **VBISUnits** interface provides access to the following lower-level interface:

- VBISUnit

Properties

- Count
- Item

VBISUnitTag Interface

The **VBISUnitTags** interface provides access to the unit tags in recipe transitions.

Properties

- Name
- Class
- Type
- Value

VBISUnitTags Interface

The **VBISUnitTags** interface is a collection of **VBISUnitTag** objects. The VBISUnitTags interface provides access to the following lower-level interface:

- VBISUnitTag

Properties

- Count
- Item

Properties

Abstract Property

Returns a detailed description abstract of the recipe, as defined in the recipe header.

Syntax

*object.***Abstract**

The **Abstract** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list

Data Type

BSTR (C++), String (Visual Basic)

Acquiring Property

Returns the acquiring status of the recipe transition.

Syntax

object.**Acquiring**

The **Acquiring** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT_BOOL

ActUnit Property

Returns the active unit of the recipe step.

Syntax

object.**ActUnit**

The **ActUnit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AdviseForRequest Property

If set, the Batch Execution Server substitutes a subscribe for a request.

Syntax

*object.***AdviseforRequest**

The **AdviseforRequest** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Application Property

Since DDE servers are no longer supported in Batch Execution, this property is no longer applicable.
In earlier versions of Batch, this property returned the DDE application name for a DDE server.

Syntax

*object.***Application**

The **Application** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ApprovedBy Property

Returns the name of the person who approved the recipe.

Syntax

*object.***ApprovedBy**

The **ApprovedBy** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ArbitrationSet Property

Returns the arbitration set.

Syntax

*object.***ArbitrationSet**

The **ArbitrationSet** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ArbMask Property

Returns a phase's arbitration mask.

Syntax

*object.***ArbMask**

The **ArbMask** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Area Property

Returns the area in which the binding prompt occurred.

Syntax

*object.***Area**

The **Area** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditPerformedByComment Property

Returns the comment, if any, that the user entered along with the Performed By signature for the specified area model, if a Performed By signature was required.

Syntax

*object.***AreaAuditPerformedByComment**

The **AreaAuditPerformedByComment** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditPerformedByName Property

Returns the full name of the user who entered the Performed By signature for the specified area model, if a Performed By signature was required.

Syntax

*object.***AreaAuditPerformedByName**

The **AreaAuditPerformedByName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditPerformedByTime Property

Returns the time when user entered the Performed By signature for the specified area model, if a Performed By signature was required.

Syntax

*object.***AreaAuditPerformedByTime**

The **AreaAuditPerformedByTime** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditPerformedByUserID Property

Returns the ID of the user who entered the Performed By signature for the specified area model, if a Performed By signature was required.

Syntax

*object.***AreaAuditPerformedByUserID**

The **AreaAuditPerformedByUserID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditVerifiedByComment Property

Returns the comment, if any, that the user entered along with the Verified By signature for the specified area model, if a Verified By signature was required.

Syntax

*object.***AreaAuditVerifiedByComment**

The **AreaAuditVerifiedByComment** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditVerifiedByName Property

Returns the full name of the user who entered the Verified By signature for the specified area model, if a Verified By signature was required.

Syntax

object.**AreaAuditVerifiedByName**

The **AreaAuditVerifiedByName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditVerifiedByTime Property

Returns the time when user entered the Verified By signature for the specified area model, if a Verified By signature was required.

Syntax

object.**AreaAuditVerifiedByTime**

The **AreaAuditVerifiedByTime** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditVerifiedByUserID Property

Returns the ID of the user who entered the Verified By signature for the specified area model, if a Verified By signature was required.

Syntax

*object.***AreaAuditVerifiedByUserID**

The **AreaAuditVerifiedByUserID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaAuditVersion Property

Returns the audit version number of the area model header.

Syntax

*object.***AreaAuditVersion**

The **AreaAuditVersion** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

AreaModel Property

Returns the current area model where the prompt is generated.

Syntax

*object.***AreaModel**

The **AreaModel** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaModelFilename Property

Returns the area model file name associated with the recipe.

Syntax

object.**AreaModelFilename**

The **AreaModelFilename** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

AreaModelValidatedAgainst Property

Returns the name of the area model the recipe was validated against.

Syntax

object.**AreaModelValidatedAgainst**

The **AreaModelValidatedAgainst** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Author Property

Returns the name of the recipe creator, as defined in the recipe header.

Syntax

*object.***Author**

The **Author** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BadValue Property

Returns the string that represents bad values from the data server (for example, @@@@@@).

Syntax

*object.***BadValue**

The **BadValue** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchBound Property

Returns the list of units bound to the batch.

Syntax

*object.***BatchBound**

The **BatchBound** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchDescription Property

Returns the batch description.

Syntax

*object.***BatchDescription**

The **BatchDescription** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchID Property

Returns the batch ID of the prompt or phase.

Syntax

*object.***BatchID**

The **BatchID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Remarks

You cannot use the following characters in the batch ID:

- [(left bracket)
-] (right bracket)
- ((left parenthesis)
-) (right parenthesis)
- , (comma)
- " (double quotes)
- ' (single quotes)
- \n (new line)
- \r (carriage return)
- \t (tab character)
- NULL

BatchMode Property

Returns the batch mode (P-AUTO, O-AUTO, MANUAL).

Syntax

*object.***BatchMode**

The **BatchMode** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchRunLength Property

Returns the expected length the batch is to run, as defined in the recipe header.

Syntax

object.**BatchRunLength**

The **BatchRunLength** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchSerialNumber Property

Returns the internal batch serial number.

Syntax

object.**BatchSerialNumber**

The **BatchSerialNumber** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchSizeDefault Property

Returns the default batch size for the recipe, as defined in the recipe header.

Syntax

*object.***BatchSizeDefault**

The **BatchSizeDefault** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchSizeMaximum Property

Returns the maximum batch size allowed for this recipe, as defined in the recipe header.

Syntax

*object.***BatchSizeMaximum**

The **BatchSizeMaximum** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchSizeMinimum Property

Returns the minimum batch size allowed for this recipe, as defined in the recipe header.

Syntax

*object.***BatchSizeMinimum**

The **BatchSizeMinimum** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchSizeUnits Property

Returns the unit of measure associated with the batch size (for example, gallons, liters).

Syntax

*object.***BatchSizeUnits**

The **BatchSizeUnits** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BatchState Property

Returns the batch state. Possible states include: Aborted, Aborting, Complete, Held, Holding, Idle, Ready, Restarting, Running, Stopping, and Stopped.

Syntax

*object.***BatchState**

The **BatchState** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

BindType Property

Returns the bind type of the recipe step.

Syntax

object.**BindType**

The **BindType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

BreakpointID Property

Returns the ID of the selected breakpoint.

Syntax

object.**BreakpointID**

The **BreakpointID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Capacity Property

Returns the unit's capacity, which is the maximum amount the unit can contain, transfer, or process. During Active Binding, the Batch Execution Server uses this criteria to select units that meet the unit procedure's minimum capacity requirement defined in the recipe.

Syntax

*object.***Capacity**

The **Capacity** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

DOUBLE

Example

If a unit procedure requires a unit capacity of 1000 Liters, the selected unit must have a capacity that is greater than or equal to 1000 Liters.

ChildRecipeElements Property

Returns the steps under this step in the hierarchical view.

Syntax

*object.***ChildRecipeElements**

The **ChildRecipeElements** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VBISRecipeElements

Class Property

Returns unit tag class.

Syntax

*object.***Class**

The **Class** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ClassName Property

Returns the class name of the equipment entity.

Syntax

*object.***ClassName**

The **ClassName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

CmdMask Property

Returns the command mask of the phase.

The numeric code represents the sum of the enabled commands. The following table lists the commands and their corresponding numeric values:

Command Number

CMD_MASK_BIT_ABORT 1

CMD_MASK_BIT_HOLD 2

CMD_MASK_BIT_STOP 4

CMD_MASK_BIT_RESET 8

CMD_MASK_BIT_PAUSE 16

CMD_MASK_BIT_SINGLESTEP 32

CMD_MASK_BIT_DOWNLOAD 64

CMD_MASK_BIT_RESUME 128

CMD_MASK_BIT_RESTART 256

CMD_MASK_BIT_START 512

CMD_MASK_BIT_AUTO 1024

CMD_MASK_BIT_MANUAL 2048

CMD_MASK_BIT_STEP 4096

CMD_MASK_BIT_CLRFAIL 8192

CMD_MASK_BIT_REMOVE 16384

Example

If you enable the Abort, Hold, Stop, and Manual buttons, the command mask value is equal to 2055 (1+2+4+2048).

Syntax

*object.***CmdMask**

The **CmdMask** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

CmdMask Property

Returns the command mask of the recipe step or phase.

The numeric code represents the sum of the enabled commands. The following table lists the commands and their corresponding numeric values:

Command	Number
CMD_MASK_BIT_ABORT	1
CMD_MASK_BIT_HOLD	2
CMD_MASK_BIT_STOP	4
CMD_MASK_BIT_RESET	8
CMD_MASK_BIT_PAUSE	16
CMD_MASK_BIT_SINGLESTEP	32
CMD_MASK_BIT_DOWNLOAD	64
CMD_MASK_BIT_RESUME	128
CMD_MASK_BIT_RESTART	256
CMD_MASK_BIT_START	512
CMD_MASK_BIT_AUTO	1024
CMD_MASK_BIT_MANUAL	2048
CMD_MASK_BIT_STEP	4096
CMD_MASK_BIT_CLRFAIL	8192
CMD_MASK_BIT_REMOVE	16384

Example

If you enable the Abort, Hold, Stop, and Manual buttons, the command mask value is equal to 2055 (1+2+4+2048).

Syntax

*object.***CmdMask**

The **CmdMask** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

CommandMask Property

Returns a numeric code that indicates which commands are available on the toolbar for the currently selected batch.

The numeric code represents the sum of the enabled commands. The following table lists the commands and their corresponding numeric values:

Command	**Number**
CMD_MASK_BIT_ABORT	1
CMD_MASK_BIT_HOLD	2
CMD_MASK_BIT_STOP	4
CMD_MASK_BIT_RESET	8
CMD_MASK_BIT_PAUSE	16
CMD_MASK_BIT_SINGLESTEP	32
CMD_MASK_BIT_DOWNLOAD	64
CMD_MASK_BIT_RESUME	128
CMD_MASK_BIT_RESTART	256
CMD_MASK_BIT_START	512
CMD_MASK_BIT_AUTO	1024
CMD_MASK_BIT_MANUAL	2048
CMD_MASK_BIT_STEP	4096
CMD_MASK_BIT_CLRFAIL	8192
CMD_MASK_BIT_REMOVE	16384

Example

If you enable the Abort, Hold, Stop, and Manual buttons, the command mask value is equal to 2055 (1+2+4+2048).

Syntax

*object.***CommandMask**

The **CommandMask** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

CommandTagName Property

Returns the command tag name for the phase.

Syntax

*object.***CommandTagName**

The **CommandTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Condition Property

Returns the condition of the recipe transition.

Syntax

*object.***Condition**

The **Condition** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Control Property

Returns the control of the recipe step.

Syntax

*object.***Control**

The **Control** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Count Property

Returns the total number of elements within the collection.

Syntax

*object.***Count**

The **Count** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Remarks

For the **VBISBatchList**, **VBISRecipeList3**, **VBISPromptList2**, or **VBISAlarmsList** interfaces, use the **Query** method to initialize the list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the list and won't reflect any changes in the Batch Execution Server until you call **Query**.

***NOTE:** It is recommended that you use the equivalent collection interfaces (VBISBatchListItem2, VBISAlarmListItems, VBISPromptListItems, and VBISEWIPromptItems). These interfaces are designed to support multiple clients.*

CountEnumSet Property

Returns the total number of enumeration sets in the Batch Execution Server's internal list. Use **QueryEnumSet** to initialize the list before you execute **CountEnumSet** or **NextEnumSet**. **CountEnumSet** and **NextEnumSet** will only return information from the list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnumSet**.

Syntax

*object.***CountEnumSet**

The **CountEnumSet** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

CurrentUnit Property

Returns a phase's current unit name.

Syntax

*object.***CurrentUnit**

The **CurrentUnit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

DataType Property

Returns the data type of the object (1 = real, 2 = long, 3 = Boolean, and 4 = string).

Syntax

*object.***DataType**

The **DataType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Default Property

Returns the default value of the prompt.

Syntax

*object.***Default**

The **Default** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

DefaultBind Property

Returns the default binding settings for the batch (0 = No Default, 1 = Unit Binding Defaults, 2 = Parameter Binding Defaults, 3 = Both Parameter and Unit Binding Defaults).

Syntax

*object.***DefaultBind**

The **DefaultBind** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

DefaultPriority Property

Returns the default unit priority for the unit.

Syntax

*object.***DefaultPriority**

The **DefaultPriority** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

DefaultServerFlag Property

Returns 1 if the data server is the Batch Execution data server. The default server is used when tags are defined in the area model.

Syntax

*object.***DefaultServerFlag**

The **DefaultServerFlag** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

DefaultUnit Property

Returns the default unit associated with the object.

Syntax

*object.***DefaultUnit**

The **DefaultUnit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

DefaultUnitName Property

Returns the default unit name for the step.

Syntax

*object.***DefaultUnitName**

The **DefaultUnitName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Description Property

Returns the description associated with the object.

Syntax

object.**Description**

The **Description** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Destination Property

Returns the name of the destination unit or manifold for the connection.

Syntax

object.**Destination**

The **Destination** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

DestinationType Property

Returns the destination type (manifold) for the connection.

Syntax

object.**DestinationType**

The **DestinationType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Remarks

0 corresponds to unit and 1 corresponds to manifold.

ElapsedTime Property

Returns the elapsed time of the currently running batch (HH24:MI:SS).

Syntax

object.**ElapsedTime**

The **ElapsedTime** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ElementID Property

Returns the recipe element ID number.

Syntax

*object.***ElementID**

The **ElementID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

EndingNodes Property

Returns all ending nodes for this recipe link.

Syntax

*object.***EndingNodes**

The **EndingNodes** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT

EngineeringUnits Property

Returns the engineering units associated with the object.

Syntax

*object.***EngineeringUnits**

The **EngineeringUnits** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Equipment Property

Returns 0 if the recipe is class-based or 1 if it is instanced-based. This applies only to unit operations and unit procedures.

Syntax

object.**Equipment**

The **Equipment** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

EquipmentID Property

Returns the equipment ID for the equipment entity.

Syntax

object.**EquipmentID**

The **EquipmentID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Event Property

Returns a string indicating Active Binding.

Syntax

object.**Event**

The **Event** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

EventID Property

Returns the internal event ID of the prompt (needed to acknowledge the prompt).

Syntax

object.**EventID**

The **EventID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

EventType Property

Returns the event type.

Syntax

object.**EventType**

The **EventType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Expression Property

Returns the breakpoint prompt transition expression.

Syntax

object.**Expression**

The **Expression** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

External Property

For internal use only.

Fail Property

Returns the failure status.

Syntax

*object.***Fail**

The **Fail** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR in C++, String in Visual Basic

Failure Property

Returns a batch failure.

Syntax

*object.***Failure**

The **Failure** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FailureMessage Property

Returns the failure message of the alarm. The failure message indicates the reason why the phase went into alarm.

Syntax

*object.***FailureMessage**

The **FailureMessage** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Failures Property

Returns any failure messages from the batch when it is running.

Syntax

*object.***Failures**

The **Failures** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FailureTagName Property

Returns the failure tag name for the phase.

Syntax

*object.***FailureTagName**

The **FailureTagName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FileName Property

Returns the name of the EIB file that is associated with the prompt.

Syntax

*object.***FileName**

The **FileName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FileVersion Property

Returns the version of the EIB file that is associated with the prompt.

Syntax

*object.***FileVersion**

The **FileVersion** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FindTransitionFromID Property

Returns the requested recipe transition.

Syntax

object.**FindTransitionFromID** (*lTransitionID*)

The **FindTransitionFromID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
lTransitionID	The ID of the transition.

Return Data Type

VBISRecipeStepTransition

FontInfo Property

Returns the font information for the recipe element.

Syntax

object.**FontInfo** (*plHeight, plWidth, plEscapement, plOrientation, plWeight, plItalic, plUnderline, plStrikeOut, plCharSet, plOutPrecision, plClipPrecision, plQuality, plPitchAndFamily*)

The **FontInfo** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
plHeight	The height of the font.
plWidth	The width of the font.
plEscapement	The escapement of the font.
plOrientation	The orientation of the font.
plWeight	The weight of the font (whether bold is used).
plItalic	The type of italics used, if any.

Part	**Description**
plUnderline	Describes the underline formatting used, if any.
plStrikeOut	Describes the strikeout formatting used, if any.
plCharSet	Describes the character set of the font.
plOutPrecision	Describes the precision of the font output.
plClipPrecision	Describes the precision of the font clipping.
plQuality	Describes the quality of the font.
plPitchAndFamily	Describes the font point size and family name.
pbstrFontName	The name of the font.

Data Type

BSTR in C++, String in Visual Basic

FormulationAuthor Property

Returns the name of the author of the formulation.

Syntax

*object.***FormulationAuthor**

The **FormulationAuthor** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationBatchSize Property

Returns the batch size for the formulation, as defined in the formulation header.

Syntax

*object.***FormulationBatchSize**

The **FormulationBatchSize** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationDescription Property

Returns the description of the currently running formulation.

Syntax

*object.***FormulationDescription**

The **FormulationDescription** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationName Property

Returns the unique formulation name.

Syntax

*object.***FormulationName**

The **FormulationName** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationProductCode Property

Returns the product code assigned to the formulation, as defined in the formulation header.

Syntax

*object.***FormulationProductCode**

The **FormulationProductCode** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationStatus Property

Returns the status of the formulation (-1=Withdrawn, 0=Draft, 1=Ready, or 2=Approved).

Syntax

*object.***FormulationStatus**

The **FormulationStatus** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

FormulationType Property

Returns the formulation type (0=Product Formulation or 1=Global Formulation).

Syntax

object.**FormulationType**

The **FormulationType** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

FormulationValid Property

Returns whether the formulation is valid (0=FALSE or 1=TRUE).

Syntax

object.**FormulationValid**

The **FormulationValid** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

Boolean (0=FALSE or 1=TRUE)

FormulationVersion Property

Returns the formulation version number.

Syntax

*object.***FormulationVersion**

The **FormulationVersion** property syntax has this part:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

FormulationVersionDateLocal Property

Returns the date of the formulation, in local time, as defined in the formulation header.

Syntax

*object.***FormulationVersionDateLocal**

The **FormulationVersionDateLocal** property syntax has this part:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

FormulationVersionDateUTC Property

Returns the date of the formulation, in UTC time, as defined in the formulation header.

Syntax

*object.***FormulationVersionDateUTC**

The **FormulationVersionDateUTC** property syntax has this part:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

GetParameters Property

Gets the list (collection) of parameters for any procedure.

Syntax

*object.***GetParameters** *(bsProcedurePath)*

The **GetParameters** property syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsProcedurePath	BSTR (C++) String (Visual Basic)	The recipe procedure ID in the form of *batchserialnumber*[\t*recipeunitprocedure]* *[*\t*recipeoperation]* *[*\t*recipephase*] where '\t' is a tab character (i.e. "34 BASE:1 MAKE_BASE:1 ADD_INGS:1")

Return Data Type

VBISParameters: the return collection, which contains the list of parameters for the given procedure and allows the user to enumerate over each of them.

GetRecipeParameter Property

Returns requested recipe parameter of the recipe step.

Syntax

*object.***GetRecipeParameter** (*lIndex, bstrName, bstrEU, bstrHigh, bstrLow, bstrValue, bstrDefault, plRespType, plScope*)

The **GetRecipeParameter** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
lIndex	The parameter ID.
bstrName	The parameter name.
bstrEU	The engineering units defined for the parameter, if any (data type is BSTR in C++ or String in Visual Basic).
bstrHigh	The highest value to which you can set the parameter.
bstrLow	The lowest value to which you can set the parameter.
bstrValue	The value of the parameter.
bstrDefault	The default parameter.
plRespType	The response type of the parameter.
plScope	The scope of the parameter.

Data Type

BSTR (C++), String (Visual Basic)

GetRecipeReport Property

Returns the requested recipe report.

Syntax

*object.***GetRecipeReport** (*lIndex, bstrName, bstrEU,bstrValue, varKeyProcRpt*)

The **GetRecipeReport** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Part	**Description**
lIndex	The phase report ID.
bstrName	The name of the report (data type is BSTR in C++ or String in Visual Basic).
bstrEU	The engineering units defined for the report, if any (data type is BSTR in C++ or String in Visual Basic).
bstrValue	The value of the report parameter.
varKeyProcRpt	Boolean. If the value is defined as a key process report in the Recipe Editor, this parameter is True. Otherwise, it's False.

GetReportParameters Property

Gets the list (collection) of report parameters for any procedure.

Syntax

*object.***GetReportParameters** *(bsProcedurePath)*

The **GetReportParameters** property syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsProcedurePath	BSTR (C++) String (Visual Basic)	The recipe procedure ID in the form of *batchserialnumber*[\t*recipeunitprocedure][*\t*recipeoperation][*\t*recipephase*] where '\t' is a tab character (i.e. "34 BASE:1 MAKE_BASE:1 ADD_INGS:1").

Return Data Type

VBISParameters: the return collection, which contains the list of report parameters for the given procedure and allows the user to enumerate over each of them.

Graphics Property

Returns 1 if graphic coordinates have been saved with the recipe to correctly render the SFC. Returns 0 otherwise.

Syntax

*object.***Graphics**

The **Graphics** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

HeaderVersionNumber Property

Returns the recipe's version number, as defined in the recipe header.

Syntax

*object.***HeaderVersionNumber**

The **HeaderVersionNumber** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

High Property

Returns the high value of the range.

Syntax

*object.***High**

The **High** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

HighLimit Property

Returns the allowable high limit for this parameter.

Syntax

object.**HighLimit**

The **HighLimit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT

HMIPicture Property

Returns the path and file name of HMI picture assigned to the unit or process cell.

Syntax

object.**HMIPicture**

The **HMIPicture** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

IconFilename Property

Returns the name and full path of the icon file.

Syntax

object.**IconFilename**

The **IconFilename** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

IconFromFilenames Property

Returns the icons when given the file names.

Syntax

object.**IconFromFilenames** (*plTypeHint*, *pVarIconFilenames*)

The **IconFromFilenames** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
plTypeHint	The type of file names.
pVarIconFilenames	The file names that are passed in.

Data Type

LONG

ID Property

Returns the unique object ID.

Syntax

object.**ID**

The **ID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

ID Property

Returns the unique object ID.

Syntax

object.**ID**

The **ID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Identifier Property

Returns the identifier for the recipe element.

Syntax

object.**Identifier**

The **Identifier** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Index Property

Returns the index value of the recipe step.

Syntax

*object.***Index**

The **Index** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Item Property

Retrieves the specified object within the collection.

Syntax

*object.***Item** *(varID)*

The **Item** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	**Data Type**	**Description**
varID	VARIANT	Either an integer index specifying the position of the object desired, *or* the Event ID, Enumeration, or Parameter name.

Return Data Type

The specified object.

ItemIconNames Property

Returns the names of the icons associated with the item names.

Syntax

*object.***ItemIconNames** (*plTypeHint, pVarItemNames*)

The **ItemIconNames** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
plTypeHint	The type of the item names.
pVarItemNames	The item names that are passed in.

Data Type

LONG

ItemName Property

Since DDE servers are no longer supported in Batch Execution, this property is no longer applicable.
In earlier versions of Batch, this property returned the OPC item name for the tag.

Syntax

*object.***ItemName**

The **ItemName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ItemPositions Property

Returns the positions of the item names.

Syntax

object.**ItemPositions** (*plTypeHint*, *pVarItemNames*)

The **ItemPositions** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
plTypeHint	The type of the item names.
pVarItemNames	The item names that are passed in.

Data Type

LONG

KeyParameterName Property

Returns the key parameter name of the recipe step.

Syntax

object.**KeyParameterName**

The **KeyParameterName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

KeyParameterValueEU Property

Returns the key parameter value of the recipe step, with engineering units.

Syntax

object.**KeyParameterValueEU**

The **KeyParameterValueEU** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

KeyParamValue Property

Returns the key parameter value of the recipe step.

Syntax

object.**KeyParamValue**

The **KeyParamValue** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Label Property

Returns the label associated with the connection.

Syntax

object.**Label**

The **Label** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Log Property

For internal use only.

Low Property

Returns the low value of the range.

Syntax

object.**Low**

The **Low** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

LowLimit Property

Returns the allowable low limit for this parameter.

Syntax

*object.***LowLimit**

The **LowLimit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT

Manifold Property

Returns the full path name of the manifold bitmap directory.

Syntax

*object.***Manifold**

The **Manifold** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

MasterRecipeAuditVersion Property

Returns the audit version number of the master recipe.

Syntax

*object.***MasterRecipeAuditVersion**

The **MasterRecipeAuditVersion** property syntax has this part:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

MaxOwners Property

Returns the maximum number of owners that can simultaneously own an equipment entity.

Syntax

*object.***MaxOwners**

The **MaxOwners** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Message Property

Returns a step's message string.

Syntax

*object.***Message**

The **Message** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Mode Property

Returns the mode of the recipe step or phase (P-AUTO, O-AUTO, MANUAL).

Syntax

object.**Mode**

The **Mode** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Msg Property

Returns a phase's message string.

Syntax

object.**Msg**

The **Msg** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Name Property

Returns the name of the object.

Syntax

*object.***Name**

The **Name** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Remarks

The recipe name differs from the recipe ID.

Next Property

Returns the next record from the Batch Execution Server's internal list. Batch list, recipe list, prompt list, and alarms list records are stored in safe arrays:

- Batch List Safe Array Values
- Recipe List Safe Array Values
- Prompt List Safe Array Values
- Alarms List Safe Array Values

Syntax

*object.***Next**

The **Next** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Remarks

Use the **Query** method to initialize the list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the list and won't reflect any changes in the Batch Execution Server until you call **Query**.

***NOTE:** It is recommended that you use the equivalent collection interfaces (VBISBatchListItem2, VBISAlarmListItems, VBISPromptListItems, and VBISEWIPromptItems). These interfaces are designed to support multiple clients.*

NextEnumSet Property

Returns the next enumeration set in the Batch Execution Server's internal list. Use **QueryEnumSet** to initialize the list before you call **CountEnumSet** or **NextEnumSet**. **CountEnumSet** and **NextEnumSet** will only return information from the list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnumSet**.

Syntax

object.**NextEnumSet**

The **NextEnumSet** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

NumberOfParameterTags Property

Returns the total number of parameter tags for the equipment phase class.

Syntax

object.**NumberOfParameterTags**

The **NumberOfParameterTags** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

NumberOfPartners Property

Returns the number of message partners for the equipment phase class.

Syntax

*object.***NumberOfPartners**

The **NumberOfPartners** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

NumberOfReportTags Property

Returns the total number of report tags for the equipment phase class.

Syntax

*object.***NumberOfReportTags**

The **NumberOfReportTags** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

NumberOfRequestTags Property

Returns the total number of request tags for the equipment phase class.

Syntax

*object.***NumberOfRequestTags**

The **NumberOfRequestTags** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Operator Property

For internal use only.

OperatorBindParameters Property

Returns a permission flag. This property is no longer used.

Syntax

*object.***OperatorBindParameters**

The **OperatorBindParameters** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

OperatorBindUnits Property

Returns a permission flag. This property is no longer used.

