Foxboro®

by Schneider Electric

Foxboro Evo™
Process Automation System

# Scripting with Direct Access User's Guide

B0750BM

Rev W
June 18, 2014

# Contents

# Before You Begin

## About This Book

This document describes the Direct Access scripting capability included with Foxboro Evo™ Control Software (hereinafter referred to as the Control Software), presents the commands that are available via the Direct Access program, and gives examples for using the commands to access the Galaxy database and manipulate the Control Software objects independent of the ArchestrA® IDE client user interface.

This document is intended for use by engineers using the Direct Access utility for scripting in the Foxboro Evo Control Editors (hereinafter referred to as Control Editors).

## Revision Information

For Revision W of this document, the following changes were made:

**Chapter 12, "Reporting Operations"**

- Added safety information to "ProduceReport Command" on page 201.

## Reference Documents

Since the Control Software is based on the ArchestrA® architecture and incorporates several Wonderware® products, much of the documentation written by Wonderware is relevant. In addition, there are documents that describe the Control Editors-specific features. Below is a list of documents that can provide additional information that is beyond the scope of this document. These documents can be accessed from the Invensys Global Customer Support web site:

https://support.ips.invensys.com/

### Related Wonderware Documentation

The following documents can be accessed at the above website. Some of these documents are also available on the corresponding Wonderware product CD.

- *Wonderware® FactorySuite A2 Deployment Guide*
- *Wonderware Industrial Application Server User's Guide*
- *ArchestrA™ Integrated Development Environment (IDE) User's Guide*
- *Wonderware FactorySuite® Terminal Services for InTouch® Deployment Guide*

- *Wonderware FactorySuite InTouch User's Guide*
- *Wonderware FactorySuite InTouch Reference Guide*
- *Wonderware FactorySuite IndustrialSQL Server™ Concepts Guide*
- *Wonderware FactorySuite IndustrialSQL Server Administration Guide*
- *Wonderware FactorySuite IndustrialSQL Server Installation Guide*
- *Wonderware FactorySuite IndustrialSQL Server Database Reference*
- *Wonderware Tech Note 368 "Network Setup for AppEngine Redundancy"*
- *Wonderware Tech Note 401 "Fine-Tuning AppEngine Redundancy Settings*

# Foxboro Evo Control Software and Foxboro Evo Control Core Services Specific Documentation

During the Control Software component installation, the documents associated with the software components being installed are also installed. These documents can be viewed from the **Start** > **Programs** > **Invensys** > **InFusion Documentation** menu. Most of the Foxboro Evo Control Core Services (hereinafter referred to as Control Core Services) documentation can be found on the Foxboro Evo Electronic Documentation media (K0174MA).

Additionally, updated user documentation can be found on the Invensys Global Customer Support (GCS) website.

- *Access Manager User's Guide* (B0750AD)
- *Appearance Object Editor User's Guide* (B0750AE)
- *Block Configurator User's Guide* (B0750AH)
- *Bulk Data Editor User's Guide* (B0750AF)
- *Common Graphical Editor Features User's Guide* (B0750AG)
- *Hardware Configuration User's Guide* (B0750BB)
- *Control Database Deployment User's Guide* (B0750AJ)
- *Configuration Utilities User's Guide* (B0750AZ)
- *Foxboro Evo Control Software Installation Guide* (B0750RA)
- *Logic Block Editor and Troubleshooting Tool User's Guide* (B0750BL)
- *PLB Ladder Logic Editor User's Guide(B0750AK)*
- *Sequence Block HLBL Editor User's Guide* (B0750AL)
- *Sequence Block SFC Editor User's Guide* (B0750AM)
- *Strategy Editor User's Guide* (B0750AN)
- *System Manager* (B0750AP)
- *Framer and Alarm Management User's Guide* (B0750AR)
- *Control HMI Application User's Guide* (B0750AQ)
- *Window Construction User's Guide* (B0750AS)

- *100 Series Fieldbus Module Upgrade User's Guide* (B0700BQ)

In addition, printed copies of the following documents are shipped with the Control Software media kit:

- *Foxboro Evo Process Automation System Deployment Guide* (B0750BA)
- *Foxboro Evo Control Software Installation Guide* (B0750RA)
- *Control Software V5.0 Release Notes* (B0750SD)
- *Foxboro Control Software V4.0.1, 4.0.2, and 4.0.3 Release Notes* (B0750SC)

C H A P T E R 1

# Direct Access Overview

This chapter gives an introduction to the Direct Access scripting capability included with the Control Editors software, describes Direct Access features, and gives an overview of the GUI and Command Line versions of Direct Access.