Syntax

*object.***OperatorBindUnits**

The **OperatorBindUnits** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

OperatorChangeBindCreate Property

Returns the flag that specifies if the operator can change the bindings of the recipe when a batch is created.

Syntax

*object.***OperatorChangeBindCreate**

The **OperatorChangeBindCreate** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

OperatorChangeBindExecute Property

Returns the flag that specifies if the operator can change the bindings of the recipe during batch execution.

Syntax

*object.***OperatorChangeBindExecute**

The **OperatorChangeBindExecute** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

OperatorInteraction Property

Returns a permission flag. This property is no longer used.

Syntax

*object.***OperatorInteraction**

The **OperatorInteraction** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Ordinal Property

Returns the index number specifiying in which position the enumeration appears in the enumeration set.

Syntax

*object.***Ordinal**

The **Ordinal** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Owner Property

Returns a step or phase's owner (Batch, Operator, or Manual).

Syntax

*object.***Owner**

The **Owner** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

OwnerID Property

Returns the owner ID for the recipe step.

Syntax

*object.***OwnerID**

The **OwnerID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

OwnerName Property

Returns the owner name for the recipe step.

Syntax

*object.***OwnerName**

The **OwnerName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

OwnerTagName Property

Returns the owner tag name (PHASE_W) for the phase.

Syntax

object.**OwnerTagName**

The **OwnerTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Parameters Property

Returns the collection of parameters for the recipe.

NOTE: *The Parameters and Steps Properties are used to get the necessary scheduling information for a recipe.*

Syntax

object.**Parameters** *(bsRecipeID, bsRecipeVersion)*

The **Parameters** property syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe ID.
bsRecipeVersion	BSTR (C++) String (Visual Basic)	The recipe version.

Return Data Type

VBISParameters

ParametersRequired Property

Returns the required parameters for the batch.

Syntax

*object.***ParametersRequired**

The **ParametersRequired** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ParametersSupplied Property

Returns the parameters supplied when the batch was scheduled.

Syntax

*object.***ParametersSupplied**

The **ParametersSupplied** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Pause Property

Returns phase's pause state.

Syntax

*object.***Pause**

The **Pause** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PausedTagName Property

Returns the paused tag name (PHASE_PD) for the phase.

Syntax

*object.***PausedTagName**

The **PausedTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PauseTagName Property

Returns the pause tag name (PHASE_P) for the phase.

Syntax

*object.***PauseTagName**

The **PauseTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Phase Property

Returns the name of the phase from which the prompt was generated.

Syntax

*object.***Phase**

The **Phase** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Phase Property

Returns the full path name of the phase directory.

Syntax

*object.***Phase**

The **Phase** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PhaseID Property

Returns the equipment ID of the phase that went into alarm.

Syntax

*object.***PhaseID**

The **PhaseID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PhaseList Property

Returns a list of phases currently executing for the batch.

Syntax

*object.***PhaseList**

The **PhaseList** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PhaseMessage Property

Returns the phase message. The phase message identifies a string that is sent to the operator when the phase executes. The message ID must correspond with the ID used by the phase logic.

Syntax

*object.***PhaseMessage**

The **PhaseMessage** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PhaseName Property

Returns the phase name for the equipment phase.

Syntax

*object.***PhaseName**

The **PhaseName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

PhaseState Property

Returns the phase state. Possible states include: Aborted, Aborting, Complete, Held, Holding, Idle, Ready, Restarting, Running, Stopping, and Stopped.

Syntax

object.**PhaseState**

The **PhaseState** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ProcessCell Property

Returns the process cell in which the prompt occurred.

Syntax

object.**ProcessCell**

The **ProcessCell** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ProcessCellClass Property

Returns the full path name of the process cell class directory.

Syntax

*object.***ProcessCellClass**

The **ProcessCellClass** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ProcessCellList Property

Returns a list of process cells that the batch requires in order to execute.

Syntax

*object.***ProcessCellList**

The **ProcessCellList** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ProductCode Property

Returns the product code assigned to the recipe, as defined in the recipe header.

Syntax

*object.***ProductCode**

The **ProductCode** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ProductName Property

Returns the product name associated with the recipe, as defined in the recipe header.

Syntax

*object.***ProductName**

The **ProductName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ReadyFlag Property

Returns the unit ready flag, which determines if your process uses a Unit Ready tag value to determine if the unit is ready for use. Batch Execution checks the value of this tag to determine if this unit can be allocated to a batch. Batch Execution checks the unit ready tag only once, when you schedule the batch.

Syntax

*object.***ReadyFlag**

The **ReadyFlag** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

Example

If the Unit Ready tag value is set to zero (0), the unit is online and Batch Execution can allocate the unit to a batch. If the tag is set to a non-zero value, Batch Execution cannot allocate this unit to a batch.

Recipe Property

Returns the recipe ID of the batch where the prompt occurred.

Syntax

*object.***Recipe**

The **Recipe** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditPerformedByComment Property

Returns the comment, if any, that the user entered along with the Performed By signature for the specified recipe.

Syntax

*object.***RecipeAuditPerformedByComment**

The **RecipeAuditPerformedByComment** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditPerformedByName Property

Returns the full name of the user who entered the Performed By signature for the specified recipe.

Syntax

object.**RecipeAuditPerformedByName**

The **RecipeAuditPerformedByName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditPerformedByTime Property

Returns the time when user entered the Performed By signature for the specified recipe.

Syntax

object.**RecipeAuditPerformedByTime**

The **RecipeAuditPerformedByTime** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditPerformedByUserID Property

Returns the ID of the user who entered the Performed By signature for the specified recipe.

Syntax

*object.***RecipeAuditPerformedByUserID**

The **RecipeAuditPerformedByUserID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditVerifiedByComment Property

Returns the comment, if any, that the user entered along with the Verified By signature for the specified recipe.

Syntax

*object.***RecipeAuditVerifiedByComment**

The **RecipeAuditVerifiedByComment** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditVerifiedByName Property

Returns the full name of the user who entered the Verified By signature for the specified recipe.

Syntax

*object.***RecipeAuditVerifiedByName**

The **RecipeAuditVerifiedByName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditVerifiedByTime Property

Returns the time when user entered the Verified By signature for the specified recipe.

Syntax

*object.***RecipeAuditVerifiedByTime**

The **RecipeAuditVerifiedByTime** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditVerifiedByUserID Property

Returns the ID of the user who entered the Verified By signature for the specified recipe.

Syntax

*object.***RecipeAuditVerifiedByUserID**

The **RecipeAuditVerifiedByUserID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeAuditVersion Property

Returns the audit version number of the recipe.

Syntax

*object.***RecipeAuditVersion**

The **RecipeAuditVersion** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

RecipeID Property

Returns the unique recipe ID.

Syntax

*object.***RecipeID**

The **RecipeID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeName Property

Returns the name of the currently running recipe.

Syntax

object.**RecipeName**

The **RecipeName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeParameterCount Property

Returns the parameter count for the recipe step.

Syntax

object.**RecipeParameterCount**

The **RecipeParameterCount** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

RecipeParameterEnumerationValues Property

Returns the enumeration value for the passed in parameter name.

Syntax

*object.***RecipeParameterEnumerationValues** (*bstrParameterName*)

The **RecipeParameterEnumerationValues** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
bstrParameterName	The name of the recipe parameter (data type is BSTR in C++ or String in Visual Basic).

Return Data Type

BSTR (C++), String (Visual Basic)

RecipeParameterValueByIndex Property

Sets the parameter value of the recipe step via the index passed in.

Syntax

*object.***RecipeParameterValueByIndex** (*lIndex, bstrValue, bsFIXUserName, varSecurityDescriptor*)

The **RecipeParameterValueByIndex** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
lIndex	The index passed in (data type is LONG).
bstrValue	The parameter value (data type is BSTR in C++ or String in Visual Basic).
bsFIXUserName	The name of the user logged in to the iFIX system that is running the batches (data type is BSTR in C++ or String in Visual Basic). If this parameter is NULL then VBIS takes the local iFIX user name.

Part	Description
varSecurityDescriptor	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

RecipeParameterValueByName Property

Sets the parameter value of the recipe step via the name passed in.

Syntax

*object.***RecipeParameterValueByName** (*lName, bstrValue, bsFIXUserName, varSecurityDescriptor*)

The **RecipeParameterValueByName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
lName	The recipe step name (data type is BSTR in C++ or String in Visual Basic).
bstrValue	The parameter value (data type is BSTR in C++ or String in Visual Basic).
bsFIXUserName	The name of the user logged in to the iFIX system that is running the batches (data type is BSTR in C++ or String in Visual Basic). If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

RecipePath Property

Returns the recipe path of the recipe step.

Syntax

*object.***RecipePath**

The **RecipePath** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RecipeReportCount Property

Returns the report count for the recipe step.

Syntax

*object.***RecipeReportCount**

The **RecipeReportCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

RecipeType Property

Returns the recipe type; for example, batch procedure (BP), unit procedure (UP), unit operation (UOP).

Syntax

*object.***RecipeType**

The **RecipeType** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

RecipeVersion Property

Returns the recipe version.

Syntax

object.**RecipeVersion**

The **RecipeVersion** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ReleasedToProduction Property

Returns a flag indicating if the recipe is released to production.

Syntax

object.**ReleasedToProduction**

The **ReleasedToProduction** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

RequestInitialValue Property

Since DDE servers are no longer supported in Batch Execution, this property is no longer applicable.
In earlier versions of Batch, this property returned the request initial value flag (1 if set or 0 if not set)

defined for the DDE server.

Syntax

*object.***RequestInitialValue**

The **RequestInitialValue** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

RequestRegister Property

Returns the value of phase request register for the recipe step.

Syntax

*object.***RequestRegister**

The **RequestRegister** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

RequestTagName Property

Returns the request tag name (PHASE_RQ) for the phase.

Syntax

*object.***RequestTagName**

The **RequestTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ResponseType Property

Returns the data type of the parameter (1 = real, 2 = long, 3 = string, or 5 = enumeration).

Syntax

object.**ResponseType**

The **ResponseType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Revision Property

Returns the revision number of the area model.

Syntax

object.**Revision**

The **Revision** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

RowCount Property

Returns the row count value for the recipe transition expression.

Syntax

*object.***RowCount** (*plCount*)

The **RowCount** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
plCount	The row count value for the recipe transition expression.

Data Type

LONG

S88Type Property

Returns the procedural element of the recipe step.

Syntax

*object.***S88Type**

The **S88Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Scalable Property

Returns a flag (1 if scalable or 0 if not scalable) that determines if the parameter value can be scaled.

Syntax

*object.***Scalable**

The **Scalable** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BOOL

Scale Property

Returns the desired percentage of a batch to be produced.

Syntax

*object.***Scale**

The **Scale** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ScaleCapacity Property

Returns a flag (1 if scalable or 0 if not scalable) indicating if the capacity setting for a unit procedure is scalable.

Syntax

*object.***ScaleCapacity**

The **ScaleCapacity** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

ScheduledUnitName Property

Returns the unit name of the scheduled recipe step.

Syntax

object.**ScheduledUnitName**

The **ScheduledUnitName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

SingleStepTagName Property

Returns the single step tag name (PHASE_SS) for the phase.

Syntax

object.**SingleStepTagName**

The **SingleStepTagName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Source Property

Returns the name of the source (object) unit or manifold for the connection.

Syntax

object.**Source**

The **Source** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

SourceType Property

Returns the source (origin) type (unit or manifold) for the connection.

Syntax

object.**SourceType**

The **SourceType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

StartingNodes Property

Returns all starting nodes for this recipe link.

Syntax

*object.***StartingNodes**

The **StartingNodes** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT

StartTime Property

Returns the start time of the batch (HH24:MI:SS).

Syntax

*object.***StartTime**

The **StartTime** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

State Property

Returns the state of the recipe step or phase.

Syntax

*object.***State**

The **State** property syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Status Property

Returns the status of the Batch Execution server (0 = Good, 1404 = Bad, 1405 = Lost, 1406 = Suspect, 1407 = Unknown Error, and 1408 = Not Found).

Syntax

*object.***Status**

The **Status** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

StatusTagName Property

Returns the status tag name (PHASE_ST) for the equipment phase.

Syntax

*object.***StatusTagName**

The **StatusTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Step Property

Returns a phase's step index.

Syntax

object.**Step**

The **Step** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

StepFromID Property

Returns the requested step ID from the recipe.

Syntax

object.**StepFromID** (*lStepID*)

The **StepFromID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.
lStepID	The ID of the step.

Data Type

LONG

Return Data Type

VBISRecipeStep

StepIndex Property

Returns the step index of the active phase's currently executing step.

Syntax

object.**StepIndex**

The **StepIndex** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

StepIndexTagName Property

Returns the step index tag name (PHASE_SI) for the equipment phase.

Syntax

object.**StepIndexTagName**

The **StepIndexTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

StepName Property

Returns the step name.

Syntax

*object.***StepName**

The **StepName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

StepName Property

Returns the step name associated with the binding prompt.

Syntax

*object.***StepName**

The **StepName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Steps Property

Returns the collection of steps which can be bound. This functionality supports the Unit Binding section of the BatchAdd ActiveX control. The collection of steps matches the list shown in the BatchClient and BatchAdd Control's Unit Binding Spreadsheet that is displayed when creating a batch. For instance based recipes, no steps are returned.

***NOTE:** The Steps and Parameters Properties are used to get the necessary scheduling information for a recipe.*

Syntax

*VBISStepsvar = object.***Steps** *(bsRecipeID, bsRecipeVersion)*

The **Steps** property syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe ID.
bsRecipeVersion	BSTR (C++) String (Visual Basic)	The recipe version.
ppVBISSteps	VBISSteps	Returns a list of steps which may be bound in the recipe. The steps are unit procedures if the recipe is a batch procedure, or the unit operations if the recipe is a unit procedure.

Return Data Type

Returns the unit procedures used for binding, or if it is an operation, it returns the operation name.

For instance based recipes, no steps are returned.

StorageType Property

Returns the storage type associated with the recipe (SQL or FILE).

Syntax

*object.***StorageType**

The **StorageType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

TagType Property

Returns the data server type for the tag (1 = Unknown or 3 = OPC).

Syntax

*object.***TagType**

The **TagType** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Time Property

Returns the time the prompt occurred within the Batch Execution Server.

Syntax

*object.***Time**

The **Time** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Topic Property

Since DDE servers are no longer supported in Batch Execution, this property is empty.

Syntax

*object.***Topic**

The **Topic** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

TransitionID Property

Returns the unique transition ID.

Syntax

*object.***TransitionID**

The **TransitionID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Type Property

Returns the recipe type.

When the batch type is:

- 1 – the batch is a Procedure Batch (created from the BatchList ActiveX control).
- 2 – the batch is a Phase Control Batch (created via the BatchManualPhase ActiveX control).

Syntax

*object.***Type**

The **Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

Type Property (VBISParameter and VBISReport)

Returns the data type of the parameter (1 = real, 2 = long, 3 = string, or 5 = enumeration).

Syntax

*object.***Type**

The **Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Type Property (VBISBatchListItem2 and VBISPhaseClass)

Returns the recipe type.

Syntax

*object.***Type**

The **Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

Type Property

Returns the type of data server (3 = OPC).

Syntax

*object.***Type**

The **Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Type Property (VBISRecipeLink)

Returns the type of link (Link, EquipmentLink, OrDivergence, OrConvergence, AndDivergence, or AndConvergence).

Syntax

*object.***Type**

The **Type** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

ENUMERATION

```
enum{

      Link,

      EquipmentLink,
      OrDivergence,
      OrConvergence,
      AndDivergence,
      And Convergence
}LinkTypes;
```

Unit Property

Returns the phase in which the prompt was generated.

Syntax

*object.***Unit**

The **Unit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Unit Property

Returns the unit for the recipe element.

Syntax

*object.***Unit**

The **Unit** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitBindMethod Property

Returns the bind method defined for the recipe (0 = none, 1 = actual unit [specified at batch creation], 2 = automatic, or 3 = operator).

Syntax

object.**UnitBindMethod**

The **UnitBindMethod** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

UnitCapacity Property

Returns the minimum required unit capacity for a unit procedure recipe. This is compared against the physical unit's capacity during Active Binding.

Syntax

object.**UnitCapacity**

The **UnitCapacity** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

DOUBLE

UnitClass Property

Returns the unit class of the recipe step.

Syntax

*object.***UnitClass**

The **UnitClass** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitClass Property

Returns the full path name of the unit class directory.

Syntax

*object.***UnitClass**

The **UnitClass** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitClassName Property

Returns the unit class name associated with the binding prompt.

Syntax

*object.***UnitClassName**

The **UnitClassName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitID Property

Returns the unit ID for the phase's owner.

Syntax

*object.***UnitID**

The **UnitID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitID Property

Returns the unit ID for the unit from which the phase is executing.

Syntax

*object.***UnitID**

The **UnitID** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitIDTagName Property

Returns the unit ID tag name (PHASE_UN) for the phase.

Syntax

*object.***UnitIDTagName**

The **UnitIDTagName** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitList Property

Returns a list of units.

Syntax

*object.***UnitList**

The **UnitList** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitName Property

Returns the unit name of the recipe step.

Syntax

*object.***UnitName**

The **UnitName** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitOfMeasure Property

Returns the unit of measure associated with the recipe.

Syntax

*object.***UnitOfMeasure**

The **UnitOfMeasure** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitsRequired Property

Returns the required units for the batch.

Syntax

*object.***UnitsRequired**

The **UnitsRequired** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitsSupplied Property

Returns the units supplied when the batch was scheduled.

Syntax

*object.***UnitsSupplied**

The **UnitsSupplied** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

UnitTags Property

Returns a collection of unit tag objects for the given procedure.

Syntax

*object.***UnitTags** *(bsUnitName)*

The **UnitTags** property syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.

bsUnitName	BSTR (C++) String (Visual Basic)	The name of the unit from which you are requesting unit tags.

Return Data Type

VBISUnitTags: the return collection, which contains the list of unit tags for the given recipe procedure.

UOM Property

Returns the units of measure associated with the unit.

Syntax

object.**UOM**

The **UOM** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ValidationTime Property

Returns the time the recipe was validated.

Syntax

object.**ValidationTime**

The **ValidationTime** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

ValidUnitList Property

Returns the valid unit list for a phase.

Syntax

*object.***ValidUnitList**

The **ValidUnitList** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

ValidUnits Property

Returns a phase's valid unit collection.

Syntax

*object.***ValidUnits**

The **ValidUnits** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VBISUnits

Value Property (VBISBindingPrompt2 and VBISPromptListItem)

Returns the current value for the prompt.

Syntax

*object.***Value**

The **Value** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR

Value Property (VBISParameter and VBISUnitTag)

Returns the default value for this parameter.

Syntax

object.**Value**

The **Value** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

VARIANT

VersionDate Property

Returns the date of the recipe, as defined in the recipe header.

Syntax

object.**VersionDate**

The **VersionDate** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

VersionNum Property

Returns the version number, as defined in the recipe header.

Syntax

object.**VersionNumber**

The **VersionNumber** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

VersionNumber Property

Returns the version number, as defined in the recipe header.

Syntax

object.**VersionNumber**

The **VersionNumber** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

Watchdog Property

Returns the name of a time-out field within the data server that is used to ensure that a connection exists between the Batch Execution Server and the process controller.

Syntax

*object.***Watchdog**

The **Watchdog** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

BSTR (C++), String (Visual Basic)

X2Pos Property

Returns the X coordinate position of the recipe step.

Syntax

*object.***X2Pos**

The **X2Pos** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

XPos Property

Returns the X coordinate position.

Syntax

*object.***XPos**

The **XPos** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Y2Pos Property

Returns the Y coordinate position of the recipe step.

Syntax

*object.***Y2Pos**

The **Y2Pos** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

YPos Property

Returns the Y coordinate position.

Syntax

*object.***YPos**

The **YPos** property syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

Data Type

LONG

Methods

The summary below alphabetically lists all of the VBIS methods. Click a method to get more information.

AbortStep Method

Executes an abort step command with the given procedure ID.

Syntax

object.**AbortStep***(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **AbortStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Acknowledge Method

Acknowledges the specified prompt.

Syntax

*object.***Acknowledge** *(bsResponse, bsUserID, varSecurityDescriptor)*

The **Acknowledge** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsResponse	BSTR (C++) String (Visual Basic)	The response to send with the acknowledgment. Responses must be valid for the type of prompt being acknowledged. For example, if the prompt is expecting a number within a specific range, the response must be within that range.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who acknowledged the specified prompt.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Acknowledge Method

Acknowledges the specified prompt.

Syntax

*object.***Acknowledge**

The **Acknowledge** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Acknowledge Method

Acknowledges the specified prompt.

Syntax

*object.***Acknowledge** *(lPromptID, bsResponse, bsUserID, varSecurityDescriptor)*

The **Acknowledge** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lPromptID	BSTR (C++) String (Visual Basic)	The prompt's unique event ID (item 11 in the safe array returned by the PrompList.query).
bsResponse	BSTR (C++) String (Visual Basic)	The response to send with the acknowledgment. Responses must be valid for the type of prompt being acknowledged. For example, if the prompt is expecting a number within a specific range, the response must be within that range.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who acknowledged the specified prompt.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

AcknowledgeBind Method

Acknowledges the binding prompt for the recipe step.

Syntax

*object.***AcknowledgeBind** (*lEventID, bstrUnit, bstrFixUserName, varSecurityDescriptor*)

The **AcknowledgeBind** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lEventID	LONG	The ID of the event.
bsUnit	BSTR (C++) String (Visual Basic)	The unit requesting the acknowledgement.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

AcquirePhase Method

Acquires the phase with the given phase ID.

Syntax

*object.***AcquirePhase** *(lPhaseID, bsFIXUserName, varSecurityDescriptor)*

The **AcquirePhase** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lPhaseID	LONG	The phase ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	Data Type	Description
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

AcquirePhase Method

Acquires the phase.

Syntax

*object.***AcquirePhase** *(bsFIXUserName, varSecurityDescriptor)*

The **AcquirePhase** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Add Method

Adds a batch to the batch list.

Syntax

*BatchSerialNumber = object.***Add** *(bsRecipeId, lRecipeVersion, bsBatchId, lBatchScale, bsUnitBindings, bsParameterBindings, lDefaultBindings, lOpInteraction, lOpBindParameters, lOpBindUnits, bsUserID, varSecurityDescriptor)*

The **Add** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The name of the recipe from which to create the batch.
lRecipeVersion	LONG	The version of the recipe (this is always 1).
bsBatchID	BSTR (C++) String (Visual Basic)	A user-defined batch ID.
lBatchScale	FLOAT	The percentage of the default batch size. Your equipment must support percentages greater than 100 if you specify values greater than 100.
bsUnitBindings	BSTR (C++) String (Visual Basic)	The unit parameters to use in place of the default unit bindings. Specify the unit bindings as tab-delimited string pairs: unit class/unit instance. For example, MIXER\tMIXER1\tMIXER\tMIXER2. ***NOTE:*** *You must specify all unit bindings in this tab-delimited string.* If you use the *bsUnitBindings* parameter for one unit, you will have to specify the *bsUnitBindings* for all units. If you are using the default bindings for all units, you do not have to specify a value for this parameter; it can be left null.

Part	**Data Type**	**Description**
bsParameterBindings	BSTR (C++) String (Visual Basic)	The parameters to use in place of the default parameter bindings. Specify the parameter bindings as tab-delimited string pairs: parameter name/value. For example, FLAVOR\tBUBBLEGUM\tBAKING SODA\t50. If you use the *bsParameterBindings* parameter for one parameter value, you have to use it for all parameter values. If you are using the default parameter bindings for all units, you do not have to specify a value for this parameter; it can be left null.
lDefaultBindings	LONG	0 — use no defaults. You must specify values for *bsUnitBindings* and *bsParameterBindings.* 1 — use the unit defaults. You must specify values for *bsParameterBindings.* 2 — use the parameter defaults. You must specify values for *bsUnitBindings.* 3 — use the defaults. The values for *bsUnitBindings* and *bsParameterBindings* can be left null.
lOpInteraction	LONG	0 (false) or 1 (true). Determines whether an operator using the Batch Execution Client can manipulate a batch scheduled from VBIS.
lOpBindParameters	LONG	0 (false) or 1 (true). Determines whether an operator can bind parameters in the Batch Execution Client for batches scheduled from VBIS.
lOpBindUnits	LONG	0 (false) or 1 (true). Determines whether an operator can bind units in the Batch Execution Client for batches scheduled from VBIS.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who added the batch.

Part	**Data Type**	**Description**
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Return Data Type

LONG. Returns the batch serial number generated by the Batch Execution Server. The batch serial number is a unique ID assigned to the batch when the batch is added to the batch list.

Remarks

The batch serial number is a required parameter for all subsequent VBISBatchControl calls.

When specifying the unit or parameter bindings, you must specify all the unit and parameter bindings or none of them. If you specify a partial list, VBIS assumes the list is complete.

VBIS8 ignores *lOpInteraction, lOpBindParameters, lOpBindUnits*. These parameters are in the code for backward compatibility.

AddEvent Method

Adds an event record to the given batch.

Syntax

*object.***AddEvent** *(bsBatchID, bsDescription, bsValue, bsEngineeringUnit, bsProcessCell, bsUnitName, bsPhaseName, bsFIXUserName, bsUserID, varSecurityDescriptor)*

The **AddEvent** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsBatchID	BSTR (C++) String (Visual Basic)	A user-defined batch ID.

Part	**Data Type**	**Description**
bsDescription	BSTR (C++) String (Visual Basic)	The event description.
bsValue	BSTR (C++) String (Visual Basic)	The value assigned to the event (if any).
bsEngineeringUnit	BSTR (C++) String (Visual Basic)	The Engineering Units assigned to the event (if any).
bsProcessCell	BSTR (C++) String (Visual Basic)	The process cell for the event.
bsUnitName	BSTR (C++) String (Visual Basic)	The unit for the event.
bsPhaseName	BSTR (C++) String (Visual Basic)	The phase name for the event.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who added the event record to the given batch.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

AddRecipe Method

Reads the recipe header information from the database for the given recipe ID and recipe version and adds it to the end of the recipe.dir file. This is useful for creating recipes outside of the Recipe Editor (for example, to copy an existing recipe and change the recipe ID or parameters within the recipe database).

Syntax

*object.***AddRecipe** *(bsRecipeID, lRecipeVersion)*

The **AddRecipe** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe name.
lRecipeVersion	LONG	The internal recipe version (1 for this release).

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

AuthenticateUser Method

Authenticates a user.

Syntax

*object.***AuthenticateUser**(*bstrNTUserName, bstrGroup, bstrPassword*)

The **AuthenticateUser** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	**Data Type**	**Description**
bstrNTUserName	BSTR (C++) String (Visual Basic)	The Windows user name.
bstrGroup	BSTR (C++) String (Visual Basic)	The Windows group from which the system authenticates the user.
bstrPassword	BSTR (C++) String (Visual Basic)	The password for the Windows user.

Return Data

Returns the Windows resolved principal (the user authenticated by Windows security, including the domain, if domains are used by security, and the user name), the Windows full user name, the error code, and error message.

AutoStep Method

Executes an auto step command with the given procedure ID.

Syntax

*object.***AutoStep** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **AutoStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	**Data Type**	**Description**
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Bind Method

Binds units and parameters to a specified batch. The batch must be in the Ready state (130).

Syntax

object.**Bind** *(lBatchSerialNumber, bsUnitBindings, bsParameterBindings, lBindings, bsUserID, varSecurityDescriptor)*

The **Bind** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lBatchSerialNumber	LONG	A unique ID generated by the Batch Execution Server when the batch was added to the batch list.
bsUnitBindings	BSTR (C++) String (Visual Basic)	The unit parameters to use in place of the default unit bindings. Specify the unit bindings as tab-delimited string pairs (unit class/unit instance). For example: MIXER\tMIXER1\tMIXER\tMIXER2 ***NOTE:** You must specify all unit bindings in this tab-delimited string.* If you use the bsUnitBindings parameter for one unit, you will have to specify the bsUnitBindings for all units. If you are using the default bindings for all units, you do not have to specify a value for this parameter; it can be left null.

Part	**Data Type**	**Description**
bsParameterBindings	BSTR (C++) String (Visual Basic)	The parameters to use in place of the default parameter bindings. Specify the parameter bindings as tab-delimited string pairs (parameter name/value). For example: FLAVOR\tBUBBLEGUM\tBAKING SODA\t50. If you use the bsParameterBindings parameter for one parameter value, you have to use it for all parameter values. If you are using the default parameter bindings for all units, you do not have to specify a value for this parameter; it can be left null.
lBindings	LONG	The binding selection: 0 — use no defaults. You must specify values for bsUnitBindings and bsParameterBindings. 1 — use the unit defaults. You must specify values for bsParameterBindings. 2 — use the parameter defaults. You must specify values for bsUnitBindings.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Remarks

When specifying the unit or parameter bindings, you must specify all the unit and parameter bindings or none of them. If you specify a partial list, VBIS assumes the list is complete.

Your application can issue multiple Bind calls; however, only the last one takes effect.

ClearAllFailures Method

Removes all failures.

Syntax

object.**ClearAllFailures** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **ClearAllFailures** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

ClearBreakpoint Method

Removes a transition breakpoints.

Syntax

object.**ClearBreakpoint** *(lBreakpointID)*

The **ClearBreakpoint** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lBreakpointID	LONG	The breakpoint ID.

Command Method (VBISBatchControl5)

Executes the specified batch command. The batch must be in a valid state for the specified command. For example, to execute the START command, the batch must be in the READY (130) state.

Syntax

*object.***Command** *(lBatchSerialNumber, bsBatchCommand, bsUserID, varSecurityDescriptor)*

The **Command** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lBatchSerialNumber	LONG	The unique Batch ID generated by the Batch Execution Server when the batch was added to the batch list.
bsBatchCommand	BSTR (C++) String (Visual Basic)	The command you want to execute against the specified batch. You can execute the following commands: ABORT AUTO CLEARFAILURES HOLD MANUAL REMOVE RESTART START STOP
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who issued the batch command.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Command Method (VBISPhase2)

Executes a phase command.

Syntax

object.**Command** *(bsPhaseCommand, bsFIXUserName, varSecurityDescriptor)*

The **Command** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsPhaseCommand	BSTR (C++) String (Visual Basic)	The actual phase command issued.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Command Method (VBISPhaseControl)

Executes a phase command with the given phase ID.

Syntax

object.**Command** *(lPhaseID, bsUnitName, bsBatchID, bsPhaseCommand, bsFIXUserName, varSecurityDescriptor)*

The **Command** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	Data Type	Description
lPhaseID	LONG	The phase ID.
bsUnitName	BSTR (C++) String (Visual Basic)	The name of the unit for which you want to issue a phase command.
bsBatchID	BSTR (C++) String (Visual Basic)	The ID of the batch for which you want to issue a phase command.
bsPhaseCommand	BSTR (C++) String (Visual Basic)	The actual phase command issued.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Command Method (VBISStepControl2)

Executes a phase command.

Syntax

*object.***Command** *(bstrProcID, bstrStepCommand, bsFIXUserName, varSecurityDescriptor)*

The **Command** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	Data Type	Description
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bstrStepCommand	BSTR (C++) String (Visual Basic)	The name of the step command for the phase.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

EWIAddEvent Method

Adds an EWI batch event to a batch record or to all batch records.

Syntax

*object.***EWIAddEvent** *(bsBatchID, bsEventType, bsEventSubType, bsValue, bsProcessCell, bsUnitName, bsPhaseName, bsUserID, varSecurityDescriptor)*

The **EWIAddEvent** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsBatchID	BSTR (C++) String (Visual Basic)	The name of the batch to which you are adding an EWI event.

Part	**Data Type**	**Description**

bsEventType	enum EWIEVENTTYPE	The event type:
		EWI_INSTRUCTION = 10
		EWI_DATAENTRY = 11
		EWI_SIGNING = 12
		EWI_DEVIATION = 13
		EWI_COMMENTS = 14
		EWI_IMMUTABLE = 15
		EWI_EIB = 16
		EIB_SIGNING = 17
		EIB_DEVIATION = 18
		EIB_COMMENTS = 19
		LOGICSTEP_DEVIATION = 20
		LOGICSTEP_COMMENTS = 21
		LOGICSTEP_EXPRESSION = 22

bsEventSubType	enum EWIEVENTSUBTYPE	The event subtype:
		EWI_EDITBOX = 30
		EWI_CHECKBOX = 31
		EWI_RADIOBUTTON = 32
		EWI_DATETIME = 33
		EWI_DONE = 34
		EWI_DONEBY = 35
		EWI_CHECKBY = 36
		EIB_HEADER = 37
		EIB_ACQUIRE = 38
		EIB_RELEASE = 39
		EWI_EMPTY = 40
		EWI_SKIP = 41
		EWI_REEXECUTE = 42
		EWI_REEXECUTE_CANCEL = 43
		LOGICSTEP_SKIP = 44

bsValue	BSTR (C++) String (Visual Basic)	The value of the EWI event.
bsProcessCell	BSTR (C++) String (Visual Basic)	The process cell for the event.
bsUnitName	BSTR (C++) String (Visual Basic)	The unit for the event.
bsPhaseName	BSTR (C++) String (Visual Basic)	The phase name for the event.

Part	Data Type	Description
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who added the EWI batch event to a batch record or to all batch records..
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

FindPhaseFromID Method

Given a phase ID, the FindPhaseFromID method returns one of the VBIS Phase2 objects in the collection.

Syntax

*object.***FindPhaseFromID** *(lPhaseID)*

The **FindPhaseFromID** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lPhaseID	LONG	The phase ID.

GetCountEnum Method

Returns the total number of enumeration values for the specified enumeration set in the Batch Execution Server's internal list. Use **QueryEnum** to initialize the list before you execute **GetCountEnum** or **GetNextEnumSet**. **GetCountEnum** and **GetNextEnum** will only return information from the list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnum**.

Syntax

*object.***GetCountEnum** *(bsEnumSetName)*

The **GetCountEnum** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsEnumSetName	BSTR (C++) String (Visual Basic)	The enumeration set from which to count enumerations.

Return Data Type

LONG. Returns the number of enumerations within the given enumeration set.

GetDefaultEnum Method

Returns the default enumeration from the specified enumeration set in the Batch Execution Server's internal list.

Syntax

*object.***GetDefaultEnum***(bsEnumSetName)*

The **GetDefaultEnum** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsEnumSetName	BSTR (C++) String (Visual Basic)	The enumeration set from which to retrieve the default enumeration.

Return Data Type

BSTR (C++) or String (Visual Basic). Returns the default enumeration for the given enumeration set.

GetGlobalFormulationHeader Method

Reads the global formulation header information for the given **RecipeID** and returns it to the **VBISFormulationHeader** object.

Syntax

object.**GetGlobalFormulationHeader** *(bsRecipeID, lRecipeVersion, ppVBISFormulationHeader)*

The **GetGlobalFormulationHeader** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The formulation name.
lRecipeVersion	LONG	The recipe version number.
ppVBISFormulationHeader	VBISRFormulationHeader	Returns the object to hold the formulation header information.

Return Data Type

VBISFormulationHeader

Remarks

The specified formulation must be stored in the relational database, meaning the formulation file type must be set to SQL in the Batch Execution WorkSpace project.

GetNextEnum Method

Returns the next enumeration value from the specified enumeration set in the Batch Execution Server's internal list. Use **QueryEnum** to initialize the list before you execute **GetCountEnum** or **GetNextEnum**. **GetCountEnum** and **GetNextEnum** will only return information from the list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnum**.

Syntax

object.**GetNextEnum** *(bsEnumSetName)*

The **GetNextEnum** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsEnumSetName	BSTR (C++) String (Visual Basic)	The enumeration set from which to get the next enumeration record.

Return Data Type

BSTR (C++) or String (Visual Basic). Returns the next enumeration for the given enumeration set.

GetProductFormulationHeader Method

Reads the formulation header information for the given formulation and returns it to the **VBISFormulationHeader** object.

Syntax

object.**GetProductFormulationHeader** *(bsFormulationName, ppVBISFormulationHeader)*

The **GetProductFormulationHeader** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsFormulationName	BSTR (C++) String (Visual Basic)	The formulation name.
ppVBISFormulationHeader	VBISRFormulationHeader	Returns the object to hold the formulation header information.

Return Data Type

VBISFormulationHeader

Remarks

The specified formulation must be stored in the relational database, meaning the formulation file type must be set to SQL in the Batch Execution WorkSpace project.

GetRecipeHeader Method

Reads the recipe header information from the recipe database for the given recipe ID and returns the **VBISRecipeHeader2** object.

Syntax

object.**GetRecipeHeader** *(bsRecipeID, lRecipeVersion, ppVBISRecipeHeader2)*

The **GetRecipeHeader** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe name.
lRecipeVersion	LONG	The recipe version (1 for this release).
ppVBISRecipeHeader2	VBISRecipeHeader2	Returns the object to hold the recipe header information.

Return Data Type

VBISRecipeHeader2

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

GetRowData Method

Returns the row data for the recipe transition expression.

Syntax

object.**GetRowData** *(lRow, bstrLeftExpr, bstrOperator, bstrRightExpr, bstrLeftValue, bstrRightValue, plColor)*

The **GetRowData** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lRow	LONG	Identifies the transition requested.
bstrLeftExpr	BSTR (C++) String (Visual Basic)	The expression on left side of the transition expression.
bstrOperator	BSTR (C++) String (Visual Basic)	The operator in the transition expression.
bstrRightExpr	BSTR (C++) String (Visual Basic)	The expression on right side of the transition expression.
bstrLeftValue	BSTR (C++) String (Visual Basic)	The value on the left side of the transition expression.
bstrRightValue	BSTR (C++) String (Visual Basic)	The value on the right side of the transition expression.
plColor	LONG	The color of the transition expression.

HoldStep Method

Executes a hold step command with the given procedure ID.

Syntax

*object.***HoldStep** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **HoldStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

ManualStep Method

Executes a manual step command with the given procedure ID.

Syntax

*object.***ManualStep***(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **ManualStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	**Data Type**	**Description**
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

Query Method

Updates the batch list, recipe list, alarm list, prompt list, enumeration set list, or enumeration string list, and sets the current index to the first record. Use the **Query** method to initialize these lists before you execute the **Count** and **Next** properties. **Count** and **Next** will only return information from the list and will not reflect any changes in the Batch Execution Server until you execute the **Query** method.

Syntax

object.**Query**

The **Query** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

QueryEnum Method

Sets the current index to the first record for the specified enumeration set. Use **QueryEnum** to initialize the list before you execute **GetCountEnum** or **GetNextEnum**. **GetCountEnum** and **GetNextEnum** will only return information from the list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnum**.

Syntax

object.**QueryEnum** *(bsEnumSetName)*

The **QueryEnumSet** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	**Data Type**	**Description**
bsEnumSetName	BSTR (C++) String (Visual Basic)	The enumeration set to query.

QueryEnumSet Method

Queries the specified enumeration set in the Batch Execution Server's internal list. Use **QueryEnumSet** before any other enumeration list calls to initialize the list. **GetCountEnumSet** and **GetNextEnumSet** will only return information from the enumeration list and won't reflect any changes in the Batch Execution Server until you execute **QueryEnumSet**.

Syntax

object.**QueryEnumSet**

The **QueryEnumSet** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

ReBind Method

Dynamically rebinds units to a specified batch after it has been scheduled.

Syntax

object.**ReBind** *(lBatchSerialNumber, bsUnitBindings, bsUserID, bsFIXUserName, varSecurityDescriptor)*

The **ReBind** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lBatchSerialNumber	LONG	A unique ID generated by the Batch Execution server when the batch was scheduled.

Part	**Data Type**	**Description**
bsUnitBindings	BSTR (C++) String (Visual Basic)	The unit bindings as tab-delimited string pairs: unit class/unit instance. For example: MIXER\tMIXER1\tMIXER\tMIXER2. ***NOTE:*** *You must specify all unit bindings in this tab-delimited string.*
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who issued the rebind command.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

ReBind Method

Rebinds the recipe step to a new unit.

Syntax

*object.***ReBind** *(bstrNewUnit, bsFIXUserName, varSecurityDescriptor)*

The **ReBind** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrNewUnit	BSTR (C++) String (Visual Basic)	The new unit to which you want to rebind the recipe step.

Part	**Data Type**	**Description**
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

RebuildRecipeDir Method

Reads the recipe header information from the SQL database for all recipes and rewrites the recipe.dir file. This is useful if the recipe.dir file gets corrupted or lost.

Syntax

*object.***RebuildRecipeDir**

The **RebuildRecipeDir** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

RecipeCollection Method

Returns a collection of the recipe header information, in XML format.

Syntax

*object.***RecipeCollection**

The **RecipeCollection** method syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

ReConnect Method

Attempts to reconnect to the Batch Execution Server.

Syntax

*object.***ReConnect**

The **ReConnect** method syntax has these parts:

Part	**Description**
object	An object expression that evaluates to an object in the Applies To list.

ReleasePhase Method

Releases the phase with the given phase ID.

Syntax

*object.***ReleasePhase** *(lPhaseID, bsFIXUserName, varSecurityDescriptor)*

The **ReleasePhase** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lPhaseID	LONG	The phase ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	Data Type	Description
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

ReleasePhase Method

Releases the phase.

Syntax

*object.***ReleasePhase** ***(****bsFIXUserName, varSecurityDescriptor****)***

The **ReleasePhase** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

ResetControl Method

Resets the control recipe and step parameter values to the master recipe and step parameter values. The specified recipe must be stored in the relational database.

Syntax

*object.***ResetControl***(bsRecipeID, lRecipeVersion)*

The **ResetControl** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe ID or name to reset.
lRecipeVersion	LONG	The version of the specified recipe. Note: In this version of Batch Execution, the recipe version is always 1.

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

RestartStep Method

Executes a restart step command with the given procedure ID.

Syntax

*object.***RestartStep** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **RestartStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	**Data Type**	**Description**
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

SecurityAddEvent Method

Add a security batch event to a batch record or all batch records.

Syntax

object. **SecurityAddEvent**(*bsBatchID, bsEventType, bsEventSubType, bsValue, bsUserName, varSecurityDescriptor*)

The **SecurityAddEvent** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsBatchID	BSTR (C++) String (Visual Basic)	A user-defined batch ID.
bsEventType	enum SECURITYEVENTTYPE	The event type: SECURITY_SIGNING = 50

bsEventSubType	enum SECURITYEVENTSUBTYPE	The event subtype:
		SECURITY_PERFORMED = 60
		SECURITY_PERFORMEDBY = 61
		SECURITY_VERIFIEDBY = 62

bsValue	BSTR (C++) String (Visual Basic)	The value assigned to the security event.

Part	Data Type	Description
bsUserName	BSTR (C++) String (Visual Basic)	The user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

SetBreakpoint Method

Sets a transition breakpoint.

Syntax

object.**SetBreakpoint** *(bstrBatchSerialNumber, bstrTransitionID)*

The **SetBreakpoint** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrBatchSerialNumber	BSTR (C++) String (Visual Basic)	The serial number of the batch to set the breakpoint for.
bstrTransitionID	BSTR (C++) String (Visual Basic)	The ID number of the transition where the breakpoint is set.

SetParameter Method

Sets a phase parameter to the specified value. The batch must be started in order to set parameters.

Syntax

object.**SetParameter** *(bsPhaseID, bsParameterName, bsValue, bsUserID, varSecurityDescriptor)*

The **SetParameter** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsPhaseID	BSTR (C++) String (Visual Basic)	The ID of the recipe phase that contains the parameter you want to set. Specify the phase ID as a tab-delimited string: *batchserialnumber* \t [*recipeunitprocedure* \t *recipeoperation* \t *recipephase*] Where: • *batchserialnumber* is the serial number assigned to the batch by the Batch Execution Server when the batch was added to the batch list. • *recipeunitprocedure* is the name of the unit procedure. • *recipeoperation* is the name of the operation. • *recipephase* is the name of the phase. • '\t' is a tab character For example: "34\tBASE:1\tMAKE_BASE:1\tADD_INGS:1" ***NOTE:*** *This example shows the required syntax when the parameter value is set at the phase level (the lowest level) of the recipe. If the parameter value is deferred to a higher level of a recipe (such as an operation), refer to the table below for the required syntax.*
bsParameterName	BSTR (C++) String (Visual Basic)	The name of the parameter to set.
bsValue	BSTR (C++) String (Visual Basic)	The parameter value to set.

Part	**Data Type**	**Description**
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who added the batch.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

If the Parameter is Deferred to the…	**The bsPhaseID is…**
Procedure level	*batchserialnumber*
Unit procedure level	*batchserialnumber \t recipeunitprocedure*
Operation level	*batchserialnumber \t recipeunitprocedure \t recipeoperation*

SetUnitTag Method

Sets the value of a unit tag.

Syntax

*object.***SetUnitTag** *(bsUnitName, bsUnitTagName, bsValue, bsUserID, varSecurityDescriptor)*

The **SetUnitTag** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	Data Type	Description
bsUnitName	BSTR (C++) String (Visual Basic)	The name of the unit to which you want to set the unit tag.
bsUnitTagName	BSTR (C++) String (Visual Basic)	The name of the unit tag that you want to set.
bsValue	BSTR (C++) String (Visual Basic)	The value of the unit tag.
bsUserID	BSTR (C++) String (Visual Basic)	The ID of the user who set unit tag value.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

StartPhase Method

Executes a start phase command with the given ID.

Syntax

object.**StartPhase** *(lPhaseID, bsUnitName, bsBatchID, bsFIXUserName, varSecurityDescriptor)*

The **StartPhase** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lPhaseID	LONG	The phase ID.

Part	**Data Type**	**Description**
bsUnitName	BSTR (C++) String (Visual Basic)	The name of the unit for which you want to start a phase.
bsBatchID	BSTR (C++) String (Visual Basic)	The ID of the batch that you want to start a phase for.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

StartPhase Method

Executes a start phase command.

Syntax

object.**StartPhase** *(bsBatchID, bsFIXUserName, varSecurityDescriptor)*

The **StartPhase** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsBatchID	BSTR (C++) String (Visual Basic)	The ID of the batch that you want to start a phase for.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.

Part	Data Type	Description
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

StartStep Method

Executes a start step command with the given procedure ID.

Syntax

*object.***StartStep** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **StartStep** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

State Method

Returns the current batch state for the specified batch. By constantly polling the Batch Execution Server for the current state, your application can determine when a batch completes.

Syntax

object.**State** *(lBatchSerialNumber)*

The **State** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
lBatchSerialNumber	LONG	The unique Batch ID generated by the Batch Execution Server when the batch was added to the batch list.

Data Type

LONG. The following are the possible states and their associated return values:

ABORTING — 10
HOLDING — 20
STOPPING — 30
RESTARTING — 40
RUNNING — 50
HELD — 60
COMPLETE — 70
STOPPED — 80
ABORTED — 90
IDLE — 100
STARTING — 110
NOT CONNECTED — 120
READY —130

StopStep Method

Executes a stop step command with the given procedure ID.

Syntax

object.**StopStep** *(bstrProcID, bsFIXUserName, varSecurityDescriptor)*

The **StopStep** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.

Part	Data Type	Description
bstrProcID	BSTR (C++) String (Visual Basic)	The procedure or unit procedure recipe ID.
bsFIXUserName	BSTR (C++) String (Visual Basic)	The name of the user logged in to the iFIX system that is running the batches. If this parameter is NULL then VBIS takes the local iFIX user name.
varSecurityDescriptor	VARIANT	For future use only. You can ignore this optional field in Visual Basic but with C++ you must pass in a VARIANT, even though you are not using the varSecurityDescriptor field.

UpdateMaster Method

Updates the master recipe and step parameter values to the control recipe and step parameter values. The specified recipes must be stored in the relational database.

Syntax

*object.***UpdateMaster***(bsRecipeID, lRecipeVersion)*

The **UpdateMaster** method syntax has these parts:

Part	Data Type	Description
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe ID or name to update.
lRecipeVersion	LONG	The version of the specified recipe. Note: In this version of Batch Execution, the recipe version is always 1.

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

Verify Method

Verifies the contents of the specified recipe to ensure that the recipe is valid. The specified recipe must be stored in the relational database.

Syntax

*object.***Verify***(bsRecipeID, lRecipeVersion)*

The **Verify** method syntax has these parts:

Part	**Data Type**	**Description**
object	not applicable	An object expression that evaluates to an object in the Applies To list.
bsRecipeID	BSTR (C++) String (Visual Basic)	The recipe ID or name to verify.
lRecipeVersion	LONG	The version of the specified recipe. Note: In this version of Batch Execution, the recipe version is always 1.

Remarks

The specified recipe must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

Safe Arrays

The following safe array values are used with Proficy Batch Execution:

- Alarms List Safe Array Values
- Batch List Safe Array Values
- Prompt List Safe Array Values
- Recipe List Safe Array Values

Alarms List Safe Array Values

The following are the safe array values returned from **VBISAlarmsList.Next**. The data types for these values are VT_BSTR.

Safe Array	**Returned Value**
Safearray[0]	Phase ID
Safearray[1]	Phase Name
Safearray[2]	Phase State
Safearray[3]	Mode
Safearray[4]	Arbitration Set
Safearray[5]	Unit ID
Safearray[6]	Unit Name
Safearray[7]	Owner
Safearray[8]	Batch ID
Safearray[9]	Fail Message
Safearray[10]	Phase Message
Safearray[11]	Valid Unit List

Batch List Safe Array Values

The following are the safe array values returned from **VBISBatchList.Next**. The data types for these values are VT_BSTR.

Safe Array	**Returned Value**
Safearray[0]	User defined batch ID.
Safearray[1]	Recipe's name.

Safe Array	**Returned Value**
Safearray[2]	Recipe’s version.
Safearray[3]	Batch’s description.
Safearray[4]	Batch’s scale.
Safearray[5]	Batch’s start time.
Safearray[6]	Batch’s elapsed time.
Safearray[7]	Batch’s highest priority failure message.
Safearray[8]	Batch’s state.
Safearray[9]	Batch’s mode.
Safearray[10]	Batch’s type. Batch type 1 – Procedure Batch (created from the BatchList ActiveX control) Batch type 2 – Phase Control Batch (created via the BatchManualPhase ActiveX control).
Safearray[11]	Batch’s required parameters.
Safearray[12]	Batch’s required units.
Safearray[13]	Batch’s supplied parameters.
Safearray[14]	Batch’s supplied units.
Safearray[15]	Batch’s binding.
Safearray[16]	Batch’s default binding.
Safearray[17]	Batch’s operation bind parameters.
Safearray[18]	Batch’s operation bind units.
Safearray[19]	Batch’s operation interaction.

Safe Array	**Returned Value**
Safearray[20]	Process cell on which the batch is running.
Safearray[21]	Active phases for the batch.
Safearray[22]	Unit binding per unit procedure. If there are multiple unit procedures, multiple units are listed. The units listed reflect units previously bound/owned, currently bound/owned, and scheduled.
Safearray[23]	Batch’s serial number.
Safearray[24]	Command mask, which is a series of bits that indicate the valid commands for this batch.
Safearray[25]	Recipe audit version number.

Prompt List Safe Array Values

The following are the safe array values returned from **VBISPromptList2.Next**. The data types for these values are VT_BSTR.

Safe Array	**Returned Value**
Safearray[0]	Prompt’s time.
Safearray[1]	Prompt’s batch ID
Safearray[2]	Recipe’s name.
Safearray[3]	Prompt’s description
Safearray[4]	Event type.
Safearray[5]	Event value.
Safearray[6]	Engineering units (EGU)
Safearray[7]	Area model (equipment database)
Safearray[8]	Process cell where the event occurred.

Safearray[9]	Unit on which the event occurred.
Safearray[10]	Phase on which the event occurred.
Safearray[11]	Unique event ID
Safearray[12]	Response's data type
Safearray[13]	Maximum value allowed
Safearray[14]	Minimum value allowed
Safearray[15]	Default value

Recipe List Safe Array Values

The following are the safe array values returned from **VBISRecipeList3.Next**. The data types for these values are VT_BSTR.

Safe Array	**Returned Value**
Safearray[0]	Recipe's ID; the unique recipe name, for example: Make_Toothpaste.
Safearray[1]	Recipe's description.
Safearray[2]	Recipe's type.
Safearray[3]	Recipe's product ID
Safearray[4]	Recipe's author
Safearray[5]	Recipe's version.
Safearray[6]	Recipe's time stamp.
Safearray[7]	Recipe's file name.
Safearray[8]	Recipe's storage type (file or SQL)
Safearray[9]	Release to production flag

Safe Array	**Returned Value**
Safearray[10]	Product name.
Safearray[11]	Recipe audit version number.
Safearray[12]	Microsoft Windows user ID of the operator (from the Performed By group) who last authorized the saving of the recipe.
Safearray[13]	Windows full user name of the operator (in the Performed By group) who last authorized the saving of the area model before the Server Manager started.
Safearray[14]	Comments, if any, entered by the operator (from the Performed By group) who last authorized the saving of the recipe.
Safearray[15]	The date and time when Batch Execution authenticated the electronic signature of the operator (from the Performed By group).
Safearray[16]	Microsoft Windows user ID of the supervisor (from the Verified By group) who last authorized the saving of the recipe.
Safearray[17]	Windows full user name of the supervisor (in the Verified By group) who last authorized the saving of the area model before the Server Manager started.
Safearray[18]	Comments, if any, entered by the supervisor (from the Verified By group) who last authorized the saving of the recipe.
Safearray[19]	The date and time when Batch Execution authenticated the electronic signature of the supervisor (from the Verified By group).

Success & Error Codes

Batch Execution error codes are listed in the following table.

Error Code	Meaning
0	VBIS_SUCCESS
2	VBIS_INIT_COMPLETE
6	VBIS_CLEANUP_COMPLETE
1001	VBIS_ERROR
1003	VBIS_FAILED_TO_INITIALIZE
1004	VBIS_FAILED_TO_CONNECT
1005	VBIS_CLEANUP_FAILED
1007	VBIS_BAD_PTR
1008	VBIS_NO_RECIPE
1009	VBIS_INVALID_VERSION
1010	VBIS_NO_BATCH
1011	VBIS_BAD_STATE
1012	VBIS_OUT_OF_MEMORY
1013	VBIS_BAD_VAR_TYPE
1014	VBIS_SUB_OUT_OF_RANGE

Error Code	**Meaning**
1015	VBIS_BAD_ARG
1202	VBIS_SS_BAD_UNIT_BIND
1203	VBIS_SS_BAD_PARM_BIND
1204	VBIS_SS_NO_BIND_UP
1205	VBIS_SS_NO_BIND_UNIT
1206	VBIS_SS_NO_BIND_PARM
1207	VBIS_SS_UP_BIND
1208	VBIS_SS_UNIT_BIND
1209	VBIS_SS_PARM_BIND
1210	VBIS_SS_INVALID_FLAG
1211	VBIS_SS_SCALE_OUT_OF_RANGE
212	VBIS_SS_BATCH_BOUND
1213	VBIS_SS_MISMATCH_BIND
1400	VBIS_BS_BAD_COMMAND
1401	VBIS_BS_NO_UP_BIND
1402	VBIS_BS_NO_UNIT_BIND
1403	VBIS_BS_NO_PARM_BIND

Error Code	**Meaning**
1600	VBIS_PS_NO_PROMPT

VBIS_SUCCESS (0)

The object interface call that your application issued was successfully completed.