## Contents



# Introduction to Direct Access Scripting

Direct Access is a general purpose scripting and query tool that allows you to access the Control Software objects and update configurable object attribute values within the Control Software Galaxy. You can use Direct Access to perform these operations with or without an active instance of the Control Editors running on the same platform.

Direct Access software, used with a graphical user interface or through a command line interface, processes commands stored in a text file in XML format. The commands allow you to manipulate data within the Galaxy database.

Direct Access software can also generate reports for documenting the results of your actions, and informs you of the success or failure of an operation. Output can be routed to the Direct Access user interface or to the command line, depending upon which version of Direct Access you are running. You can also output messages directly to the System Management Console (SMC) logfile.

# Features

The Direct Access tool provides the following features:

- SQL Table Operations allow you to create one or more SQL tables and store export and report data in the tables.
- Support Operations allow you to create, delete, or back up a galaxy, bulk compile control strategy templates or instances, specify a command file from which one or more additional commands are to be executed, insert an informational log message into the SMC log, check in objects, or perform string substitution in any command by specifying a variable and associated value.
- Query Filters allow you to look for the Control Software and ArchestrA objects, including block and ECB derived templates (**QueryFilter)**, define which block and ECB instances are to be processed (**BlockQueryFilter**), and define which attributes are to be processed (**AttributeQueryFilter**). With Direct Access querying capabilities, you can use the results of a filter in another operation such as generating a report, returning a value, updating an attribute, or performing an action on a Control Software object or attribute.
- Timer Functions allow you to start and stop a named timer, view the time elapsed for a specific timer, and temporarily suspend the execution of Direct Access for a specified period of time. When a timer is started, Direct Access creates a .csv file associated with the specified timer. This file contains the results of timer commands encountered within the command file.
- Assign Operations allow you to assign software and hardware system components to other system components, for example, you can assign a compound to a controller, a controller to an equipment unit or Ethernet switch, an FBM to an FCM or controller, and so forth.
- Create Operations allow you to create new derived Control Software objects, instances, or templates.
- Delete Operations allow you to delete derived Control Software objects.
- Deployment/Undeployment Operations allow you to deploy specified Control Software objects like blocks, compounds, controllers, ECBs, objects, or strategies, mark system components as deployed or undeployed (without actually deploying or undeploying them), selectively deploy attributes of Control Software objects that have been marked as modified in the Galaxy since last deployment, as well as upload specified attributes of deployed Control Software objects.
- Import/Export Operations allow you to import specified Control Software files of various formats from a specified URL and export specified Control Software objects from the Galaxy to Control Software files of various formats to a user-specified directory.
- Rename Operations allow you to rename derived Control Software objects.
- Reporting Operations allow you to produce a customizable report for query results. Direct Access software supports textual and graphical report

formats that can contain selected object attributes and values or strategy and network drawings, for example.

- Repeat Operations allow you to perform single or nested loops of commands. Repeat operations are useful for quickly building a large configuration with uniquely named objects with only a few commands.
- Reset Operations allow you to reset, or change, the most recently referenced object so that you do not need to re-enter it each time it's referenced.
- Attribute Update Operations allow you to update the attributes of various configurable Control Software objects like blocks, compounds, controllers, ECBs, software and so forth.
- Lock/Unlock Operations allow you to lock/unlock an attribute on a block template or instance on a strategy template, lock/unlock an attribute on one or more compound templates, and lock/unlock a user attribute within a strategy template.
- Command execution is limited to authorized users already established by the Control Software security model.
- The results of a session can be printed or stored in a user-specified file. In addition, an error log allows you to record all command execution failures to execute a command contained within a script.
- Writing Direct Access XML scripts can be made easier using the supplied XML schema definition for Direct Access in combination with most freely available XML text editors. This will give the script writer prompts for Direct Access commands, attributes and attribute values while editing Direct Access scripts. This schema file, "`DirectAccessSchema.xsd`", is installed along with the Direct Access executables in: "`Program Files\Archestra\Framework\bin\Invensys`".