If the call was...	**Then...**
VBISAlarmsList.Next	The Batch Execution Server retrieved the next alarm in the internal alarm list.
VBISAlarmsList.Count	The Batch Execution Server returned the number of alarms in the internal alarm list.
VBISAlarmsList.Query	The Batch Execution Server updated the internal alarm list and set the current index to the first record.
VBISBatchControl5.State	The Batch Execution Server returned the state of the specified batch. The possible batch states and their return values are: • Aborting — 10 • Holding — 20 • Stopping — 30 • Restarting — 40 • Running — 50 • Held — 60 • Complete — 70 • Stopped — 80 • Aborted — 90 • Idle — 100 • Starting — 110 • Not Connected — 120 • Ready — 130

If the call was...	Then...
VBISBatchControl5.Add	The Batch Execution Server scheduled the batch and placed it in the Ready state. No unit or parameter binding is required or the necessary bindings were supplied. The Batch Execution Server also returns the batch ID of the scheduled batch.
VBISBatchControl5.Bind	The Batch Execution Server successfully bound the specified units and parameters and placed the associated batch in a Ready state.
VBISBatchControl5.Command	The Batch Execution Server executed the specified batch command.
VBISBatchList.Next	The Batch Execution Server retrieved the next batch from the internal batch list.
VBISBatchList.Count	The Batch Execution Server returned the number of batches in the internal batch list.
VBISBatchList.Query	The Batch Execution Server updated the internal batch list and set the current index the first record.
VBISEnumerations.GetCountEnum	The Batch Execution Server returned the number of enumerations within the specified enumeration set.
VBISEnumerations.CountEnumSet	The Batch Execution Server returned the number of enumeration sets in the internal enumeration set list.
VBISEnumerations.GetDefaultEnum	The Batch Execution Server retrieved the default enumeration from the specified enumeration set.
VBISEnumerations.GetNextEnum	The Batch Execution Server retrieved the next enumeration from the specified enumeration set.
VBISEnumerations.NextEnumSet	The Batch Execution Server retrieved the next enumeration set from the internal enumeration set list.
VBISEnumerations.QueryEnum	The Batch Execution Server set the current index to the first record.
VBISEnumerations.QueryEnumSet	The Batch Execution Server set the current index to the first record.

If the call was...	**Then...**
VBISPromptList2.Acknowledge	The Batch Execution Server acknowledged the specified prompt.
VBISPromptList2.Next	The Batch Execution Server retrieved the next prompt in the internal prompt list.
VBISPromptList2.Count	The Batch Execution Server returned the number of prompts from the internal prompt list.
VBISPromptList2.Query	The Batch Execution Server updated the internal prompt list and set the current index to the first record.
VBISRecipeList3.Count	The Batch Execution Server retrieved the number of recipes from internal recipe list.
VBISRecipeList3.Next	The Batch Execution Server retrieved the next recipe from the internal recipe list.
VBISRecipeList3.Query	The Batch Execution Server updated the internal recipe list and set the current index to the first record.
VBISRecipe3.ResetControl	The specified control recipe has been reset to use the values from the associated master recipe.
VBISRecipe3.UpdateMaster	The specified master recipe has been updated to use the values from the associated control recipe.
VBISRecipe3.Verify	The Batch Execution server verified the specified recipe.

VBIS_INIT_COMPLETE (2)

VBIS completed initialization.

Note: *Visual Basic (On Error) does not report this as an error.*

VBIS_CLEANUP_COMPLETE (6)

VBIS completed the release of the internal objects created by your application.

Note: *Visual Basic (On Error) does not report this as an error.*

VBIS_ERROR (1001)

The Batch Execution Server could not execute one of the commands your application issued. When this happens, the Batch Execution Server records the error in its error log, VBEXEC.LOG. This file resides in the LOGS directory of your project.

Try this

Examine the error log for your application to determine which command failed to execute and verify all the information pertaining to that command. For example, make sure that each parameter in the command is supported by your process equipment. Also make sure that the batch affected actually exists.

Applies To

All of the interface calls can return this error.

VBIS_FAILED_TO_INITIALIZE (1003)

VBIS could not initialize itself.

VBIS_FAILED_TO_CONNECT (1004)

VBIS could not connect to the Batch Execution Server.

Try this

Make sure the Batch Execution Server is running and your computer is connected to the network.

VBIS_CLEANUP_FAILED (1005)

VBIS could not release the internal objects created by your application.

Note: *Visual Basic (On Error) does not report this as an error.*

VBIS_BAD_PTR (1007)

A pointer used by your application is no longer valid.

Try this

Re-run your application. If you receive the error again, verify each pointer used by your program.

VBIS_NO_RECIPE (1008)

An invalid recipe name was specified.

Try this

Verify the recipe name your application is passing is correct for the batch and run your application again.

VBIS_INVALID_VERSION (1009)

An invalid recipe version was specified.

Try this

Verify that the recipe version your application is passing is correct for the batch and run your application again.

VBIS_NO_BATCH (1010)

An incorrect batch ID was specified or the specified batch is not in the current batch list.

Try this

Verify that the batch ID is correct and try to run your application again. If the application is requesting the state of the batch, make sure the batch is running and has not been removed from the batch list.

If your application is adding a batch to the batch list, verify that the batch ID does not contain invalid characters such as commas (,) brackets, ([ ]), quotation marks (“ ‘), parentheses(( )), tabs, carriage returns, or line feeds.

VBIS_BAD_STATE (1011)

The batch is not in the proper state to complete the requested call.

Try this

Verify that the batch is in the correct state, as shown below, and try to run your application again.

To use this call...	**The batch state must be...**
VBISBatchControl5.Bind	READY (130)
VBISBatchControl5.Command	READY (130), STOPPED (80), COMPLETE (70), or ABORTED (90)

VBIS_OUT_OF_MEMORY (1012)

There was not enough system memory to allocate an essential object.

Try this

Shut down unnecessary applications to free up memory.

Applies To

All the interface calls can return this error.

VBIS_BAD_VAR_TYPE (1013)

One or more of the parameter VARIANTs passed in were of the wrong data type. VARIANTs that are passed in to receive information are not checked for the data type that is required by the call.

Try this

Verify that the data type of the VARIANT parameters are the correct data type.

Applies To

All calls with input parameters can generate this error.

VBIS_SUB_OUT_OF_RANGE (1014)

The program has called the **Next** property too many times causing the program to go past the end of the list.

Try this

Use the **Query** method to set the current index to the first record.

VBIS_BAD_ARG (1015)

One or more of the input parameters were missing or invalid.

Try this

Verify that all of the input parameters sent by the call contain data.

VBIS_SS_BAD_UNIT_BIND (1202)

The unit binding parameter being passed exceeds the maximum number of characters, does not contain the minimum number of characters, is missing, or is the wrong type (for example, specifying a string when an integer is expected).

Try this

Verify the value of each parameter binding parameter and run your application again.

VBIS_SS_BAD_PARM_BIND (1203)

The binding parameter being passed exceeds the maximum number of characters, does not contain the minimum number of characters, is missing, or is the wrong type (for example, specifying a string when an integer is expected).

Try this

Verify the value of each binding parameter and run your application again.

VBIS_SS_NO_BIND_UP (1204)

Unit and parameter bindings were specified when they were not required.

Try this

Do not specify unit and parameter bindings with **VBISBatchControl5**.

VBIS_SS_NO_BIND_UNIT (1205)

A unit binding was specified when it was not required.

Try this

Do not specify unit bindings with **VBISBatchControl5.Bind**.

VBIS_SS_NO_BIND_PARM (1206)

A parameter binding was specified when it was not required.

Try this

Do not specify parameter bindings with **VBISBatchControl5.Bind**.

VBIS_SS_UP_BIND (1207)

The Batch Execution Server added the batch to the batch list and placed it in the Ready state. Your application must now bind both units and parameters.

Try this

Use VBISBatchControl5.Bind to bind the required units and parameters.

VBIS_SS_UNIT_BIND (1208)

The Batch Execution Server scheduled the batch and placed it in the Ready state. Your application must now bind the units required.

Try this

Use VBISBatchControl5.Bind to bind the units.

VBIS_SS_PARM_BIND (1209)

The Batch Execution Server added the batch to the batch list and placed it in the Ready state. Your application must now bind one or more parameters.

Try this

Use VBISBatchControl5.Bind to bind the parameters.

VBIS_SS_INVALID_FLAG (1210)

One of the parameters being passed exceeds the maximum number of characters, does not contain the minimum number of characters, is missing, or is the wrong type (for example, specifying a string when an integer is expected).

Try this

Verify each parameter’s value and run your application again.

VBIS_SS_SCALE_OUT_OF_RANGE (1211)

The specified scaling factor is greater than the maximum value supported by your equipment or is less than the minimum value supported by your equipment. The scaling factor could be missing when a value was expected or is the wrong type (for example, specifying a string when an integer is expected).

Try this

Verify the scaling value and run your application again.

VBIS_SS_BATCH_BOUND (1212)

Your application attempted to bind units for a batch that was scheduled by the Batch Execution Client application. Batches scheduled by Batch Execution Client are bound when the batch is scheduled and cannot be rebound.

Try this

Verify the batch ID your application passed and run your application again.

VBIS_SS_MISMATCH_BIND (1213)

The binding data supplied does not match the binding flag or the binding data was not supplied.

Try this

Specify the correct binding information and try to bind the units and parameters again.

VBIS_BS_BAD_COMMAND (1400)

Your application issued a command that is not supported by **VBISBatchControl5.Command**.

Try this

Verify that **VBISBatchControl5.Command** is not issuing any unsupported batch commands and run your application again.

VBIS_BS_NO_UP_BIND (1401)

An application issued the Start command with **VBISBatchControl5.Command** before binding the required units and parameters.

Try this

Use VBISBatchControl5.Bind to bind the units and parameters required.

VBIS_BS_NO_UNIT_BIND (1402)

An application issued the Start command with **VBISBatchControl5.Command** before binding the required units.

Try this

Use VBISBatchControl5.Bind to bind the units.

VBIS_BS_NO_PARM_BIND (1403)

An application issued the Start command with **VBISBatchControl5.Command** before binding the required parameters.

Try this

Use VBISBatchControl5.Bind to bind the parameters.

VBIS_PS_NO_PROMPT (1600)

No prompt with the specified ID was found.

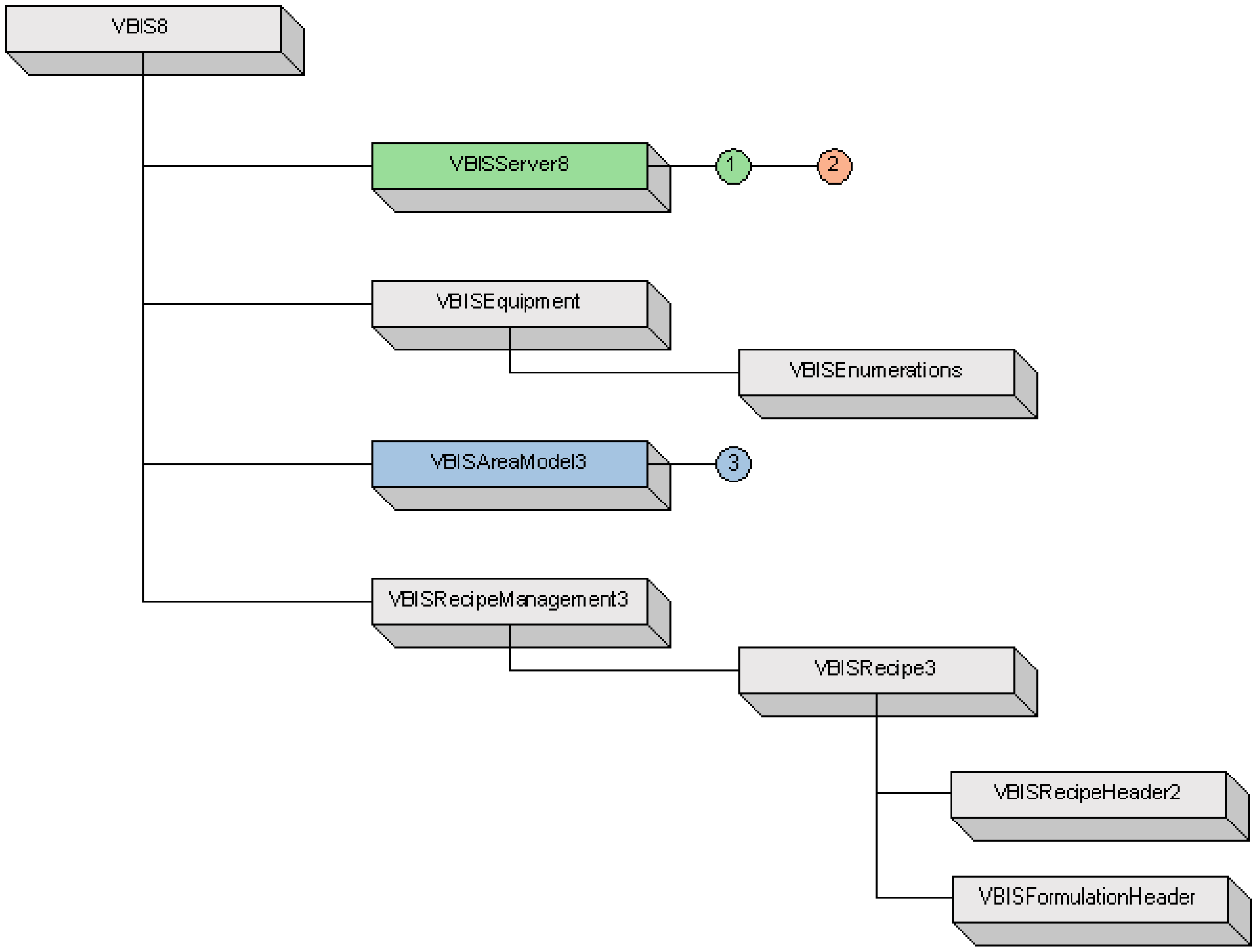
Try this

Verify the prompt ID specified and resend the command with a valid prompt ID.

VBIS8 Automation Interface Hierarchy

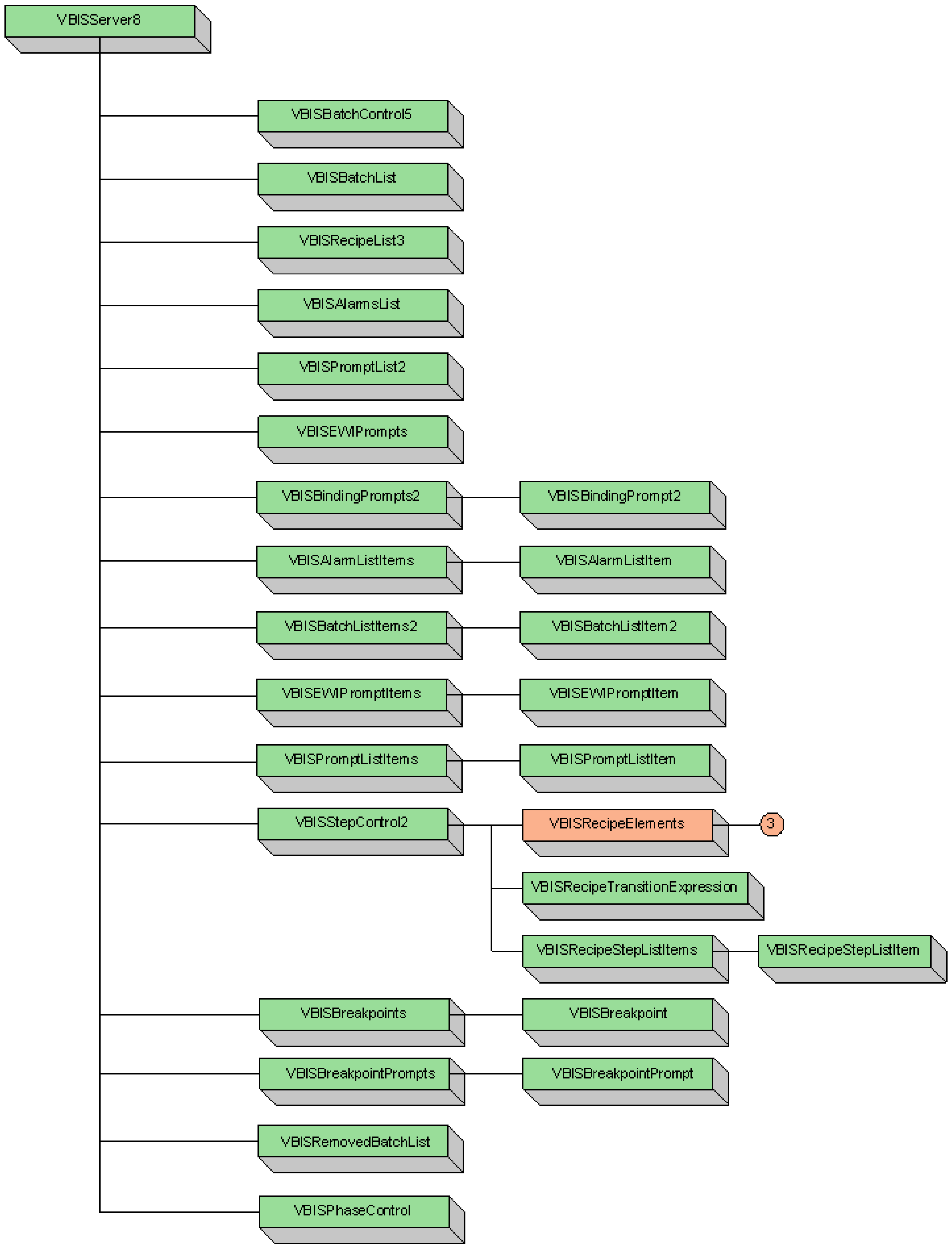
Understanding the VBIS8 Automation Interface Hierarchy

The graphic below shows the **VBIS8** automation interface hierarchy. To get more information on each object, click the object name in the graphic.



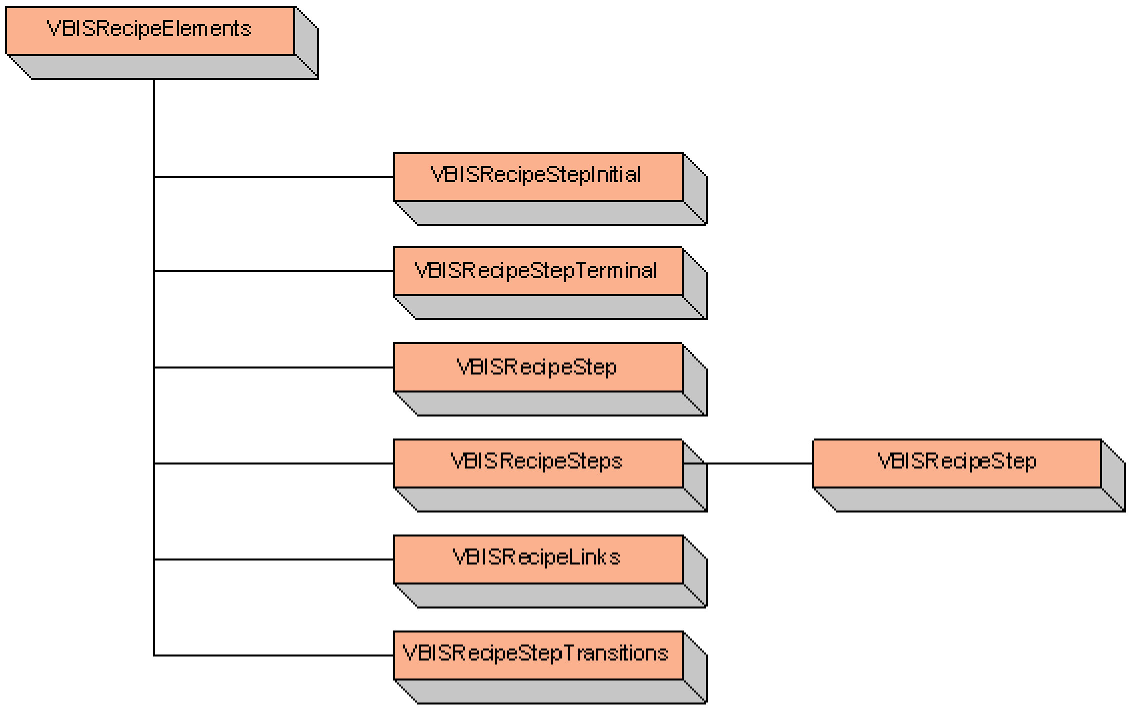
Understanding the VBISServer8 Hierarchy

The graphic below shows the **VBISServer8** hierarchy. To get more information on each object, click the object name in the graphic.



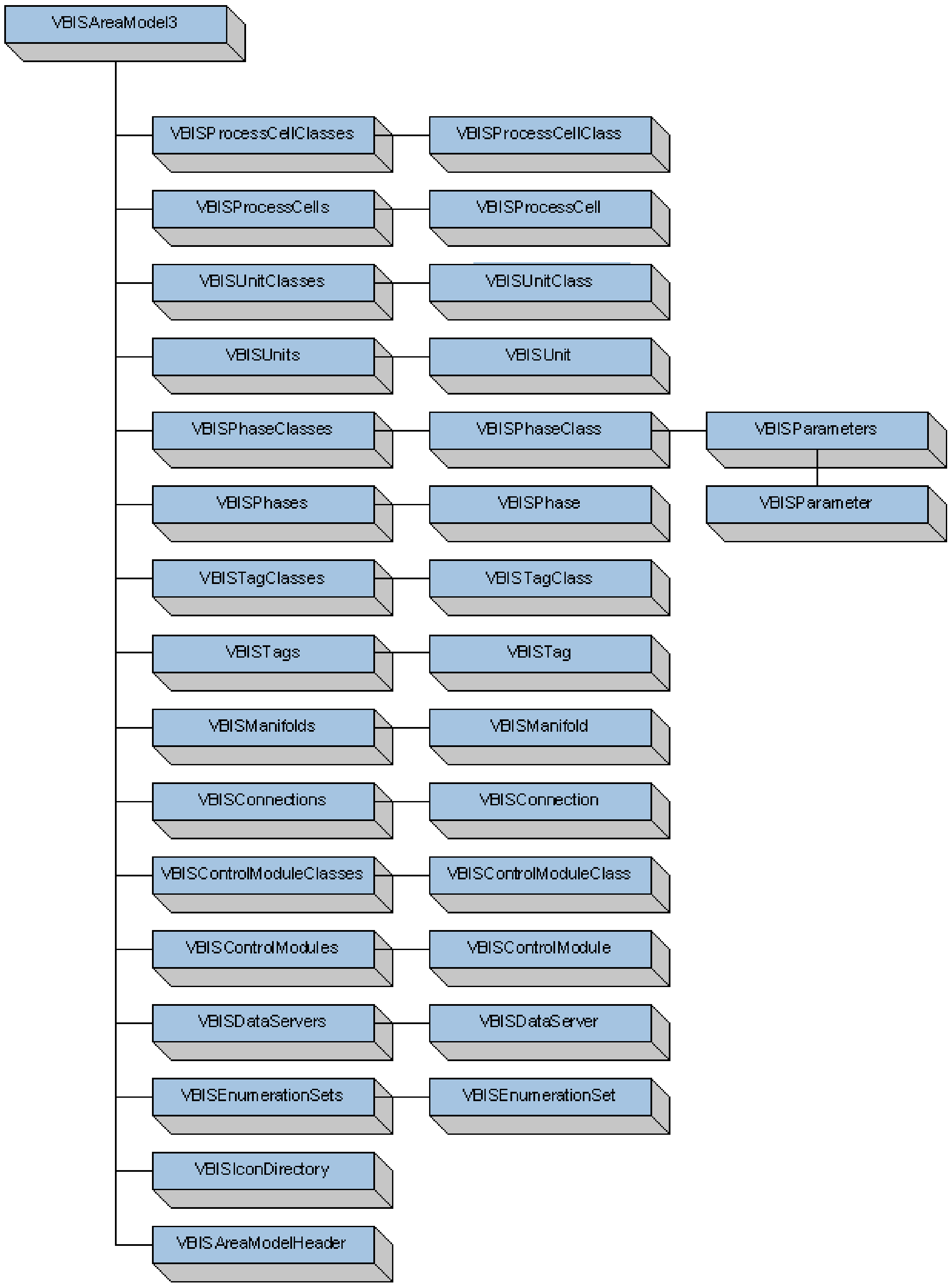
Understanding the VBISRecipeElements Hierarchy

The graphic below shows the **VBISRecipeElements** hierarchy. To get more information on each object, click the object name in the graphic.



Understanding the VBISAreaModel3 Hierarchy

The graphic below shows the **VBISAreaModel3** hierarchy. To get more information on each object, click the object name in the graphic.



VBIS8 Interface

The **VBIS8** interface is the root object in the **VBIS8** automation interface hierarchy. The **VBIS8** interface provides access to the following lower-level interfaces:

- VBISServer8
- VBISEquipment
- VBISAreaModel3
- VBISRecipeManagement3

VBISServer8 Interface

The **VBISServer8** interface is used to communicate with the Batch Execution server. The **VBISServer8** interface provides access to the following lower-level interfaces:

- VBISBatchControl5
- VBISBatchList
- VBISRecipeList3
- VBISAlarmsList
- VBISPromptList2
- VBISBindingPrompts2
- VBISEWIPromptItems
- VBISBatchListItems2
- VBISAlarmListItems
- VBISPromptListItems
- VBISStepControl2
- VBISPhaseControl
- VBISEWIPrompts
- VBISBreakpoints Interface
- VBISBreakpointPrompts Interface

- VBISRemovedBatchList Interface

You must instantiate **VBISServer8** from the **VBIS8** object interface.

Property

- Status

Methods

- ReConnect
- AuthenticateUser
- SetBreakpoint
- ClearBreakpoint

VBISBatchControl5 Interface

The **VBISBatchControl5** batch server control interface provides access 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. You must instantiate **VBISBatchControl5** from the **VBISServer8** object interface.

Properties

- GetParameters
- GetReportParameters
- UnitTags

Methods

- Add
- Bind
- State
- Command
- ReBind
- SetParameter
- AddEvent

- SetUnitTag
- EWIAddEvent
- SecurityAddEvent

VBISBatchList Interface

IMPORTANT: *VBISBatchList is provided for backwards compatibility only. For new application development, use the VBISBatchListItems2 Interface instead.*

The **VBISBatchList** interface provides access to batch list data stored in the Batch Execution Server. You must instantiate **VBISBatchList** from the **VBISServer8** object interface.

Properties

- Count
- Type
- Next

Method

- Query

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won’t reflect any changes in the Batch Execution Server until you call **Query**.

Batch list records are stored in safe arrays.

VBISRecipeList3 Interface

The **VBISRecipeList3** batch server recipe list interface provides access to recipe list data stored in the Batch Execution Server. You must instantiate **VBISRecipeList3** from the **VBISServer8** object interface.

Properties

- Count
- Next
- Parameters
- Steps

Methods

- Query
- RecipeCollection

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Recipe list records are stored in safe arrays.

VBISAlarmsList Interface

IMPORTANT: *VBISAlarmList is provided for backwards compatibility only. For new application development, use the VBISAlarmListItems Interface instead.*

The **VBISAlarmsList** interface provides access to alarm list data stored in the Batch Execution Server. You must instantiate **VBISAlarmsList** from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Alarms list records are stored in safe arrays.

VBISPromptList2 Interface

IMPORTANT: *VBISPromptList2 is provided for backwards compatibility only. For new application development, use the VBISPromptListItems Interface instead.*

The **VBISPromptList2** interface provides access to prompt list data stored in the Batch Execution Server. You must instantiate **VBISPromptList2** from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query
- Acknowledge

Remarks

Use the **Query** method to initialize the batch list before you call the **Count** and **Next** properties. **Count** and **Next** will only return information from the batch list and won't reflect any changes in the Batch Execution Server until you call **Query**.

Prompt list records are stored in safe arrays.

VBISBindingPrompts2 Interface

The **VBISBindingPrompts2** interface is a collection of **VBISBindingPrompt2** objects. The **VBISBindingPrompts2** interface provides access to the following lower-level interface:

- VBISBindingPrompt2

Properties

- Count
- Item

VBISEWIPromptItems Interface

The **VBISEWIPromptItems** interface is a collection of **VBISEWIPromptItem** objects. The **VBISEWIPromptItems** interface provides access to the following lower-level interface:

- VBISEWIPromptItem

Properties

- Count
- Item

VBISBatchListItems2 Interface

The **VBISBatchListItems2** interface is a collection of **VBISBatchListItem2** objects. The **VBISBatchListItems2** interface provides access to the following lower-level interface:

- VBISBatchListItem2

Properties

- Count
- Item

VBISAlarmListItems Interface

The **VBISAlarmListItems** interface is a collection of **VBISAlarmListItem** objects. The **VBISAlarmListItems** interface provides access to the following lower-level interface:

- VBISAlarmListItem

Properties

- Count
- Item

VBISPromptListItems Interface

The **VBISPromptListItems** interface is a collection of **VBISPromptListItem** objects. The **VBISPromptListItems** interface provides access to the following lower-level interface:

- VBISPromptListItem

Properties

- Count
- Item

VBISStepControl2 Interface

The **VBISStepControl2** interface provides manual phase control to phases. You must instantiate VBISStepControl2 from the **VBISServer8** object interface. The VBISStepControl2 interface provides access to the following lower-level interfaces:

- VBISRecipeElements
- VBISRecipeTransitionExpression
- VBISRecipeStepListItems

Methods

- Command

- StartStep
- HoldStep
- RestartStep
- AbortStep
- StopStep
- ManualStep
- AutoStep
- ClearAllFailures
- VBISActiveRecipeStepListItems

VBISPhaseControl Interface

The **VBISPhaseControl** interface returns phase control interface object. You must instantiate VBISPhaseControl from the **VBISServer8** object interface.

Property

- VBISPhases2

Methods

- AcquirePhase
- ReleasePhase
- Command
- StartPhase

VBISEWIPrompts Interface

The **VBISEWIPrompts** interface provides access to EWI prompts stored in the Batch Execution Server. You must instantiate VBISEWIPrompts from the **VBISServer8** object interface.

Properties

- Count
- Next

Methods

- Query
- Acknowledge

VBISBreakpoints Interface

The **VBISBreakpoints interface** is a collection of **VBISBreakpoint** objects. The **VBISBreakpoints interface** provides access to the following lower-level interface:

- VBISBreakpoint Interface

Properties

- Count
- Item

VBISBreakpointPrompts Interface

The **VBISBreakpointPrompts** interface is a collection of **VBISBreakpointPrompt** objects. The **VBISBreakpointPrompts** interface provides access to the following lower-level interface:

- VBISBreakpointPrompt Interface

Properties

- Count
- Item

VBISRemovedBatchList Interface

The **VBISRemovedBatchList** interface provides the final state of batches that have been removed from the Batch Server.

Properties

- Count
- Item
- Next

Methods

- Query

VBISEquipment Interface

The **VBISEquipment** interface provides access to the following lower-level object interface:

- VBISEnumerations

You must instantiate **VBISEquipment** from the **VBIS8** or **VBIS** object interface.

Properties

- VBISEnumerations

VBISAreaModel3

The **VBISAreaModel3** interface provides access to the equipment defined in the Batch Execution area model. You must instantiate **VBISAreaModel3** from the **VBIS8** object interface.

Properties

- Name
- Revision
- VBISEnumerationSets
- VBISProcessCellClasses
- VBISProcessCells
- VBISUnitClasses
- VBISUnits
- VBISPhaseClasses
- VBISPhases
- VBISTagClasses
- VBISTags
- VBISManifolds
- VBISConnections
- VBISControlModuleClasses
- VBISControlModules

- VBISDataServers
- VBISIconDirectory
- ItemPositions
- ItemIconNames
- IconFromFilenames
- VBISAreaModelHeader

VBISProcessCellClasses Interface

The **VBISProcessCellClasses** interface is a collection of **VBISProcessCellClass** objects defined in the area model. The **VBISProcessCellClasses** interface provides access to the following lower-level interface:

- VBISProcessCellClass

Properties

- Count
- Item

VBISProcessCells Interface

The **VBISProcessCells** interface is the collection of **VBISProcessCell** objects. The **VBISProcessCells** interface provides access to the following lower-level interface:

- VBISProcessCell

Properties

- Count
- Item

VBISUnitClasses Interface

The **VBISUnitClasses** interface is a collection of **VBISUnitClass** objects defined in the area model. The **VBISUnitClasses** interface provides access to the following lower-level interface:

- VBISUnitClass

Properties

- Count

- Item

VBISUnits Interface

The **VBISUnits** interface is a collection of **VBISUnit** objects. The **VBISUnits** interface provides access to the following lower-level interface:

- VBISUnit

Properties

- Count

- Item

VBISPhaseClasses Interface

The **VBISPhaseClasses** interface is a collection of **VBISPhaseClass** objects. The **VBISPhaseClasses interface** provides access to the following lower-level interface:

- VBISPhaseClass

Properties

- Count

- Item

VBISPhases Interface

The **VBISPhases** interface is a collection of **VBISPhase** objects. The **VBISPhases** interface provides access to the following lower-level interface:

- VBISPhase

Properties

- Count

- Item

VBISTagClasses Interface

The **VBISTagClasses** interface is a collection of **VBISTagClass** objects. The **VBISTagClasses** interface provides access to the following lower-level interface:

- VBISTagClass

Properties

- Count
- Item

VBISTags Interface

The **VBISTags** interface is a collection of **VBISTag** objects. The **VBISTags** interface provides access to the following lower-level interface:

- VBISTag

Properties

- Count
- Item

VBISManifolds Interface

The **VBISManifolds** interface is a collection of **VBISManifold** objects. The **VBISManifolds** interface provides access to the following lower-level interface:

- VBISManifold

Properties

- Count
- Item

VBISConnections Interface

The **VBISConnections** interface is a collection of **VBISConnection** objects. The **VBISConnections** interface provides access to the following lower-level interface:

- VBISConnection

Properties

- Count
- Item

VBISControlModuleClasses Interface

The **VBISControlModuleClasses** interface is a collection of **VBISControlModuleClass** objects. The **VBISControlModuleClasses** interface provides access to the following lower-level interface:

- VBISControlModuleClass

Properties

- Count
- Item

VBISControlModules Interface

The **VBISControlModules** interface is a collection of **VBISControlModule** objects. The **VBISControlModules** interface provides access to the following lower-level interface:

- VBISControlModule

Properties

- Count
- Item

VBISDataServers Interface

The **VBISDataServers** interface is a collection of **VBISDataServer** objects. The **VBISDataServers** interface provides access to the following lower-level interface:

- VBISDataServer

Properties

- Count
- Item

VBISEnumerationSets Interface

The **VBISEnumerationSets** interface is a collection of **VBISEnumerationSet** objects. The **VBISEnumerationSets** interface provides access to the following lower-level interface:

- VBISEnumerationSet

Properties

- Count
- Item

VBISIconDirectory Interface

The **VBISIconDirectory** interface provides access to the icon (bitmap) directories in the Batch Execution area model.

Properties

- ProcessCellClass
- UnitClass
- Phase
- Manifold

VBISAreaModelHeader Interface

The **VBISAreaModelHeader** interface provides access to audit trail data collected for the current area model.

Properties

- AreaAuditVersion
- AreaAuditPerformedByUserID
- AreaAuditPerformedByName
- AreaAuditPerformedByTime
- AreaAuditPerformedByComment
- AreaAuditVerifiedByUserID
- AreaAuditVerifiedByName
- AreaAuditVerifiedByTime
- AreaAuditVerifiedByComment

VBISRecipeManagement3 Interface

The **VBISRecipeManagement3** interface allows you to create and maintain recipes. The **VBISRecipeManagement3** interface provides access to the following lower-level object interface:

- VBISRecipe3

You must instantiate **VBISRecipeManagement3** from the **VBIS8** object interface.

Remarks

To use this interface your Batch Execution recipes must be stored in the relational database, meaning the recipe file type must be set to SQL in the Batch Execution WorkSpace project.

VBISRecipe3 Interface

The **VBISRecipe3** interface provides access to the recipe data stored in the Batch Execution Server. You must instantiate **VBISRecipe3** from the **VBISRecipeManagement3** object interface.

Methods

- ResetControl
- UpdateMaster
- Verify
- RebuildRecipeDir
- AddRecipe
- GetRecipeHeader
- GetProductFormulationHeader
- GetGlobalFormulationHeader

Remarks

To use this interface, your Batch Execution recipes and formulations must be stored in the relational database, meaning the recipe and formulation file types must be set to SQL in the Batch Execution WorkSpace project.

Examples

Visual Basic Examples

VBISActiveRecipeStepListItems Example

The following subroutine shows an example of how to access a filtered collection of VBISRecipeStepListItem objects. In this example, only Running and Held batches are returned.

```
Const cRUNNING =1

Const cHELD =8

Dim myMaskedValues as Long


myMaskedValues = cRUNNING And cHELD

          ' Interested in only seeing batches that are running and held

Set myVBISActiveRecipeStepListItems =
myVBISStepControl2.VBISActiveRecipeStepListItems("3\tMyRecipe",
MyMaskedValues)

End Sub
```

VBISAlarmListItems Example

The following example demonstrates how to populate the rows of the spreadsheet:

```
Public VBISApp As VBIS8

Public VBISServer As VBISServer8

Public VBISAlarmListItem As VBISAlarmListItem

' Instantiate VBIS and server interface

Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISServer = VBISApp.VBISServer8

vaAlarmSpread.Row = 0


' loop through each collection and display them in the spreadsheet

For Each VBISAlarmListItem In VBISServer.VBISAlarmListItems

    vaAlarmSpread.Row = vaAlarmSpread.Row + 1

    vaAlarmSpread.Col = BATCHID_COLUMN

    vaAlarmSpread.Text = VBISAlarmListItem.BatchID
```

```
        vaAlarmSpread.Col = PHASEID_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.PhaseID

        vaAlarmSpread.Col = FAILURE_MSG_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.FailureMessage

        vaAlarmSpread.Col = PHASE_NAME_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.PhaseName

        vaAlarmSpread.Col = PHASE_STATE_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.PhaseState

        vaAlarmSpread.Col = MODE_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.Mode

        vaAlarmSpread.Col = ARBITRATION_SET_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.ArbitrationSet


        vaAlarmSpread.Col = UNIT_ID_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.UnitID

        vaAlarmSpread.Col = UNIT_NAME_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.UnitName

        vaAlarmSpread.Col = OWNER_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.Owner

        vaAlarmSpread.Col = PHASEMSG_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.PhaseMessage

        vaAlarmSpread.Col = VALID_UNIT_LIST_COLUMN

        vaAlarmSpread.Text = VBISAlarmListItem.ValidUnitList

    Next
```

VBISBatchControl5.Add (Parameter Binding and Unit Binding)

```
    On Error GoTo ErrorHandler


    Dim VBISApp As VBIS8

    Dim VSObj As VBISServer8

    Dim BCObj As VBISBatchControl5
```

```
Dim lCampaignID as long

Dim RecipeID As String

Dim RecipeVersion As Long

Dim BatchID As String

Dim BatchScaling As Single

Dim UnitBindings As String

Dim ParameterBindings As String

Dim UseDefaultBindings As Long

Dim OpInteraction As Long

Dim OpBindParameters As Long

Dim OpBindUnits As Long

Dim BatchUniqueID As Long

Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


lCampaignID = 123   ‘ campaign id that this batch belongs to

RecipeID = "MAKE_TOOTHPASTE"

RecipeVersion = 1

BatchID = "MAKE_MINT_TOOTHPASTE"

BatchScaling = 100

ParameterBindings = "BAKINGSODA_AMT" + Chr(9) + "50" + Chr(9) + "FLAVOR"
+ Chr(9) + "SPEARMINT" +
Chr(9) + "FLAVOR_AMT" + Chr(9) + "22" + Chr(9) + "FLUORIDE_AMT" + Chr(9)
+ "11" + Chr(9) + "GUM_AMT" +
Chr(9) + "27" + Chr(9) + "PH_AMT" + Chr(9) + "55" + Chr(9) + "WATER_AMT"
+ Chr(9) + "78" + Chr(9) +
"WHITENER_AMT" + Chr(9) + "12"

UnitBindings = "BASE:1" + Chr(9) + "MIX1" + Chr(9) + "ADDITIVE:1" +
Chr(9) + "MIX2" + Chr(9) +
"FINAL:1" + Chr(9) + "REACTFLAVOR"

UseDefaultBindings = 0 'use 0 when specifying BOTH parameter bindings and
unit bindings

OpInteraction = 0
```

```
OpBindParameters = 0

OpBindUnits = 0

strCurrentUser = “Gary”


BatchUniqueID = BCObj.Add(lCampaignID, RecipeID, RecipeVersion, BatchID,
BatchScaling, UnitBindings,
ParameterBindings, UseDefaultBindings, OpInteraction, OpBindParameters,
OpBindUnits, strCurrentUser)


Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp=Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing

 Set VSObj = Nothing

 Set VBISApp=Nothing


End Sub
```

VBISBatchControl5.Add (Parameter Binding Only)

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5

Dim lCampaignID as long

Dim RecipeID As String

Dim RecipeVersion As Long

Dim BatchID As String

Dim BatchScaling As Single
Dim UnitBindings As String

Dim ParameterBindings As String

Dim UseDefaultBindings As Long

Dim OpInteraction As Long

Dim OpBindParameters As Long

Dim OpBindUnits As Long

Dim BatchUniqueID As Long

Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


lCampaignID = 123   ‘ campaign id that this batch belongs to

RecipeID = "MAKE_TOOTHPASTE"

RecipeVersion = 1

BatchID = "MAKE_MINT_TOOTHPASTE"

BatchScaling = 100

ParameterBindings = "BAKINGSODA_AMT" + Chr(9) + "50" + Chr(9) + "FLAVOR"
+ Chr(9) + "SPEARMINT" +
Chr(9) + "FLAVOR_AMT" + Chr(9) + "22" + Chr(9) + "FLUORIDE_AMT" + Chr(9)
+ "11" + Chr(9) +
"GUM_AMT" + Chr(9) + "27" + Chr(9) + "PH_AMT" + Chr(9) + "55" + Chr(9) +
"WATER_AMT" + Chr(9) + "78" +
Chr(9) + "WHITENER_AMT" + Chr(9) + "12"

UnitBindings = ""

UseDefaultBindings = 1 'use 1 when specifying parameter binding only

OpInteraction = 0

OpBindParameters = 0

OpBindUnits = 0

strCurrentUser = “Gary”


BatchUniqueID = BCObj.Add(lCampaignID, RecipeID, RecipeVersion, BatchID,
BatchScaling, UnitBindings,
ParameterBindings, UseDefaultBindings, OpInteraction, OpBindParameters,
OpBindUnits, strCurrentUser)
```

```
Set BCObj = Nothing
Set VSObj = Nothing
Set VBISApp=Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set BCObj = Nothing
 Set VSObj = Nothing
 Set VBISApp=Nothing
End Sub
```

VBISBatchControl5.Add (Unit Binding Only)

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VSObj As VBISServer8
Dim BCObj As VBISBatchControl5
Dim lCampaignID as long
Dim RecipeID As String
Dim RecipeVersion As Long
Dim BatchID As String
Dim BatchScaling As Single
Dim UnitBindings As String
Dim ParameterBindings As String
Dim UseDefaultBindings As Long
Dim OpInteraction As Long
Dim OpBindParameters As Long
Dim OpBindUnits As Long
Dim BatchUniqueID As Long
Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


lCampaignID = 123   ‘ campaign id that this batch belongs to

RecipeID = "MAKE_TOOTHPASTE"

RecipeVersion = 1

BatchID = "MAKE_MINT_TOOTHPASTE"

BatchScaling = 100

ParameterBindings = ""

UnitBindings = "BASE:1" + Chr(9) + "MIX1" + Chr(9) + "ADDITIVE:1" +
Chr(9) + "MIX2" +
Chr(9) + "FINAL:1" + Chr(9) + "REACTFLAVOR"

UseDefaultBindings = 2 'use 2 to specify unit bindings only

OpInteraction = 0

OpBindParameters = 0

OpBindUnits = 0

strCurrentUser = “Gary”


BatchUniqueID = BCObj.Add(lCampaignID, RecipeID, RecipeVersion, BatchID,
BatchScaling, UnitBindings,
ParameterBindings, UseDefaultBindings, OpInteraction, OpBindParameters,
OpBindUnits, strCurrentUser)


Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp=Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing
 Set VSObj = Nothing

 Set VBISApp=Nothing


End Sub
```

VBISBatchControl5.Add (Default Parameter Binding and Unit Binding)

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5

Dim lCampaignID as long

Dim RecipeID As String

Dim RecipeVersion As Long

Dim BatchID As String

Dim BatchScaling As Single

Dim UnitBindings As String

Dim ParameterBindings As String

Dim UseDefaultBindings As Long

Dim OpInteraction As Long

Dim OpBindParameters As Long

Dim OpBindUnits As Long

Dim BatchUniqueID As Long

Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


lCampaignID = 123   ‘ campaign id that this batch belongs to

RecipeID = "MAKE_TOOTHPASTE"

RecipeVersion = 1

BatchID = "MAKE_MINT_TOOTHPASTE"
```

```
BatchScaling = 100

ParameterBindings = ""

UnitBindings = ""

UseDefaultBindings = 3 'use 3 to use default parameter bindings and
default unit binding

OpInteraction = 0

OpBindParameters = 0

OpBindUnits = 0

strCurrentUser = “Gary”


BatchUniqueID = BCObj.Add(lCampaignID, RecipeID, RecipeVersion, BatchID,
BatchScaling, UnitBindings, ParameterBindings, UseDefaultBindings,
OpInteraction, OpBindParameters, OpBindUnits, strCurrentUser)


Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp=Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing

 Set VSObj = Nothing

 Set VBISApp=Nothing


End Sub
```

VBISBatchControl5.Bind

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5
Dim UnitBindings As String

Dim ParameterBindings As String

Dim Bindings As Long

Dim strCurrentUser As String

Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5



ParameterBindings = "BAKINGSODA_AMT" + Chr(9) + "50" + Chr(9) + "FLAVOR"
+ Chr(9) + "BUBBLEGUM" +
Chr(9) + "FLAVOR_AMT" + Chr(9) + "22" + Chr(9) + "FLUORIDE_AMT" + Chr(9)
+ "11" + Chr(9) +
"GUM_AMT" + Chr(9) + "27" + Chr(9) + "PH_AMT" + Chr(9) + "55" + Chr(9) +
"WATER_AMT" + Chr(9) +
"78" + Chr(9) + "WHITENER_AMT" + Chr(9) + "12"

UnitBindings = "BASE:1" + Chr(9) + "MIX1" + Chr(9) + "ADDITIVE:1" +
Chr(9) + "MIX2" + Chr(9) +
"FINAL:1" + Chr(9) + "REACTFLAVOR"

Bindings = 1

strCurrentUser = “Gary”



BatchUniqueID = 1

BCObj.Bind BatchUniqueID, UnitBindings, ParameterBindings, Bindings,
strCurrentUser



Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing



Exit Sub



ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing
 Set VSObj = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchControl5.State

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5

Dim BatchState As Long

Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


BatchUniqueID = 1

BatchState = BCObj.State(BatchUniqueID)


Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing

Exit Sub

ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing

 Set VSObj = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchControl5.Command

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5

Dim BatchCommand As String

Dim strCurrentUser As String

Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


BatchCommand = "REMOVE"

strCurrentUser = “Gary”


BatchUniqueID = 1

BCObj.Command BatchUniqueID, BatchCommand, strCurrentUser

Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing

 Set VSObj = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchControl5.GetParameters

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISSRVR As VBISServer8

Dim VBISBATCTL As VBISBatchControl5

Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim VBISREPPARMs As VBISParameters

Dim VBISREPPARM As VBISParameter

Dim StepName As String

Dim StepSelected As Integer

Dim REPstr As String

Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISSRVR = VBISApp.VBISServer8

Set VBISBATCTL = VBISSRVR.VBISBatchControl5


BatchUniqueID = 1

StepName = "BASE:1"

Set VBISPARMs = VBISBATCTL.GetParameters(Str(BatchUniqueID) + Chr(9) +
StepName)

StepName = "BASE:1" + Chr(9) + "MAKE_BASE:1" + Chr(9) + "COOL:1"

Set VBISREPPARMs = VBISBATCTL.GetReportParameters(Str(BatchUniqueID) +
Chr(9) + StepName)


For Each VBISPARM In VBISPARMs

strParmName = VBISPARM.Name

Next

For Each VBISREPPARM In VBISREPPARMs

strParmName = VBISREPPARM.Name

Next
```

```
Set VBISREPPARM = Nothing
Set VBISREPPARMs = Nothing
Set VBISParm = Nothing
Set VBISPARMs = Nothing
Set VBISBATCTL = Nothing
Set VBISSRVR = Nothing
Set VBISApp = Nothing
Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISREPPARM = Nothing
 Set VBISREPPARMs = Nothing
 Set VBISParm = Nothing
 Set VBISPARMs = Nothing
 Set VBISBATCTL = Nothing
 Set VBISSRVR = Nothing
 Set VBISApp = Nothing
End Sub
```

VBISBatchControl5.ReBind

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISBATCTL As VBISBatchControl5
Dim VBISPARMs As VBISParameters
Dim VBISPARM As VBISParameter
Dim VBISREPPARMs As VBISParameters
Dim VBISREPPARM As VBISParameter
Dim StepUNIT As String
Dim BindUNIT As String
Dim strCurrentUser As String
Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISSRVR = VBISApp.VBISServer8

Set VBISBATCTL = VBISSRVR.VBISBatchControl5

StepUNIT = "BASE:1"

BindUNIT = "MIX1"

strCurrentUser = “Gary”


BatchUniqueID = 1

VBISBATCTL.ReBIND BatchUniqueID, (StepUNIT + Chr(9) + BindUNIT),
strCurrentUser


Set VBISBATCLT = Nothing

Set VBISSRVR = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISBATCLT = Nothing

 Set VBISSRVR = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchControl5.SetParameter

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BCObj As VBISBatchControl5
Dim PhaseID As String

Dim ParameterName As String

Dim Value As String

Dim strCurrentUser As String

Dim BatchUniqueID As Long 'BatchUniqueID was returned when the batch was
scheduled


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BCObj = VSObj.VBISBatchControl5


BatchUniqueID = 1

PhaseID = Str(BatchUniqueID) + CHR(9) + "BASE:1" + CHR(9) + "MAKE_BASE:1"
+ CHR(9) + "ADD_INGS:1"

ParameterName = "FLAVOR_AMT"

Value = "44"

strCurrentUser = “Gary”

BCObj.SetParameter PhaseID, ParameterName, Value, strCurrentUser


Set BCObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set BCObj = Nothing

 Set VSObj = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchControl5.AddEvent

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISSRVR As VBISServer8

Dim VBISBATCTL As VBISBatchControl5

Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim VBISREPPARMs As VBISParameters

Dim VBISREPPARM As VBISParameter

Dim BatchID As String

Dim DESC As String

Dim Val As String

Dim EGU As String

Dim Cell As String

Dim Unit As String

Dim Phase As String

Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISSRVR = VBISApp.VBISServer8

Set VBISBATCTL = VBISSRVR.VBISBatchControl5


BatchId = "MAKE_MINT_TOOTHPASTE" ' must be the same name as at least one
batch id in

the batchlist

DESC = "Batch Event Description"

Val = "12"

EGU = "GALLONS"

Cell = "TOOTHPASTE"

Unit = "MIX1"

Phase = "AGITATE"

strCurrentUser = “Gary”
```

```
VBISBATCTL.AddEvent BatchID, DESC, Val, EGU, Cell, Unit, Phase,
strCurrentUser


Set VBISBATCTL = Nothing

Set VBISSRVR = Nothing

Set VBISApp = Nothing

Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISBATCTL = Nothing

 Set VBISSRVR = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBatchList: Count, Next, and Query

The following subroutine shows an example of refreshing and iterating through the batch list.

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim BLObj As VBISBatchList

Dim lCount As Long

Dim i As Integer

Dim varRecord As Variant

Dim BatchID As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set BLObj = VSObj.VBISBatchList


BLObj.Query 'Update the Batch list and count. This must be called prior
```

```
to
'getting the count and records to ensure that the list is up to date

lCount = BLObj.count 'Get the number of Batches in list

'Loop through the batches

For i = 1 To lCount
varRecord = BLObj.Next  'Get the next record in list
BatchID = varRecord(0)
Next i     'Process the next batch (if there is one)

Set BLObj = Nothing
Set VSObj = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set BLObj = Nothing
 Set VSObj = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISBatchListItems2 Example

The following example demonstrates how to populate the rows of the spreadsheet:

```
Public VBISApp As VBIS8
Public VBISServer As VBISServer8
Public VBISBatchListItem As VBISBatchListItem2
' Instantiate VBIS and server interface
```

```
Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISServer = VBISApp.VBISServer8

vaBatchListSpread.Row=0


' loop through each collection and display them in the spreadsheet

For Each VBISBatchListItem In VBISServer.VBISBatchListItems2

    vaBatchListSpread.Row = vaBatchListSpread.Row + 1


    vaBatchListSpread.Col = BATCHID_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.BatchID


    vaBatchListSpread.Col = RECIPE_NAME_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.RecipeName

    vaBatchListSpread.Col = RECIPE_VERSION_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.RecipeVersion

    vaBatchListSpread.Col = RECIPE_DESC_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.BatchDescription

    vaBatchListSpread.Col = BATCH_SCALE_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.Scale

    vaBatchListSpread.Col = BATCH_STATE_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.BatchState

    vaBatchListSpread.Col = START_TIME_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.StartTime

    vaBatchListSpread.Col = ELAPSED_TIME_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.ElapsedTime

    vaBatchListSpread.Col = FAILURES_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.Failures

    vaBatchListSpread.Col = BATCH_MODE_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.BatchMode


    vaBatchListSpread.Col = TYPE_COLUMN

    vaBatchListSpread.Text = VBISBatchListItem.Type
    vaBatchListSpread.Col = PARAMETERS_REQUIRED_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.ParametersRequired


    vaBatchListSpread.Col = UNITS_REQUIRED_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.UnitsRequired


    vaBatchListSpread.Col = PARAMETERS_SUPPORTED_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.ParametersSupplied


    vaBatchListSpread.Col = UNITS_SUPPORTED_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.UnitsSupplied


    vaBatchListSpread.Col = BATCH_BOUND_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.BatchBound


    vaBatchListSpread.Col = DEFAULT_BINDING_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.DefaultBind


    vaBatchListSpread.Col = OP_BIND_PARAMETERS_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.OperatorBindParameters


    vaBatchListSpread.Col = OP_BIND_UNITS_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.OperatorBindUnits


    vaBatchListSpread.Col = OP_INTERACTION_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.OperatorInteraction


    vaBatchListSpread.Col = PROCESS_CELL_LIST_COLUMN
    vaBatchListSpread.Text = VBISBatchListItem.ProcessCellList


    vaBatchListSpread.Col = PHASE_LIST_COLUMN
```

```
        vaBatchListSpread.Text = VBISBatchListItem.PhaseList


        vaBatchListSpread.Col = UNIT_LIST_COLUMN

        vaBatchListSpread.Text = VBISBatchListItem.UnitList


        vaBatchListSpread.Col = INTERNAL_ID_COLUMN

        vaBatchListSpread.Text = VBISBatchListItem.BatchSerialNumber


        vaBatchListSpread.Col = COMMAND_MASK_COLUMN

        vaBatchListSpread.Text = VBISBatchListItem.CommandMask


        vaBatchListSpread.Col = RECIPE_AUDIT_VERSION_COLUMN

        vaBatchListSpread.Text = VBISBatchListItem.RecipeAuditVersion

    Next
```

VBISRecipeList3: Count, Next, and Query

This following subroutine shows an example of refreshing and iterating through the recipe list.

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim lCount As Long

Dim i As Integer

Dim varRecord As Variant

Dim VSObj As VBISServer8

Dim RLObj As VBISRecipeList3

Dim RecipeID As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set RLObj = VSObj.VBISRecipeList3
```

```
RLObj.Query 'Update the Recipe list and count. This must be called prior
to

'getting the count and records to ensure that the list is up to date


lCount = RLObj.count 'Get the number of Recipes in list


'Loop through the recipes


For i = 1 To lCount

varRecord = RLObj.Next 'Get the next record in list

 RecipeID = varRecord(0)

Next i     'Process the next recipe (if there is one)


Set RLObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set RLObj = Nothing

 Set VSObj = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISRecipeList3.Parameters

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISSRVR As VBISServer8

Dim VBISRCPL As VBISRecipeList3
Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim RecipeID As String

Dim RecipeVer As String

Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISSRVR = VBISApp.VBISServer8

Set VBISRCPL = VBISSRVR.VBISRecipeList3

RecipeID = "MAKE_TOOTHPASTE"

RecipeVer = "1.0"

Set VBISPARMs = VBISRCPL.Parameters(RecipeID, RecipeVer)


For Each VBISPARM In VBISPARMs

Name = VBISPARM.Name

Next

Set VBISPARM = Nothing

Set VBISPARMs = Nothing

Set VBISRCPL = Nothing

Set VBISSRVR = Nothing

Set VBISApp = Nothing


Exit Sub

ErrorHandler:

 MsgBox (Err.Description)

 Set VBISPARM = Nothing

 Set VBISPARMs = Nothing

 Set VBISRCPL = Nothing

 Set VBISSRVR = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISRecipeList3.Steps

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISRCPL As VBISRecipeList3
Dim VBISRCPSTEPs As VBISSteps
Dim VBISRCPSTEP As VBISStep
Dim RecipeID As String
Dim RecipeVer As String
Dim Name As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISSRVR = VBISApp.VBISServer8
Set VBISRCPL = VBISSRVR.VBISRecipeList3
RecipeID = "MAKE_TOOTHPASTE"
RecipeVer = "1.0"
Set VBISRCPSTEPs = VBISRCPL.Steps(RecipeID, RecipeVer)

For Each VBISRCPSTEP In VBISRCPSTEPs
Name = VBISRCPSTEP.Name
Next

Set VBISRCPSTEP = Nothing
Set VBISRCPSTEPs = Nothing
Set VBISRCPL = Nothing
Set VBISSRVR = Nothing
Set VBISApp = Nothing

Exit Sub
```

```
ErrorHandler:

 MsgBox (Err.Description)

 Set VBISRCPSTEP = Nothing

 Set VBISRCPSTEPs = Nothing

 Set VBISRCPL = Nothing

 Set VBISSRVR = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAlarmsList: Count, Next, and Query

The following subroutine shows an example of refreshing and iterating through the alarm list and clearing the alarms.

```
Private Declare Sub Sleep Lib "kernel32" _

(ByVal dwMilliseconds As Long)


:

:

:


On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VSObj As VBISServer8

Dim ALObj As VBISAlarmsList

Dim lCount As Long

Dim i As Integer

Dim varRecord As Variant

Dim strPhaseID As String

Dim strBatchID As String

Dim strUnit As String

Dim FailMessage As String
```

```
Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set ALObj = VSObj.VBISAlarmsList


Sleep(1000) 'Allow VBIS to subscribe before querying for alarms

ALObj.Query 'Update the Alarm list and count. This must be called prior
to

'getting the count and records to ensure that the list is up to date


lCount = ALObj.count 'Get the number of Alarms in list


For i = 1 To lCount

varRecord = ALObj.Next 'Get the next record in list

 FailMessage = varRecord(9)

Next i     'Process the next alarm (if there is one)


Set ALObj = Nothing

Set VSObj = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set ALObj = Nothing

 Set VSObj = Nothing

 Set VBISApp = Nothing

End Sub
```

VBISPromptList2: Count, Next, Query, and Acknowledge

The following subroutine shows an example of refreshing and iterating through the prompt list and acknowledging the prompts.

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim lCount As Long

Dim i As Integer

Dim varRecord As Variant

Dim lPromptID As Long

Dim strPromptResponse As String

Dim VSObj As VBISServer8

Dim PLObj As VBISPromptList2

Dim strCurrentUser As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VSObj = VBISApp.VBISServer8

Set PLObj = VSObj.VBISPromptList2


PLObj.Query 'Update the Prompt list and count. This must be called prior
to

'getting the count and records to ensure that the list is up to date


lCount = PLObj.count 'Get the number of Prompts in list


strCurrentUser = “Gary”


'Loop through the prompts acknowledging them

For i = 1 To lCount


varRecord = PLObj.Next 'Get the next record in list

lPromptID = CLng(varRecord(11)) 'Get the prompt ID from the record
```

```
strPromptResponse = "0" 'Respond to the prompt. In this example, we
      'respond with 0. In an actual application,
      'you must respond with a valid response
PLObj.Acknowledge lPromptID, strPromptResponse, strCurrentUser

Next i     'Process the next prompt (if there is one)
Set PLObj = Nothing
Set VSObj = Nothing
Set VBISApp = Nothing
Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set PLObj = Nothing
 Set VSObj = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISBindingPrompts2 Get All Prompts

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISBindingPRs As VBISBindingPrompts2
Dim VBISBindingPR As VBISBindingPrompt2
Dim BindingIndex As Integer
Dim Start As Integer
Dim Description As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISSRVR = VBISApp.VBISServer8
Set VBISBindingPRs = VBISSRVR.VBISBindingPrompts2
```

```
For Each VBISBindingPR In VBISBindingPRs
Description = VBISBindingPR.Description
Next

Set VBISBindingPR = Nothing
Set VBISBindingPRs = Nothing
Set VBISSRVR = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISBindingPR = Nothing
 Set VBISBindingPRs = Nothing
 Set VBISSRVR = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISBindingPrompt2 Details

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISBindingPRs As VBISBindingPrompts2
Dim VBISBindingPR As VBISBindingPrompt2
Dim VBISBindingUTs As VBISBindingUnits
Dim VBISBindingUT As VBISBindingUnit
Dim EventID As String
Dim Name As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISSRVR = VBISApp.VBISServer8

Set VBISBindingPRs = VBISSRVR.VBISBindingPrompts2

EventID = "1"

If VBISBindingPRs.Count>0 Then

 Set VBISBindingPR = VBISBindingPRs.Item(EventID)


Set VBISBindingUTs = VBISBindingPR.VBISBindingUnits

For Each VBISBindingUT In VBISBindingUTs

Name = VBISBindingUT.Name

Next

End If


Set VBISBindingPR = Nothing

Set VBISBindingPRs = Nothing

Set VBISSRVR = Nothing

Set VBISApp = Nothing



Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISBindingPR = Nothing

 Set VBISBindingPRs = Nothing

 Set VBISSRVR = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISBindingPrompt2.Acknowledge

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8

Dim VBISSRVR As VBISServer8
Dim VBISBindingPRs As VBISBindingPrompts2
Dim VBISBindingPR As VBISBindingPrompt2
Dim strCurrentUser As String
Dim EventID As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISSRVR = VBISApp.VBISServer8
Set VBISBindingPRs = VBISSRVR.VBISBindingPrompts2

EventID = "1" ' Event id of an outstanding binding prompt
If VBISBindingPRs.Count>0 Then
 Set VBISBindingPR = VBISBindingPRs.Item(EventID)
 strCurrentUser = "Gary"
 VBISBindingPR.Acknowledge "MIX1", strCurrentUser
End If

Set VBISBindingPR = Nothing
Set VBISBindingPRs = Nothing
Set VBISSRVR = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISBindingPR = Nothing
 Set VBISBindingPRs = Nothing
 Set VBISSRVR = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISProcessCellClasses, VBISAreaModel3.VBISProcessCellClass

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPCs As VBISProcessCells

Dim VBISPC As VBISProcessCell

Dim VBISPCCs As VBISProcessCellClasses

Dim VBISPCC As VBISProcessCellClass

Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPCCs = VBISAM.VBISProcessCellClasses

Set VBISPCC = VBISPCCs.Item("INTYPLANT")

Set VBISPCs = VBISPCC.VBISProcessCells


For Each VBISPC In VBISPCs

Name = VBISPC.Name

Next


Set VBISPCC = Nothing

Set VBISPCCs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISPCC = Nothing
 Set VBISPCCs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISProcessCells, VBISAreaModel3.VBISProcessCell

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPCs As VBISProcessCells

Dim VBISPC As VBISProcessCell

Dim VBISUTs As VBISUnits

Dim VBISUT As VBISUnit

Dim VBISCONs As VBISConnections

Dim VBISCON As VBISConnection

Dim VBISMFs As VBISManifolds

Dim VBISMF As VBISManifold

Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPCs = VBISAM.VBISProcessCells

Set VBISPC = VBISPCs.Item("TOOTHPASTE")


Set VBISUTs = VBISPC.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name

Next


Set VBISCONs = VBISPC.VBISConnections

For Each VBISCON In VBISCONs
```

```
Name = VBISCON.Name
Next

Set VBISMFs = VBISPC.VBISManifolds
For Each VBISMF In VBISMFs
Name = VBISMF.Name
Next



Set VBISMF = Nothing
Set VBISMFs = Nothing
Set VBISCON = Nothing
Set VBISCONs = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISPC = Nothing
Set VBISPCs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISMF = Nothing
 Set VBISMFs = Nothing
 Set VBISCON = Nothing
 Set VBISCONs = Nothing
 Set VBISUT = Nothing
 Set VBISUTs = Nothing
 Set VBISPC = Nothing
 Set VBISPCs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISProcessCell, VBISProcessCell.VBISNeededEquipment

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPCs As VBISProcessCells
Dim VBISPC As VBISProcessCell
Dim VBISUTs As VBISUnits
Dim VBISUT As VBISUnit
Dim VBISPHs As VBISPhases
Dim VBISPH As VBISPhase
Dim VBISCONs As VBISConnections
Dim VBISCON As VBISConnection
Dim VBISCMs As VBISControlModules
Dim VBISCM As VBISControlModule
Dim VBISMFs As VBISManifolds
Dim VBISMF As VBISManifold
Dim VBISEQUIP As VBISNeededEquipment
Dim Name As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISPCs = VBISAM.VBISProcessCells
Set VBISPC = VBISPCs.Item("TOOTHPASTE")
Set VBISEQUIP = VBISPC.VBISNeededEquipment

Set VBISCONs = VBISEQUIP.VBISConnections
```

```
For Each VBISCON In VBISCONs

Name = VBISCON.Name

Next

Set VBISPCs = VBISEQUIP.VBISProcessCells

For Each VBISPC In VBISPCs

Name = VBISPC.Name

Next

Set VBISUTs = VBISEQUIP.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name

Next

Set VBISPHs = VBISEQUIP.VBISPhases

For Each VBISPH In VBISPHs

Name = VBISPH.Name

Next

Set VBISCMs = VBISEQUIP.VBISControlModules

For Each VBISCM In VBISCMs

Name = VBISCM.Name

Next

Set VBISMFs = VBISEQUIP.VBISManifolds

For Each VBISMF In VBISMFs

Name = VBISMF.Name

Next


Set VBISMF = Nothing

Set VBISMFs = Nothing

Set VBISCM = Nothing

Set VBISCMs = Nothing

Set VBISPH = Nothing

Set VBISPHs = Nothing

Set VBISUT = Nothing

Set VBISUTs = Nothing
Set VBISCON = Nothing
Set VBISCONs = Nothing
Set VBISEQUIP = Nothing
Set VBISPC = Nothing
Set VBISPCs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISMF = Nothing
 Set VBISMFs = Nothing
 Set VBISCM = Nothing
 Set VBISCMs = Nothing
 Set VBISPH = Nothing
 Set VBISPHs = Nothing
 Set VBISUT = Nothing
 Set VBISUTs = Nothing
 Set VBISCON = Nothing
 Set VBISCONs = Nothing
 Set VBISEQUIP = Nothing
 Set VBISPC = Nothing
 Set VBISPCs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISUnitClasses, VBISAreaModel3.VBISUnitClass

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISUTs As VBISUnits

Dim VBISUT As VBISUnit

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISUTCs As VBISUnitClasses

Dim VBISUTC As VBISUnitClass

Dim VBISPHCs As VBISPhaseClasses

Dim VBISPHC As VBISPhaseClass

Dim VBISTGCs As VBISTagClasses

Dim VBISTGC As VBISTagClass

Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISUTCs = VBISAM.VBISUnitClasses

Set VBISUTC = VBISUTCs.Item("MIXER")


Set VBISUTs = VBISUTC.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name

Next


Set VBISTGCs = VBISUTC.VBISTagClasses

For Each VBISTGC In VBISTGCs

Name = VBISTGC.Name

Next
```

```
Set VBISPHCs = VBISUTC.VBISPhaseClasses

For Each VBISPHC In VBISPHCs

Name = VBISPHC.Name

Next


Set VBISPHs = VBISUTC.VBISPhases

For Each VBISPH In VBISPHs

Name = VBISPH.Name

Next


Set VBISPHC = Nothing

Set VBISPHCs = Nothing

Set VBISTGC = Nothing

Set VBISTGCs = Nothing

Set VBISUT = Nothing

Set VBISUTs = Nothing

Set VBISUTC = Nothing

Set VBISUTCs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISPHC = Nothing

 Set VBISPHCs = Nothing

 Set VBISTGC = Nothing

 Set VBISTGCs = Nothing

 Set VBISUT = Nothing

 Set VBISUTs = Nothing

 Set VBISUTC = Nothing
 Set VBISUTCs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISUnits, VBISAreaModel3.VBISUnit

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISUTs As VBISUnits
Dim VBISUT As VBISUnit
Dim VBISPHs As VBISPhases
Dim VBISPH As VBISPhase
Dim VBISTGs As VBISTags
Dim VBISTG As VBISTag
Dim VBISTGCs As VBISTagClasses
Dim VBISTGC As VBISTagClass
Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISUTs = VBISAM.VBISUnits
Set VBISUT = VBISUTs.Item("MIX1")
'Works with or without .Item
'Set VBISUT = VBISUTs("MIX1")

Set VBISPHs = VBISUT.VBISPhases
For Each VBISPH In VBISPHs
Name = VBISPH.Name
Next
```

```
Set VBISTGs = VBISUT.VBISTags
For Each VBISTG In VBISTGs
Name = VBISTG.Name
Next

Set VBISTGCs = VBISUT.VBISTagClasses
For Each VBISTGC In VBISTGCs
Name = VBISTGC.Name
Next



Set VBISGC = Nothing
Set VBISGCs = Nothing
Set VBISTG = Nothing
Set VBISTGs = Nothing
Set VBISPH = Nothing
Set VBISPHs = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISGC = Nothing
 Set VBISGCs = Nothing
 Set VBISTG = Nothing
 Set VBISTGs = Nothing
 Set VBISPH = Nothing
 Set VBISPHs = Nothing
 Set VBISUT = Nothing
 Set VBISUTs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISUnit, VBISUnit.VBISNeededEquipment

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPCs As VBISProcessCells
Dim VBISPC As VBISProcessCell
Dim VBISUTs As VBISUnits
Dim VBISUT As VBISUnit
Dim VBISPHs As VBISPhases
Dim VBISPH As VBISPhase
Dim VBISCONs As VBISConnections
Dim VBISCON As VBISConnection
Dim VBISCMs As VBISControlModules
Dim VBISCM As VBISControlModule
Dim VBISMFs As VBISManifolds
Dim VBISMF As VBISManifold
Dim VBISEQUIP As VBISNeededEquipment
Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISUTs = VBISAM.VBISUnits
Set VBISUT = VBISUTs.Item("MIX1")
Set VBISEQUIP = VBISUT.VBISNeededEquipment


Set VBISCONs = VBISEQUIP.VBISConnections
For Each VBISCON In VBISCONs
Name = VBISCON.Name
Next
Set VBISPCs = VBISEQUIP.VBISProcessCells
For Each VBISPC In VBISPCs
Name = VBISPC.Name
Next
Set VBISUTs = VBISEQUIP.VBISUnits
For Each VBISUT In VBISUTs
Name = VBISUT.Name
Next
Set VBISPHs = VBISEQUIP.VBISPhases
For Each VBISPH In VBISPHs
Name = VBISPH.Name
Next
Set VBISCMs = VBISEQUIP.VBISControlModules
For Each VBISCM In VBISCMs
Name = VBISCM.Name
Next
Set VBISMFs = VBISEQUIP.VBISManifolds
For Each VBISMF In VBISMFs
Name = VBISMF.Name
Next


Set VBISMF = Nothing
Set VBISMFs = Nothing
Set VBISCM = Nothing
Set VBISCMs = Nothing
Set VBISPH = Nothing
Set VBISPHs = Nothing

Set VBISUT = Nothing

Set VBISUTs = Nothing

Set VBISPC = Nothing

Set VBISPCs = Nothing

Set VBISEQUIP = Nothing

Set VBISUT = Nothing

Set VBISUTs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISMF = Nothing

 Set VBISMFs = Nothing

 Set VBISCM = Nothing

 Set VBISCMs = Nothing

 Set VBISPH = Nothing

 Set VBISPHs = Nothing

 Set VBISUT = Nothing

 Set VBISUTs = Nothing

 Set VBISPC = Nothing

 Set VBISPCs = Nothing

 Set VBISEQUIP = Nothing

 Set VBISUT = Nothing

 Set VBISUTs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISPhaseClasses, VBISAreaModel3.VBISPhaseClass

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISREPs As VBISReports

Dim VBISREP As VBISReport

Dim VBISPHCs As VBISPhaseClasses

Dim VBISPHC As VBISPhaseClass

Dim VBISMESs As VBISMessages

Dim VBISMES As VBISMessage

Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim Name As String




Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPHCs = VBISAM.VBISPhaseClasses

Set VBISPHC = VBISPHCs.Item("COOL")


Set VBISPHs = VBISPHC.VBISPhases

For Each VBISPH In VBISPHs

Name = VBISPH.Name

Next

Set VBISMESs = VBISPHC.VBISMessages

For Each VBISMES In VBISMESs

Name = VBISMES.Name

Next

Set VBISREPs = VBISPHC.VBISReports
```

```
For Each VBISREP In VBISREPs

Name = VBISREP.Name

Next

Set VBISPARMs = VBISPHC.VBISParameters

For Each VBISPARM In VBISPARMs

Name = VBISPARM.Name

Next


Set VBISPARM = Nothing

Set VBISPARMs = Nothing

Set VBISREP = Nothing

Set VBISREPs = Nothing

Set VBISMES = Nothing

Set VBISMESs = Nothing

Set VBISPH = Nothing

Set VBISPHs = Nothing

Set VBISPHC = Nothing

Set VBISPHCs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISPARM = Nothing

 Set VBISPARMs = Nothing

 Set VBISREP = Nothing

 Set VBISREPs = Nothing

 Set VBISMES = Nothing

 Set VBISMESs = Nothing

 Set VBISPH = Nothing
 Set VBISPHs = Nothing

 Set VBISPHC = Nothing

 Set VBISPHCs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISPhases, VBISAreaModel3.VBISPhase

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISUTs As VBISUnits

Dim VBISUT As VBISUnit

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISTGs As VBISTags

Dim VBISTG As VBISTag

Dim VBISREPs As VBISReports

Dim VBISREP As VBISReport

Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPHs = VBISAM.VBISPhases

Set VBISPH = VBISPHs.Item("COOL1")


Set VBISTGs = VBISPH.VBISParameterTags

For Each VBISTG In VBISTGs
```

```
Name = VBISTG.Name

Next


Set VBISTGs = VBISPH.VBISReportTags

For Each VBISTG In VBISTGs

Name = VBISTG.Name

Next


Set VBISTGs = VBISPH.VBISRequestTags

For Each VBISTG In VBISTGs

Name = VBISTG.Name

Next


Set VBISUTs = VBISPH.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name

Next


Set VBISPARMs = VBISPH.VBISParameters

For Each VBISPARM In VBISPARMs

Name = VBISPARM.Name

Next


Set VBISREPs = VBISPH.VBISPhaseReports

For Each VBISREP In VBISREPs

Name = VBISREP.Name

Next


Set VBISREP = Nothing

Set VBISREPs = Nothing

Set VBISPARM = Nothing

Set VBISPARMs = Nothing
Set VBISUT = Nothing

Set VBISUTs = Nothing

Set VBISTG = Nothing

Set VBISTGs = Nothing

Set VBISPH = Nothing

Set VBISPHs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISREP = Nothing

 Set VBISREPs = Nothing

 Set VBISPARM = Nothing

 Set VBISPARMs = Nothing

 Set VBISUT = Nothing

 Set VBISUTs = Nothing

 Set VBISTG = Nothing

 Set VBISTGs = Nothing

 Set VBISPH = Nothing

 Set VBISPHs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISUnit, VBISUnit.VBISNeededEquipment

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPCs As VBISProcessCells
Dim VBISPC As VBISProcessCell

Dim VBISUTs As VBISUnits

Dim VBISUT As VBISUnit

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISCONs As VBISConnections

Dim VBISCON As VBISConnection

Dim VBISCMs As VBISControlModules

Dim VBISCM As VBISControlModule

Dim VBISMFs As VBISManifolds

Dim VBISMF As VBISManifold

Dim VBISEQUIP As VBISNeededEquipment

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISUTs = VBISAM.VBISUnits

Set VBISUT = VBISUTs.Item("MIX1")

Set VBISEQUIP = VBISUT.VBISNeededEquipment


Set VBISCONs = VBISEQUIP.VBISConnections

For Each VBISCON In VBISCONs

Name = VBISCON.Name

Next

Set VBISPCs = VBISEQUIP.VBISProcessCells

For Each VBISPC In VBISPCs

Name = VBISPC.Name

Next

Set VBISUTs = VBISEQUIP.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name
```

```
Next
Set VBISPHs = VBISEQUIP.VBISPhases
For Each VBISPH In VBISPHs
Name = VBISPH.Name
Next
Set VBISCMs = VBISEQUIP.VBISControlModules
For Each VBISCM In VBISCMs
Name = VBISCM.Name
Next
Set VBISMFs = VBISEQUIP.VBISManifolds
For Each VBISMF In VBISMFs
Name = VBISMF.Name
Next

Set VBISMF = Nothing
Set VBISMFs = Nothing
Set VBISCM = Nothing
Set VBISCMs = Nothing
Set VBISPH = Nothing
Set VBISPHs = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISPC = Nothing
Set VBISPCs = Nothing
Set VBISEQUIP = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub
```

```
ErrorHandler:
 MsgBox (Err.Description)
 Set VBISMF = Nothing
 Set VBISMFs = Nothing
 Set VBISCM = Nothing
 Set VBISCMs = Nothing
 Set VBISPH = Nothing
 Set VBISPHs = Nothing
 Set VBISUT = Nothing
 Set VBISUTs = Nothing
 Set VBISPC = Nothing
 Set VBISPCs = Nothing
 Set VBISEQUIP = Nothing
 Set VBISUT = Nothing
 Set VBISUTs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISTagClasses, VBISAreaModel3.VBISTagClass

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISTGs As VBISTags
Dim VBISTG As VBISTag
Dim VBISTGCs As VBISTagClasses
Dim VBISTGC As VBISTagClass
Dim Name As String

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISTGCs = VBISAM.VBISTagClasses
Set VBISTGC = VBISTGCs.Item("COMMAND")

Set VBISTGs = VBISTGC.VBISTags
For Each VBISTG In VBISTGs
Name = VBISTG.Name
Next

Set VBISTGC = Nothing
Set VBISTGCs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISTGC = Nothing
 Set VBISTGCs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing
End Sub
```

VBISAreaModel3.VBISTags, VBISAreaModel3.VBISTag

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISTGs As VBISTags
Dim VBISTG As VBISTag
Dim Name As String
```

```
Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISTGs = VBISAM.VBISTags
Set VBISTG = VBISTGs.Item("COOL1_VC")

Set VBISTG = Nothing
Set VBISTGs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISTG = Nothing
 Set VBISTGs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISManifolds, VBISAreaModel3.VBISManifold

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPCs As VBISProcessCells
Dim VBISPC As VBISProcessCell
Dim VBISCONs As VBISConnections
Dim VBISCON As VBISConnection
Dim VBISMFs As VBISManifolds
Dim VBISMF As VBISManifold
Dim Name As String
```

```
Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISMFs = VBISAM.VBISManifolds

Set VBISMF = VBISMFs.Item("HDR_FLAVOR")


Set VBISCONs = VBISMF.VBISConnections

For Each VBISCON In VBISCONs

Name = VBISCON.Name

Next


Set VBISPCs = VBISMF.VBISProcessCells

For Each VBISPC In VBISPCs

Name = VBISPC.Name

Next


Set VBISPC = Nothing

Set VBISPCs = Nothing

Set VBISCON = Nothing

Set VBISCONs = Nothing

Set VBISMF = Nothing

Set VBISMFs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISPC = Nothing

 Set VBISPCs = Nothing
 Set VBISCON = Nothing
 Set VBISCONs = Nothing
 Set VBISMF = Nothing
 Set VBISMFs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISManifold, VBISManifold.VBISNeededEquipment

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPCs As VBISProcessCells
Dim VBISPC As VBISProcessCell
Dim VBISUTs As VBISUnits
Dim VBISUT As VBISUnit
Dim VBISPHs As VBISPhases
Dim VBISPH As VBISPhase
Dim VBISCONs As VBISConnections
Dim VBISCON As VBISConnection
Dim VBISCMs As VBISControlModules
Dim VBISCM As VBISControlModule
Dim VBISMFs As VBISManifolds
Dim VBISMF As VBISManifold
Dim VBISEQUIP As VBISNeededEquipment
Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISMFs = VBISAM.VBISManifolds
Set VBISMF = VBISMFs.Item("HDR_FLAVOR")
Set VBISEQUIP = VBISMF.VBISNeededEquipment

Set VBISCONs = VBISEQUIP.VBISConnections
For Each VBISCON In VBISCONs
CONList1.AddItem VBISCON.Name
Next
Set VBISPCs = VBISEQUIP.VBISProcessCells
For Each VBISPC In VBISPCs
Name = VBISPC.Name
Next
Set VBISUTs = VBISEQUIP.VBISUnits
For Each VBISUT In VBISUTs
Name = VBISUT.Name
Next
Set VBISPHs = VBISEQUIP.VBISPhases
For Each VBISPH In VBISPHs
Name = VBISPH.Name
Next
Set VBISCMs = VBISEQUIP.VBISControlModules
For Each VBISCM In VBISCMs
Name = VBISCM.Name
Next
Set VBISMFs = VBISEQUIP.VBISManifolds
For Each VBISMF In VBISMFs
Name = VBISMF.Name
Next

Set VBISMF = Nothing
Set VBISMFs = Nothing
Set VBISCM = Nothing
Set VBISCMs = Nothing
Set VBISPH = Nothing

Set VBISPHs = Nothing

Set VBISUT = Nothing

Set VBISUTs = Nothing

Set VBISPC = Nothing

Set VBISPCs = Nothing

Set VBISEQUIP = Nothing

Set VBISMF = Nothing

Set VBISMFs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub

ErrorHandler:

 MsgBox (Err.Description)

 Set VBISMF = Nothing

 Set VBISMFs = Nothing

 Set VBISCM = Nothing

 Set VBISCMs = Nothing

 Set VBISPH = Nothing

 Set VBISPHs = Nothing

 Set VBISUT = Nothing

 Set VBISUTs = Nothing

 Set VBISPC = Nothing

 Set VBISPCs = Nothing

 Set VBISEQUIP = Nothing

 Set VBISMF = Nothing

 Set VBISMFs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISCONs As VBISConnections

Dim VBISCON As VBISConnection

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISCONs = VBISAM.VBISConnections

Set VBISCON = VBISCONs.Item("CONNECTION1")


Name = VBISCON.Name


Set VBISCON = Nothing

Set VBISCONs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISCON = Nothing

 Set VBISCONs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISControlModuleClasses, VBISAreaModel3.VBISControlModuleClass

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISCMs As VBISControlModules

Dim VBISCM As VBISControlModule

Dim VBISCMCs As VBISControlModuleClasses

Dim VBISCMC As VBISControlModuleClass

Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISCMCs = VBISAM.VBISControlModuleClasses

If VBISCMCs.Count > 0 Then

Set VBISCMC = VBISCMCs.Item("RESOURCE1")


Set VBISCMs = VBISCMC.VBISControlModules

For Each VBISCM In VBISCMs

Name = VBISCM.Name

Next


Set VBISCMC = Nothing

Set VBISCMCs = Nothing

End If

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:
```

```
MsgBox (Err.Description)

Set VBISCMC = Nothing

Set VBISCMCs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISControlModules, VBISAreaModel3.VBISControlModule

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISCMs As VBISControlModules

Dim VBISCM As VBISControlModule

Dim Name As String




Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISCMs = VBISAM.VBISControlModules

If VBISCMs.Count > 0 Then

Set VBISCM = VBISCMs.Item("RESOURCE1")

Name = VBISCM.Name

Set VBISCM = Nothing

Set VBISCMs = Nothing

End If

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub
```

```
ErrorHandler:

MsgBox (Err.Description)

Set VBISCM = Nothing

Set VBISCMs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISControlModule, VBISControlModule.VBISNeededEquipment

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPCs As VBISProcessCells

Dim VBISPC As VBISProcessCell

Dim VBISUTs As VBISUnits

Dim VBISUT As VBISUnit

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISCONs As VBISConnections

Dim VBISCON As VBISConnection

Dim VBISCMs As VBISControlModules

Dim VBISCM As VBISControlModule

Dim VBISMFs As VBISManifolds

Dim VBISMF As VBISManifold

Dim VBISEQUIP As VBISNeededEquipment

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISCMs = VBISAM.VBISControlModules
```

```
If VBISCMs.Count > 0 Then

Set VBISCM = VBISCMs.Item("RESOURCE1")

Set VBISEQUIP = VBISCM.VBISNeededEquipment


Set VBISCONs = VBISEQUIP.VBISConnections

For Each VBISCON In VBISCONs

Name = VBISCON.Name

Next

Set VBISPCs = VBISEQUIP.VBISProcessCells

For Each VBISPC In VBISPCs

Name = VBISPC.Name

Next

Set VBISUTs = VBISEQUIP.VBISUnits

For Each VBISUT In VBISUTs

Name = VBISUT.Name

Next

Set VBISPHs = VBISEQUIP.VBISPhases

For Each VBISPH In VBISPHs

Name = VBISPH.Name

Next

Set VBISCMs = VBISEQUIP.VBISControlModules

For Each VBISCM In VBISCMs

Name = VBISCM.Name

Next

Set VBISMFs = VBISEQUIP.VBISManifolds

For Each VBISMF In VBISMFs

Name = VBISMF.Name

Next


Set VBISMF = Nothing

Set VBISMFs = Nothing

Set VBISCM = Nothing
Set VBISCMs = Nothing
Set VBISPH = Nothing
Set VBISPHs = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISPC = Nothing
Set VBISPCs = Nothing
Set VBISEQUIP = Nothing
End If

Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
MsgBox (Err.Description)
Set VBISMF = Nothing
Set VBISMFs = Nothing
Set VBISCM = Nothing
Set VBISCMs = Nothing
Set VBISPH = Nothing
Set VBISPHs = Nothing
Set VBISUT = Nothing
Set VBISUTs = Nothing
Set VBISPC = Nothing
Set VBISPCs = Nothing
Set VBISEQUIP = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISDataServers, VBISAreaModel3.VBISDataServer

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISDSs As VBISDataServers

Dim VBISDS As VBISDataServer

Dim Name As String




Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISDSs = VBISAM.VBISDataServers

Set VBISDS = VBISDSs.Item("VBSIM")


Name = VBISDS.Name


Set VBISDS = Nothing

Set VBISDSs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

 MsgBox (Err.Description)

 Set VBISDS = Nothing

 Set VBISDSs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISEnumerationSets, VBISAreaModel3.VBISEnumerationSet, VBISAreaModel3.VBISEnumerations2, VBISAreaModel3.VBISEnumeration

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISENUMs As VBISEnumerationSets

Dim VBISENUM As VBISEnumerationSet

Dim VBISENs As VBISEnumerations2

Dim VBISEN As VBISEnumeration

Dim ENUMstr As String

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISENUMs = VBISAM.VBISEnumerationSets

Set VBISENUM = VBISENUMs.Item("FLAVORS")


Set VBISENs = VBISENUM.VBISEnumerations2

For Each VBISEN In VBISENs

Name = VBISEN.Name

Next


Set VBISENUM = Nothing

Set VBISENUMs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:
 MsgBox (Err.Description)

 Set VBISENUM = Nothing

 Set VBISENUMs = Nothing

 Set VBISAM = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISAreaModel3.VBISReports, VBISAreaModel3.VBISReport

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISAM As VBISAreaModel3

Dim VBISPHs As VBISPhases

Dim VBISPH As VBISPhase

Dim VBISREPs As VBISReports

Dim VBISREP As VBISReport

Dim Name As String




Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPHs = VBISAM.VBISPhases

Set VBISPH = VBISPHs.Item("COOL1")

Set VBISREPs = VBISPH.VBISPhaseReports

Set VBISREP = VBISREPs.Item("COOL_TEMP")


Name = VBISREP.Name


Set VBISREP = Nothing

Set VBISREPs = Nothing

Set VBISPH = Nothing

Set VBISPHs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VBISREP = Nothing
 Set VBISREPs = Nothing
 Set VBISPH = Nothing
 Set VBISPHs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISAreaModel3.VBISMessages, VBISAreaModel3.VBISMessage

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPHCs As VBISPhaseClasses
Dim VBISPHC As VBISPhaseClass
Dim VBISMESs As VBISMessages
Dim VBISMES As VBISMessage
Dim Name As String


Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISAM = VBISApp.VBISAreaModel3
Set VBISPHCs = VBISAM.VBISPhaseClasses
Set VBISPHC = VBISPHCs.Item("AGITATE")
Set VBISMESs = VBISPHC.VBISMessages
```

```
If VBISMESs.Count > 0 Then
Set VBISMES = VBISMESs.Item("MESSAGE1")
Name = VBISMES.Name
Set VBISMES = Nothing
End If

Set VBISMESs = Nothing
Set VBISPHC = Nothing
Set VBISPHCs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing
Exit Sub

ErrorHandler:
MsgBox (Err.Description)
Set VBISMES = Nothing
Set VBISMESs = Nothing
Set VBISPHC = Nothing
Set VBISPHCs = Nothing
Set VBISAM = Nothing
Set VBISApp = Nothing
End Sub
```

VBISAreaModel3.VBISParameters, VBISAreaModel3.VBISParameter

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISAM As VBISAreaModel3
Dim VBISPCs As VBISPhaseClasses
Dim VBISPC As VBISPhaseClass
Dim VBISPHs As VBISPhases
Dim VBISPH As VBISPhase
Dim VBISENs As VBISEnumerations2
Dim VBISEN As VBISEnumeration

Dim VBISPARMs As VBISParameters

Dim VBISPARM As VBISParameter

Dim Name As String



Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISAM = VBISApp.VBISAreaModel3

Set VBISPHs = VBISAM.VBISPhases

Set VBISPH = VBISPHs.Item("AGITATE1")

Set VBISPARMs = VBISPH.VBISParameters

Set VBISPARM = VBISPARMs.Item("SPEED")


If VBISPARM.Type = 5 Then

Set VBISENs = VBISPARM.Enumerations

For Each VBISEN In VBISENs

Name = VBISEN.Name

Next

End If


Set VBISEN = Nothing

Set VBISENs = Nothing

Set VBISPHC = Nothing

Set VBISPHCs = Nothing

Set VBISPARM = Nothing

Set VBISPARMs = Nothing

Set VBISPH = Nothing

Set VBISPHs = Nothing

Set VBISAM = Nothing

Set VBISApp = Nothing

Exit Sub
```

```
ErrorHandler:
 MsgBox (Err.Description)
 Set VBISEN = Nothing
 Set VBISENs = Nothing
 Set VBISPHC = Nothing
 Set VBISPHCs = Nothing
 Set VBISPARM = Nothing
 Set VBISPARMs = Nothing
 Set VBISPH = Nothing
 Set VBISPHs = Nothing
 Set VBISAM = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISPromptListItems Example

The following example demonstrates how to populate the rows of the spreadsheet:

```
Public VBISApp As VBIS8
Public VBISServer As VBISServer8
Public VBISPromptListItem As VBISPromptListItem
' Instantiate VBIS and server interface
Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISServer = VBISApp.VBISServer8
vaOperatorPromptsSpread.Row = 0

' loop through each collection and display them in the spreadsheet
For Each VBISPromptListItem In VBISServer.VBISPromptListItems

    vaOperatorPromptsSpread.Row = vaOperatorPromptsSpread.Row + 1

    vaOperatorPromptsSpread.Col = BATCHID_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.BatchID
    vaOperatorPromptsSpread.Col = RECIPE_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Recipe
    vaOperatorPromptsSpread.Col = EQUIPMENT_DESCRIPTION_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Description
    vaOperatorPromptsSpread.Col = TIME_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Time
    vaOperatorPromptsSpread.Col = EVENT_TYPE_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.EventType
    vaOperatorPromptsSpread.Col = VALUE_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Value
    vaOperatorPromptsSpread.Col = EU_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.EngineeringUnits

    vaOperatorPromptsSpread.Col = AREA_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.AreaModel
    vaOperatorPromptsSpread.Col = PROCESS_CELL_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.ProcessCell
    vaOperatorPromptsSpread.Col = UNIT_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Unit
    vaOperatorPromptsSpread.Col = PHASE_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.Phase
    vaOperatorPromptsSpread.Col = EVENTID_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.EventID

    vaOperatorPromptsSpread.Col = RESPONSE_TYPE_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.ResponseType

    vaOperatorPromptsSpread.Col = HIGH_COLUMN
    vaOperatorPromptsSpread.Text = VBISPromptListItem.High

    vaOperatorPromptsSpread.Col = LOW_COLUMN
```

```
        vaOperatorPromptsSpread.Text = VBISPromptListItem.Low


        vaOperatorPromptsSpread.Col = DEFAULT_COLUMN

        vaOperatorPromptsSpread.Text = VBISPromptListItem.Default


    Next
```

VBISRecipe3.ResetControl

```
    On Error GoTo ErrorHandler


    Dim VBISApp As VBIS8


    Set VBISApp = CreateObject("Intellution.VBIS.8")


    Dim RecipeID As String

    Dim RecipeVersion As Long


    Dim RMObj As VBISRecipeManagement3

    Dim VRObj As VBISRecipe3


    Set RMObj = VBISApp.VBISRecipeManagement3

    Set VRObj = RMObj.VBISRecipe3


    RecipeID = "MAKE_TOOTHPASTE"

    RecipeVersion = 1


    VRObj.ResetControl RecipeID, RecipeVersion

    Set VRObj = Nothing

    Set RMObj = Nothing

    Set VBISApp = Nothing


    Exit Sub
```

```
ErrorHandler:
 MsgBox (Err.Description)
 Set VRObj = Nothing
 Set RMObj = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISRecipe3.UpdateMaster

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8

Set VBISApp = CreateObject("Intellution.VBIS.8")

Dim RecipeID As String
Dim RecipeVersion As Long

Dim RMObj As VBISRecipeManagement3
Dim VRObj As VBISRecipe3

Set RMObj = VBISApp.VBISRecipeManagement3
Set VRObj = RMObj.VBISRecipe3

RecipeID = "MAKE_TOOTHPASTE"
RecipeVersion = 1

VRObj.UpdateMaster RecipeID, RecipeVersion

Set VRObj = Nothing
Set RMObj = Nothing
Set VBISApp = Nothing
```

```
Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VRObj = Nothing
 Set RMObj = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISRecipe3.Verify

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8

Set VBISApp = CreateObject("Intellution.VBIS.8")

Dim RecipeID As String
Dim RecipeVersion As Long

Dim RMObj As VBISRecipeManagement3
Dim VRObj As VBISRecipe3

Set RMObj = VBISApp.VBISRecipeManagement3
Set VRObj = RMObj.VBISRecipe3

RecipeID = "MAKE_TOOTHPASTE"
RecipeVersion = 1

VRObj.Verify RecipeID, RecipeVersion
Set VRObj = Nothing
Set RMObj = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)
 Set VRObj = Nothing
 Set RMObj = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISRecipe3.RebuildRecipeDir

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISRCPMAN As VBISRecipeManagement3
Dim VBISRCP As VBISRecipe3

Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISRCPMAN = VBISApp.VBISRecipeManagement3
Set VBISRCP = VBISRCPMAN.VBISRecipe3

VBISRCP.RebuildRecipeDir

Set VBISRCP = Nothing
Set VBISRCPMAN = Nothing
Set VBISApp = Nothing

Exit Sub

ErrorHandler:
 MsgBox (Err.Description)

 Set VBISRCP = Nothing

 Set VBISRCPMAN = Nothing

 Set VBISApp = Nothing


End Sub
```

VBISRecipe3.AddRecipe, VBISRecipe3.VBISRecipeHeader2

```
On Error GoTo ErrorHandler


Dim VBISApp As VBIS8

Dim VBISSRVR As VBISServer8

Dim VBISRCPMAN As VBISRecipeManagement3

Dim VBISRCPHDR As VBISRecipeHeader2

Dim VBISRCP As VBISRecipe3

Dim AddRecipeID As String

Dim AddRecipeVersion As Long




AddRecipeID = "MAKE_TOOTHPASTE"

AddRecipeVersion = 1

Set VBISApp = CreateObject("Intellution.VBIS.8")

Set VBISRCPMAN = VBISApp.VBISRecipeManagement3

Set VBISRCP = VBISRCPMAN.VBISRecipe3


VBISRCP.AddRecipe AddRecipeID, AddRecipeVersion


Set VBISRCP = Nothing

Set VBISRCPMAN = Nothing

Set VBISApp = Nothing


Exit Sub
```

```
ErrorHandler:
 MsgBox (Err.Description)
 Set VBISRCP = Nothing
 Set VBISRCPMAN = Nothing
 Set VBISApp = Nothing

End Sub
```

VBISRecipe3.GetRecipeHeader

```
On Error GoTo ErrorHandler

Dim VBISApp As VBIS8
Dim VBISSRVR As VBISServer8
Dim VBISRCPMAN As VBISRecipeManagement3
Dim VBISRCPHDR As VBISRecipeHeader2
Dim VBISRCP As VBISRecipe3
Dim RecipeID As String
Dim RecipeVersion As Long
Dim Name As String



RecipeID = "MAKE_TOOTHPASTE"
RecipeVersion = 1
Set VBISApp = CreateObject("Intellution.VBIS.8")
Set VBISRCPMAN = VBISApp.VBISRecipeManagement3
Set VBISRCP = VBISRCPMAN.VBISRecipe3
Set VBISRCPHDR = VBISRCP.GetRecipeHeader(RecipeID, RecipeVersion)
Name = VBISRCPHDR.RecipeID

Set VBISRCPHDR = Nothing
Set VBISRCP = Nothing
Set VBISRCPMAN = Nothing
Set VBISApp = Nothing
```

```
Exit Sub


ErrorHandler:
 MsgBox (Err.Description)
 Set VBISRCPHDR = Nothing
 Set VBISRCP = Nothing
 Set VBISRCPMAN = Nothing
 Set VBISApp = Nothing


End Sub
```

VBISEnumerations: CountEnumSet, NextEnumSet, QueryEnumSet, GetCountEnum, GetNextEnum, QueryEnum, GetDefaultEnum

```
On Error GoTo ErrorHandler
Dim VBISApp As VBIS8
Set VBISApp = CreateObject("Intellution.VBIS.8")
Dim lCount As Long
Dim lSetCount As Long
Dim i As Integer
Dim strEnumSet As String
Dim strEnum As String
Dim VEObj As VBISEquipment
Dim ENObj As VBISEnumerations
Set VEObj = VBISApp.VBISEquipment
Set ENObj = VEObj.VBISEnumerations
ENObj.QueryEnumSet
lSetCount = ENObj.CountEnumSet
If (lSetCount > 0) Then
strEnumSet = ENObj.NextEnumSet
ENObj.QueryEnum (strEnumSet)
lCount = ENObj.GetCountEnum(strEnumSet)
If lCount > 0 Then
```

```
strDefEnum = ENObj.GetDefaultEnum(strEnumSet)

For i = 1 To lCount

strEnum = ENObj.GetNextEnum(strEnumSet)

If (strEnum = strDefEnum) Then

Exit For

End If

Next i

End If

End If


Set ENObj = Nothing

Set VEObj = Nothing

Set VBISApp = Nothing


Exit Sub


ErrorHandler:

MsgBox (Err.Description)

Set ENObj = Nothing

Set VEObj = Nothing

Set VBISApp = Nothing


End Sub
```

C++ Examples

VBISBatchControl5.Add

```
HRESULT hr;

IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISServer interface from IVBIS

VBISServer8* pVBISServer = NULL;

hr = pIVBIS->QueryInterface (IID_VBISServer8,
       (void**)&pVBISServer);
if (SUCCEEDED (hr))

{

 // Get a pointer to VBISBatchControl5 interface from VBISServer8

 VBISBatchControl5* pVBISBatchControl = NULL;

 hr = pVBISServer->get_VBISBatchControl5

  ((VBISBatchControl5**)&pVBISBatchControl);

 if (SUCCEEDED (hr))

 {

  // Set the values for Add method

  CString strRecipeID = "MAKE_TOOTHPASTE";

  LONG lCampaignID = 123L

  BSTR bsRecipeID = strRecipeID.AllocSysString();

  LONG lRecipeVersion = 1L;

  CString strBatchID = "Batch001";

  BSTR bsBatchID = strBatchID.AllocSysString();

  FLOAT lBatchScaling = 100.0f;

  CString strParmBind = "";

  BSTR bsParmBind = strParmBind.AllocSysString();

  CString strUnitBind = "";

  BSTR bsUnitBind = strUnitBind.AllocSysString();

  CString strCurrentUser = “Gary”;

  BSTR bsCurrentUser = strCurrentUser.AllocSysString();

  LONG lUseDefaultBindings = 3L;

  LONG lOpInteraction = 1L;

  LONG lOpBindParameters = 1L;

  LONG lOpBindUnits = 1L;

  LONG lBatchUniqueID = 0L; // Value will be set by Add() method

  VARIANT varSecurity;

  VariantInit(&varSecurity);


  // Schedule the Batch

  hr = pVBISBatchControl->Add (
       lCampaignID
       bsRecipeID,
       lRecipeVersion,
       bsBatchID,
       lBatchScaling,
       bsUnitBind,
       bsParmBind,
       lUseDefaultBindings,
       lOpInteraction,
       lOpBindParameters,
       lOpBindUnits,
       &lBatchUniqueID,
       bsCurrentUser,
       varSecurity);
  ::SysFreeString (bsRecipeID);
  ::SysFreeString (bsBatchID);
  ::SysFreeString (bsUnitBind);
  ::SysFreeString (bsParmBind);
  ::SysFreeString (bsCurrentUser);

  if (SUCCEEDED (hr))
  {
   // Display the results
   CString strTemp;
   strTemp.Format ("%ld", lBatchUniqueID);
   SetOutput ("BatchControl5.Add()", strTemp, hr);
  }
  else
  {
   // Display the negative results
   SetOutput ("BatchControl5.Add()", "ERROR", hr);
  }
```

```
  pVBISBatchControl->Release ();
 }
 else
 {
  SetOutput ("BatchControl5.QueryInterface()", "ERROR", hr);
 }

 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISBatchControl5.Bind

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();


// Get a pointer to VBISServer interface from IVBIS8
VBISServer8* pVBISServer = NULL;
hr = pIVBIS->QueryInterface (IID_VBISServer8,
       (void**)&pVBISServer);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISBatchControl5 interface from VBISServer8
 VBISBatchControl5* pVBISBatchControl = NULL;
 hr = pVBISServer->get_VBISBatchControl5
  ((VBISBatchControl5**)&pVBISBatchControl);
 if (SUCCEEDED (hr))
 {
  // Set the values for Add method
  LONG lBatchUniqueID = 167L;
  CString strParmBind = ""; // No Parameter Binding
```

```
BSTR bsParmBind = strParmBind.AllocSysString();

CString strUnitBind = "ADDITIVE:1\tMIX1";

BSTR bsUnitBind = strUnitBind.AllocSysString();

LONG lBindings = 1L; // Unit Binding

CString strCurrentUser = “Gary”;

BSTR bsCurrentUser = strCurrentUser.AllocSysString();

VARIANT varSecurity;

VariantInit(&varSecurity);


// Bind the Batch

hr = pVBISBatchControl->Bind (lBatchUniqueID,

     bsUnitBind,

     bsParmBind,

     lBindings,

     bsCurrentUser,

     varSecurity);


::SysFreeString (bsUnitBind);

::SysFreeString (bsParmBind);

::SysFreeString (bsCurrentUser);

if (SUCCEEDED (hr))

{

 // Display the results

 CString strTemp;

 strTemp.Format ("%ld", lBatchUniqueID);

 SetOutput ("BatchControl5.Bind()", strTemp, hr);

}

else

{

 // Display the negative results

 SetOutput ("BatchControl5.Bind()", "ERROR", hr);

}
```

```
  pVBISBatchControl->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("BatchControl5.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISBatchControl5.State

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISServer8 interface from IVBIS
VBISServer8* pVBISServer = NULL;
hr = pIVBIS->QueryInterface (IID_VBISServer8,
       (void**)&pVBISServer);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISBatchControl5 interface from VBISServer8
 VBISBatchControl5* pVBISBatchControl = NULL;
 hr = pVBISServer->get_VBISBatchControl5
  ((VBISBatchControl5**)&pVBISBatchControl);
 if (SUCCEEDED (hr))
 {
  // Set the values for Bind method
  LONG lBatchUniqueID = 166L;
  LONG lBatchState = 0L;
  // the Batch
  hr = pVBISBatchControl->State (lBatchUniqueID, &lBatchState);

  if (SUCCEEDED (hr))
  {
   // Display the results
   CString strTemp;
   strTemp.Format ("%ld", lBatchState);
   SetOutput ("BatchControl5.State()", strTemp, hr);
  }
  else
  {
   // Display the negative results
   SetOutput ("BatchControl5.State()", "ERROR", hr);
  }
  pVBISBatchControl->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("BatchControl5.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISBatchControl5.Command

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();


// Get a pointer to VBISServer8 interface from IVBIS8
VBISServer8* pVBISServer = NULL;
hr = pVBISServer->get_VBISBatchControl5
 (VBISBatchControl5**)&pVBISBatchControl);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISBatchControl interface from VBISServer
 VBISBatchControl5* pVBISBatchControl = NULL;
 hr = pVBISServer->get_VBISBatchControl5
  ((VBISBatchControl5**)&pVBISBatchControl);
 if (SUCCEEDED (hr))
 {
  // Set the values for Bind method
  LONG lBatchUniqueID = 166L;
  CString strCommand = "START";
  BSTR bsCommand = strCommand.AllocSysString();
  CString strCurrentUser = “Gary”;
  BSTR bsCurrentUser = strCurrentUser.AllocSysString();
  VARIANT varSecurity;
  VariantInit(&varSecurity);


  // the Batch
  hr = pVBISBatchControl->Command (lBatchUniqueID,
         bsCommand,
         bsCurrentUser,
         varSecurity);
  ::SysFreeString (bsCommand);
  ::SysFreeString (bsCurrentUser);
  if (SUCCEEDED (hr))
  {
   // Display the results
   SetOutput ("BatchControl5.Command()", "OK", hr);
  }
  else
  {
   // Display the negative results
   SetOutput ("BatchControl5.Command()", "ERROR", hr);
  }
  pVBISBatchControl->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("BatchControl5.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISBatchControl5.SetParameter

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();
// Get a pointer to VBISServer8 interface from IVBIS
VBISServer8* pVBISServer = NULL;
hr = pVBISServer->get_VBISBatchControl5
 (VBISBatchControl5**)&pVBISBatchControl);
if (SUCCEEDED (hr))
{
```

```
// Get a pointer to VBISBatchControl5 interface from VBISServer8
VBISBatchControl5* pVBISBatchControl = NULL;
hr = pVBISServer->get_VBISBatchControl5
 ((VBISBatchControl5**)&pVBISBatchControl);
if (SUCCEEDED (hr))
{
 CString strPhaseID = "34\tBASE:1\tMAKE_BASE:1\tADD_INGS:1";
 BSTR bsPhaseID = strPhaseID.AllocSysString();
 CString strParameterName = "FLAVOR_AMT";
 BSTR bsParameterName = strParameterNameID.AllocSysString();
 CString strValue = "44";
 BSTR bsValue = strValue.AllocSysString();
 CString strCurrentUser = "44";
 BSTR bsCurrentUser = strCurrentUser.AllocSysString();
 VARIANT varSecurity;
 VariantInit(&varSecurity);

 hr = pVBISBatchControl->SetParameter (bsPhaseID,
         bsParameterName,
         bsValue,
         bsCurrentUser,
         varSecurity);
 ::SysFreeString (bsPhaseID);
 ::SysFreeString (bsParameterName);
 ::SysFreeString (bsValue);
 ::SysFreeString (bsCurrentUser);
 if (SUCCEEDED (hr))
 {
  // Display the results
  SetOutput ("BatchControl5.SetParameter()", "OK", hr);
 }
 else
```

```
        {
         // Display the negative results
         SetOutput ("BatchControl5.SetParameter()", "ERROR", hr);
        }
        pVBISBatchControl->Release ();
       }
       else
       {
        // Display the negative results
        SetOutput ("BatchControl5.QueryInterface()", "ERROR", hr);
       }
       pVBISServer->Release ();
      }
      else
      {
       SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
      }
     }
```

VBISBatchList: Count, Next, and Query

The following subroutine shows an example of refreshing and iterating through the batch list.

```
HRESULT hr;


IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();


// Get a pointer to VBISServer8 interface from IVBIS8
VBISServer8* pVBISServer = NULL;
hr = pIVBIS->QueryInterface (IID_VBISServer8,
      (void**)&pVBISServer);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISBatchList interface from VBISServer8
```

```
VBISBatchList* pVBISBatchList = NULL;
hr = pVBISServer->get_VBISBatchList ((VBISBatchList**)&pVBISBatchList);

if (SUCCEEDED (hr))
{

 // Query for all Batches
 hr = pVBISBatchList->Query ();

 if (SUCCEEDED (hr))
 {
  long lCount = 0L;
  hr = pVBISBatchList->get_Count (&lCount);

  if (SUCCEEDED (hr))
  {
   for (int i=0; i<lCount; i++)
   {
    VARIANT varNext; // Variant to receive one Batch list
       // all data associated with the batch

    // Initialize our 'Next' Variant
    VariantInit (&varNext);

    // Ask the VBIS Batch list for the next Batch record
    hr = pVBISBatchList->get_Next (&varNext);

    if (SUCCEEDED (hr))
    {

     // Extract the Safe Array data out of the Variant
     SAFEARRAY* psaData;
     psaData = varNext.parray;

     VARIANT varData;
     VariantInit (&varData);

     // Assign Batch ID
     long lIndex = 0L;
     SafeArrayGetElement (psaData, &lIndex, &varData);
     CString strBatchID = varData.bstrVal;
     VariantClear (&varData);

     // Clear Variant and free all associated data
     VariantClear (&varNext);
    }
    else
    {
     SetOutput ("BatchList.Next()", "ERROR", hr);
    }
   } // for
  }
  else
  {
   SetOutput ("BatchList.Count()", "ERROR", hr);
  }
 }
 else
 {
  SetOutput ("BatchList.Query()", "ERROR", hr);
 }
 pVBISBatchList->Release ();
}
else
```

```
 {
  SetOutput ("BatchList.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISAlarmsList: Count, Next, Query

The following subroutine shows an example of refreshing and iterating through the alarm list.

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISServer8 interface from IVBIS8
VBISServer8* pVBISServer = NULL;
hr = pIVBIS->QueryInterface (IID_VBISServer8,
      (void**)&pVBISServer);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISAlarmsList interface from VBISServer8
 VBISAlarmsList* pVBISAlarmsList = NULL;
 hr = pVBISServer->get_VBISAlarmsList
((VBISAlarmsList**)&pVBISAlarmsList);

 if (SUCCEEDED (hr))
 {
  hr = pVBISAlarmsList->Query ();
  if (SUCCEEDED (hr))
  {
   long lCount;
   hr = pVBISAlarmsList->get_Count (&lCount);
```

```
if (SUCCEEDED (hr))
{
 for (int i=0; i<lCount; i++)
 {
  VARIANT varNext; // Variant to receive one alarm list
     // & all data associated with the alarm

  // Initialize our 'Next' Variant
  VariantInit (&varNext);

  // Ask the VBIS Alarm list for the next alarm record
  hr = pVBISAlarmsList->get_Next (&varNext);
  if (SUCCEEDED (hr))
  {
   // Extract the Safe Array data out of the Variant
   SAFEARRAY* psaData;
   psaData = varNext.parray;

   // Extract Alarm Values
   VARIANT varData;
   VariantInit (&varData);

   // Assign Phase ID
   long lIndex = 0L;
   SafeArrayGetElement (psaData, &lIndex, &varData);
   CString strPhaseID = varData.bstrVal;
   VariantClear (&varData);
  }
  else
  {
   SetOutput ("VBISServer8.Next()", "ERROR", hr);
  }
```

```
            } // for
           }
           else
           {
            SetOutput ("VBISServer8.Count()", "ERROR", hr);
           }
          }
          else
          {
           SetOutput ("VBISServer8.Query()", "ERROR", hr);
          }
          pVBISAlarmsList->Release ();
         }
         else
         {
          SetOutput ("AlarmsList.QueryInterface()", "ERROR", hr);
         }
         pVBISServer->Release ();
        }
        else
        {
         SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
        }
```

VBISPromptList2 : Count, Next, Query, Acknowledge

The following subroutine shows an example of refreshing and iterating through the prompt list and acknowledging the prompts.

```
        HRESULT hr;


        IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();


        // Get a pointer to VBISServer8 interface from IVBIS8

        VBISServer8* pVBISServer = NULL;
```

```
hr = pIVBIS->QueryInterface (IID_VBISServer8,

      (void**)&pVBISServer);

if (SUCCEEDED (hr))

{

 // Get a pointer to VBISPromptList2 interface from VBISServer8

 VBISPromptList2* pVBISPromptList2 = NULL;

 hr = pVBISServer->get_VBISPromptList2
((VBISPromptList2**)&pVBISPromptList2);


 if (SUCCEEDED (hr))

 {


  hr = pVBISPromptList2->Query ();


  if (SUCCEEDED (hr))

  {

   long lCount = 0L;

   hr = pVBISPromptList2->get_Count (&lCount);

   if (SUCCEEDED (hr))

   {

    for (int i=0; i<lCount; i++)

    {


     VARIANT varNext; // Variant to receive one Recipe list

        // & all data associated with the recipe


     // Initialize our 'Next' Variant

     VariantInit (&varNext);


     // Ask the VBIS8 prompt list for the next prompt

     hr = pVBISPromptList2->get_Next (&varNext);

     if (SUCCEEDED (hr))

     {
      // Extract the Safe Array data out of the Variant
      SAFEARRAY* psaData;
      psaData = varNext.parray;

      // Respond to the prompt. In this example, we respond
      // with 0. In an actual application, you must respond
      // with a valid response
      CString strPromptResponse = "0";
      BSTR bsPromptResponse = strPromptResponse.AllocSysString();
      CString strCurrentUser = "Gary";
      BSTR bsCurrentUser = strCurrentUser.AllocSysString();

      VARIANT varData;
      VariantInit (&varData);

      // Get Prompt ID
      long lIndex = 11L;
      SafeArrayGetElement (psaData, &lIndex, &varData);
      CString strPromptID = varData.bstrVal;
      long lPromptID = atol (strPromptID);
      VariantClear (&varData);

      VARIANT varSecurity;
      VariantInit(&varSecurity);
      hr = pVBISPromptList2->Acknowledge (lPromptID,
             bsPromptResponse,
bsCurrentUser,
varSecurity);
      ::SysFreeString (bsPromptResponse);
      ::SysFreeString (bsCurrentUser);
      if (SUCCEEDED (hr))
```

```
      {
       SetOutput("PromptList.Acknowledge()","OK", hr);
      }
      else
      {
       SetOutput ("PromptList.Acknowledge()","ERROR", hr);
      }

     }
     else
     {
      SetOutput ("PromptList.Next()", "ERROR", hr);
     }
    } // for
   }
   else
   {
    SetOutput ("PromptList.Count()", "ERROR", hr);
   }
  }
  else
  {
   SetOutput ("PromptList.Query()", "ERROR", hr);
  }
  pVBISPromptList2->Release ();
 }
 else
 {
  SetOutput ("PromptList.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
```

```
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISRecipeList3: Count, Next, Query

This following subroutine shows an example of refreshing and iterating through the recipe list.

```
RESULT hr;

IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISServer interface from IVBIS
VBISServer8* pVBISServer = NULL;
hr = pIVBIS->QueryInterface (IID_VBISServer8,
      (void**)&pVBISServer);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISRecipeList3 interface from VBISServer8
 VBISRecipeList3* pVBISRecipeList = NULL;
 hr = pVBISServer->get_VBISRecipeList3
  ((VBISRecipeList3**)&pVBISRecipeList);

 if (SUCCEEDED (hr))
 {

  // Query for all Recipes
  hr = pVBISRecipeList->Query ();

  if (SUCCEEDED (hr))
  {
   long lCount = 0L;
   hr = pVBISRecipeList->get_Count (&lCount);
```

```
if (SUCCEEDED (hr))
{
 for (int i=0; i<lCount; i++)
 {
  VARIANT varNext; // Variant to receive one Recipe list
     // & all data associated with the recipe

  // Initialize our 'Next' Variant
  VariantInit (&varNext);

  // Ask the VBIS recipe list for the next recipe record
  hr = pVBISRecipeList->get_Next (&varNext);

  if (SUCCEEDED (hr))
  {

   // Extract the Safe Array data out of the Variant
   SAFEARRAY* psaData;
   psaData = varNext.parray;

   VARIANT varData;
   VariantInit (&varData);

   // Assign Recipe ID
   long lIndex = 0L;
   SafeArrayGetElement (psaData, &lIndex, &varData);
   CString strRecipeID = varData.bstrVal;
   VariantClear (&varData);

   // Clear Variant and free all associated data
   VariantClear (&varNext);
```

```
     }
     else
     {
      SetOutput ("RecipeList3.Next()", "ERROR", hr);
     }
    } // for
   }
   else
   {
    SetOutput ("RecipeList3.Count()", "ERROR", hr);
   }
  }
  else
  {
   SetOutput ("RecipeList3.Query()", "ERROR", hr);
  }
  pVBISRecipeList->Release ();
 }
 else
 {
  SetOutput ("RecipeList3.QueryInterface()", "ERROR", hr);
 }
 pVBISServer->Release ();
}
else
{
 SetOutput ("VBISServer8.QueryInterface()", "ERROR", hr);
}
```

VBISRecipe3.ResetControl

```
HRESULT hr;

IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISRecipeManagement3 interface from IVBIS8

VBISRecipeManagement3* pVBISRecipeManagement = NULL;

hr = pIVBIS->QueryInterface (IID_VBISRecipeManagement3,

       (void**)&pVBISRecipeManagement);

if (SUCCEEDED (hr))

{

 // Get a pointer to VBISRecipe3 interface from VBISRecipeManagement3

 VBISRecipe* pVBISRecipe = NULL;

 hr = pVBISRecipeManagement->get_VBISRecipe3
((VBISRecipe3**)&pVBISRecipe);


 if (SUCCEEDED (hr))

 {


  // Set the values for the recipe

  CString strRecipeID = "MAKE_TOOTHPASTE";

  BSTR bsRecipeID = strRecipeID.AllocSysString ();

  long lRecipeVersion = 1L;

  hr = pVBISRecipe->ResetControl (bsRecipeID,

        lRecipeVersion);

  ::SysFreeString (bsRecipeID);

  if (SUCCEEDED (hr))

  {

   // Display the results

   SetOutput ("Recipe2.ResetControl()", "OK", hr);

  }

  else

  {

   // Display the negative results

   SetOutput ("Recipe2.ResetControl()", "ERROR", hr);
  }
  pVBISRecipe->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("Recipe2.QueryInterface()", "ERROR", hr);
 }
 pVBISRecipeManagement->Release ();
}
else
{
 SetOutput ("VBISRecipeManagement3.QueryInterface()", "ERROR", hr);
}
```

VBISRecipe3.UpdateMaster

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISRecipeManagement3 interface from IVBIS8
VBISRecipeManagement3* pVBISRecipeManagement = NULL;
hr = pIVBIS->QueryInterface (IID_VBISRecipeManagement3,
      (void**)&pVBISRecipeManagement);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISRecipe3 interface from VBISRecipeManagement3
 VBISRecipe3* pVBISRecipe = NULL;
 hr = pVBISRecipeManagement->get_VBISRecipe3
((VBISRecipe3**)&pVBISRecipe);

 if (SUCCEEDED (hr))
 {
  // Set the values for the recipe
  CString strRecipeID = "MAKE_TOOTHPASTE";
  BSTR bsRecipeID = strRecipeID.AllocSysString ();
  long lRecipeVersion = 1L;
  hr = pVBISRecipe->UpdateMaster (bsRecipeID,
        lRecipeVersion);
  ::SysFreeString (bsRecipeID);
  if (SUCCEEDED (hr))
  {
   // Display the results
   SetOutput ("Recipe2.UpdateMaster()", "OK", hr);
  }
  else
  {
   // Display the negative results
   SetOutput ("Recipe2.UpdateMaster()", "ERROR", hr);
  }
  pVBISRecipe->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("Recipe2.QueryInterface()", "ERROR", hr);
 }
 pVBISRecipeManagement->Release ();
}
else
{
 SetOutput ("VBISRecipeManagement3.QueryInterface()", "ERROR", hr);
}
```

VBISRecipe3.Verify

```
HRESULT hr;

IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();


// Get a pointer to VBISRecipeManagement3 interface from IVBIS8

VBISRecipeManagement3* pVBISRecipeManagement = NULL;

hr = pIVBIS->QueryInterface (IID_VBISRecipeManagement3,

       (void**)&pVBISRecipeManagement);

if (SUCCEEDED (hr))

{

 // Get a pointer to VBISRecipe3 interface from VBISRecipeManagement3

 VBISRecipe3* pVBISRecipe = NULL;

 hr = pVBISRecipeManagement->get_VBISRecipe3
((VBISRecipe3**)&pVBISRecipe);


 if (SUCCEEDED (hr))

 {


  // Set the values for the recipe

  CString strRecipeID = "MAKE_TOOTHPASTE";

  BSTR bsRecipeID = strRecipeID.AllocSysString ();

  long lRecipeVersion = 1L;

  hr = pVBISRecipe->Verify (bsRecipeID,

       lRecipeVersion);

  ::SysFreeString (bsRecipeID);

  if (SUCCEEDED (hr))

  {

   // Display the results

   SetOutput ("Recipe2.Verify()", "OK", hr);

  }

  else

  {

   // Display the negative results
```

```
    SetOutput ("Recipe2.Verify()", "ERROR", hr);

   }

   pVBISRecipe->Release ();

  }

  else

  {

   // Display the negative results

   SetOutput ("Recipe2.QueryInterface()", "ERROR", hr);

  }

  pVBISRecipeManagement->Release ();

 }

 else

 {

  SetOutput ("VBISRecipeManagement3.QueryInterface()", "ERROR", hr);

 }
```

VBISEnumerations: CountEnumSet, NextEnumSet, QueryEnumSet, GetCountEnum, GetNextEnum, QueryEnum, GetDefaultEnum

The following subroutine shows an example of refreshing and iterating through an enumeration list.

```
HRESULT hr;
IVBIS8* pIVBIS = getVBISMFCDlg ()->getIVBIS ();

// Get a pointer to VBISEquipment interface from IVBIS8
VBISEquipment* pVBISEquipment = NULL;
hr = pIVBIS->QueryInterface (IID_VBISEquipment,
      (void**)&pVBISEquipment);
if (SUCCEEDED (hr))
{
 // Get a pointer to VBISEnumerations interface from VBISEquipment
 VBISEnumerations* pVBISEnumerations = NULL;
 hr = pVBISEquipment->get_VBISEnumerations
    ((VBISEnumerations**)&pVBISEnumerations);

 if (SUCCEEDED (hr))
 {
  // Query for all the enumeration sets
  hr = pVBISEnumerations->QueryEnumSet ();
  if (SUCCEEDED (hr))
  {
   // Count all the enumeration sets
   long lSetCount = 0L;
   hr = pVBISEnumerations->get_CountEnumSet (&lSetCount);
   if (SUCCEEDED (hr))
   {
```

```
if (lSetCount > 0)
{
 // Get first enumeration set
 BSTR bsEnumSet;
 hr = pVBISEnumerations->get_NextEnumSet (&bsEnumSet);
 if (SUCCEEDED (hr))
 {
  // Query for all the enumeration values within the enumeration set
  hr = pVBISEnumerations->QueryEnum (bsEnumSet);
  if (SUCCEEDED (hr))
  {
   // Count all the enumeration values within the enumeration set
   long lCount = 0L;
   hr = pVBISEnumerations->GetCountEnum (bsEnumSet,
          &lCount);
   if (SUCCEEDED (hr))
   {
    // Get the default value within the enumeration set
    BSTR bsEnumDefaultValue;
    hr = pVBISEnumerations->GetDefaultEnum (bsEnumSet,
             &bsEnumDefaultValue);
    if (SUCCEEDED (hr))
    {
     for (int i=0; i<lCount; i++)
     {
      // Get the next value within the Enumeration set
      BSTR bsEnumValue;
      hr = pVBISEnumerations->GetNextEnum (bsEnumSet,
             &bsEnumValue);
      if (SUCCEEDED (hr))
      {
       // Test if this enumeration is the default enumeration
       CString strEnumValue = bsEnumValue;
       CString strEnumDefaultValue = bsEnumDefaultValue;
       if (strEnumValue == strEnumDefaultValue)
       {
        break;
       }
      }
      else
      {
       // Display the negative results
       SetOutput ("Enumerations.GetNextEnum()", "ERROR", hr);
      }
     } // for
    }
    else
    {
     // Display the negative results
     SetOutput ("Enumerations.GetDefaultEnum()", "ERROR", hr);
    }
   }
   else
   {
    // Display the negative results
    SetOutput ("Enumerations.GetCountEnum()", "ERROR", hr);
   }
  }
```

```
      else
      {
       // Display the negative results
       SetOutput ("Enumerations.QueryEnum()", "ERROR", hr);
      }
     }
     else
     {
      // Display the negative results
      SetOutput ("Enumerations.NextEnumSet()", "ERROR", hr);
     }
    } // lSetCount > 0
   }
   else
   {
    // Display the negative results
    SetOutput ("Enumerations.CountEnumSet()", "ERROR", hr);
   }
  }
  else
  {
   // Display the negative results
   SetOutput ("Enumerations.QueryEnumSet()", "ERROR", hr);
  }
  pVBISEnumerations->Release ();
 }
 else
 {
  // Display the negative results
  SetOutput ("VBISEnumerations.QueryInterface()", "ERROR", hr);
 }
 pVBISEquipment->Release ();
}
else
{
 // Display the negative results
 SetOutput ("VBISEquipment.QueryInterface()", "ERROR", hr);
}
```

Error-Handling

The VBIS error descriptions explain the meaning of each error generated by the VBIS object interface. The general steps you can do to correct the error and the calls(s) that can generate the error are also listed.

VBIS returns error codes as a custom HRESULT. How you receive and handle this code depends on whether you are programming in C++ or Visual Basic:

Error Handling within C++ Programs

To make use of VBIS returned errors within C++, implement an error strategy that uses the ErrorInfo object. When calling a VBIS method, use an error handler to monitor for any errors.

The following is an example of how to handle errors within Visual C++:

```
HRESULT hr = interfaceobjectpointer->method (parameterlist);

if (FAILED(hr))
{
 IErrorInfo *pIErrorInfo = NULL;
 HRESULT hr = GetErrorInfo (NULL, &pIErrorInfo);
 CString strError;
 if ((SUCCEEDED (hr)) && (pIErrorInfo != NULL))
 {
  BSTR bsError = NULL;
  pIErrorInfo->GetDescription (&bsError);
  strError = bsError;
 }
 else
 {
  strError = "Fatal VBIS Error";
 }

 AfxMessageBox (strError);
}
```

Error Handling within Visual Basic Programs

To make use of error handling within Visual Basic, you can implement an error strategy that uses the Visual Basic built-in error object (err). When calling a VBIS method, use an error handler to monitor all returns. All of the Visual Basic examples provided in this help system implement this basic error strategy.

To make use of VBIS returned errors within Visual Basic, implement an error strategy that uses the Visual Basic built-in error object (err). When calling a VBIS method, use an error handler to monitor for any errors. The following is an example of how to handle errors within Visual Basic:

```
On Error GoTo ErrorHandler
 :
 :
Interfaceobject.method(parameterlist);
 :
 :
Exit Sub
ErrorHandler:
Dim lmyError As Long
If (Err.Number > 0) And (Err.Number < 65535) Then ' System Error
lmyError = Err.Number
Else
lmyError = Err.Number - vbObjectError ' VBIS Application Error
End If
MsgBox Err.Description, vbOKOnly, “VBIS Error” ‘ Display the Error
End Sub
```

Success and Error Codes Listing

To make the number meaningful, write error-checking routines that evaluate the success and error codes and display the appropriate message box. Use the following list to determine the meaning of each error code.

Error Code	Meaning
0	VBIS_SUCCESS
2	VBIS_INIT_COMPLETE
6	VBIS_CLEANUP_COMPLETE
1001	VBIS_ERROR
1003	VBIS_FAILED_TO_INITIALIZE
1004	VBIS_FAILED_TO_CONNECT
1005	VBIS_CLEANUP_FAILED
1007	VBIS_BAD_PTR
1008	VBIS_NO_RECIPE
1009	VBIS_INVALID_VERSION
1010	VBIS_NO_BATCH
1011	VBIS_BAD_STATE
1012	VBIS_OUT_OF_MEMORY
1013	VBIS_BAD_VAR_TYPE
1014	VBIS_SUB_OUT_OF_RANGE
1015	VBIS_BAD_ARG
1202	VBIS_SS_BAD_UNIT_BIND

Error Code	**Meaning**
1203	VBIS_SS_BAD_PARM_BIND
1204	VBIS_SS_NO_BIND_UP
1205	VBIS_SS_NO_BIND_UNIT
1206	VBIS_SS_NO_BIND_PARM
1207	VBIS_SS_UP_BIND
1208	VBIS_SS_UNIT_BIND
1209	VBIS_SS_PARM_BIND
1210	VBIS_SS_INVALID_FLAG
1211	VBIS_SS_SCALE_OUT_OF_RANGE
212	VBIS_SS_BATCH_BOUND
1213	VBIS_SS_MISMATCH_BIND
1400	VBIS_BS_BAD_COMMAND
1401	VBIS_BS_NO_UP_BIND
1402	VBIS_BS_NO_UNIT_BIND
1403	VBIS_BS_NO_PARM_BIND
1600	VBIS_PS_NO_PROMPT

Troubleshooting VBIS

Use the following to help troubleshoot VBIS issues:

- Using the VBIS log
- "READOPCSTREAM" error in VBIS Log

Using the VBIS log

The VBIS log file (vbis.log) can be a useful tool for troubleshooting connections. This file resides in the Batch Execution Log directory.

"READOPCSTREAM failed!!" error in VBIS log

You may notice the following error in the VBIS.log file: "READOPCSTREAM failed!!". When VBIS requests data from the Batch server, the server returns a 'readopcstream' object. When VBIS fails to read this object it will log an error message; however, it does not state what failed and where. To verify, look for a match between errors in the VBIS.log and VBEXEC.log files.

Glossary

Active Binding

Proficy Batch supports Active Binding, which allows Batch to bind and re-bind units at multiple stages in a batch's life cycle including when a batch is created, started, or in production. Recipe authors can configure recipes to automatically allocate equipment to batches based on (1) the properties of the equipment entities and (2) the real-time conditions on the plant floor.

Area Model

A database that contains the definitions of the process cells, units, and equipment phases that represent a physical, geographical, or logical grouping of equipment used to build and execute recipes. Typically, your area model contains all of the equipment at your plant.

Collection

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. The VBIS area model is made up of objects that represent S88.01 entities, such as process cells, units, and equipment phases. VBIS groups these objects together as collections based upon the class of the object. A class represents a collection, and the items of that class are the instances within the area model. 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.

Control Module

Consists of sensors and other control modules that together perform a specific task. Control modules perform regulatory or state control over their constituent parts.

Destination Unit

The unit where the equipment pathing connection ends. For example, if a reactor feeds into a fermenter, the reactor is the origin unit and the fermenter is the destination unit.

Enumeration

A list of strings that can be referenced by their ordinal offset in a list.

Example: Sunday=0, Monday=1, Tuesday=2.

Enumeration Set

A logical grouping of enumerations.

Equipment ID

A unique ID that is assigned to all equipment configured in the Equipment Editor. This ID is used to acquire and release resources. It must match the equipment ID used by the phase logic in the process controller.

Equipment Phase

A phase that is part of the equipment control. The logic for an equipment phase resides in the process controller.

Equipment Phase Tags

A phase that is part of the equipment control. The logic for an equipment phase resides in the process controller.

Formulation Header

Administrative information about the formulation. This information includes the version number, version date, and author.

Global Formulation Header

Contains the set of parameters that are constant for all product formulations. The global formulation header is optional and there can be only one per recipe.

Manifold Object

A control module that is used to connect multiple units as part of the area model's equipment pathing.

Maximum Owners

Identifies the maximum number of owners that can simultaneously own an equipment module. It is used to arbitrate resources and is typically set to one to allow only one owner at a time.

Object Expressions

An expression that specifies a particular object. This expression can include any of the object's containers. For example, if your application has an Application object that contains a Document object that contains a Text object, the following are valid object expressions:

```
Application.Document.Text

Application.Text

Document.Text

Text
```

OPC Item

A named data structure accessed through OPC (OLE for Process Control).

Operator Message

Identifies a string that is sent to the operator when the phase executes. The message ID must correspond with the ID used by the phase logic.

Phase Report

Reports that detail actual process values or batch values used by the equipment phase. This information is uploaded from the phase logic in the process controller to the Proficy Batch Server after the phase completes.

Procedure

Defines a process strategy for making a batch. Procedures consist of unit procedures defined for a recipe.

Process Cell

Consists of all the production and supporting equipment necessary to make a batch. It may include one or more production lines.

Project

The entire set of elements needed to deliver a batch solution. These elements include the recipes, pictures, configuration files, and equipment database.

Recipe Header

Administrative information about the recipe. This information includes the procedure identifier, version number, version date, and author.

Sequential Function Chart

A graphic representation of a recipe.

Step

A logical piece of an SFC (Sequential Function Chart). In the Recipe Editor, steps define the logic of a recipe.

Tab Delimiters

To add a tab to a string in Visual Basic, concatenate it as follows:

```
"string" + CHR(9) + "string"
```

To add a tab to a string in C++, use \t as follows:

```
"string \t string"
```

Transition

Defines when a recipe moves from one step to another in the sequential function chart.

Unit

A major piece of equipment in a process cell that performs a specific task. It consists of all the equipment and control modules that are needed to perform a task.

Unit Class

Defines common properties for a class of units. Used to create class-based recipes.

Unit Operation

A procedural element defining an independent processing activity that controls phases on a single piece of equipment.

Unit Procedure

Operations that control the function of a single piece of equipment.

Unit Tags

Tags that are associated with a unit, such as temperature and level tags. Unit tags are accessible to all phases that execute on that unit.

Unit Priority

Indicates the priority of the unit, as compared to other units in the same unit class. If multiple units are available for a batch, Proficy Batch selects the unit with the highest priority value. You can configure a

UNIT_PRIORITY tag to determine the priority value for a unit or you can assign a static priority value to the unit in the area model configuration.

Index