# Installation

Direct Access is installed as part of the Control Editors. No additional steps are required to install Direct Access.

# Executing the GUI Version of Direct Access

To execute the GUI version of Direct Access, issue the following command from a DOS command prompt:

**DirectAccess**

from the same directory that DirectAccess.exe is located in, or have that directory somewhere in your **PATH** system environment. Alternatively, simply double-click **DirectAccess.exe**. The DirectAccess.exe executable is located in the D:\Program Files\Archestra\framework\bin\Invensys directory by default. However, if the Wonderware Industrial Application Server (IAS) software is installed in a folder other than the default, the executable will be located in <*IAS install folder*>\framework\bin\Invensys.

When executed, Direct Access will display the following dialog box:

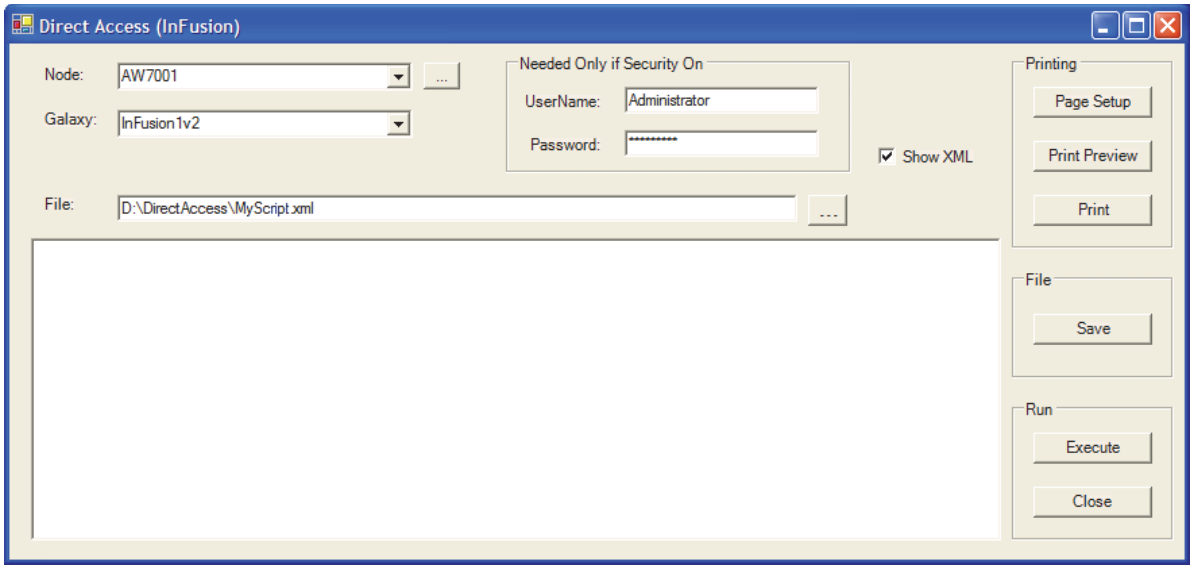



**Figure 1-1. Setting Up Direct Access**

Enter the following information in the Direct Access dialog box:

1. The **Node** field will be initialized to the last Galaxy node file repository you accessed. If this is the first time you have run Direct Access on your platform, it will be initialized to the name of your platform. To access a Galaxy running on another node, select the browse button next to the **Node** field, and select the desired domain and Galaxy server.

   Alternatively, you can enter the name of the remote Galaxy node, and press **Enter**, or press **Tab** to navigate to the next field. You will be informed if the node you entered is not reachable, or if there are no Galaxies found on that platform.

2. The **Galaxy** field will be initialized to the last Galaxy you accessed. If this is the first time you have run Direct Access on your platform, it will be initialized to the first Galaxy repository found on your platform. If no galaxies exist at the currently indicated node, then the text **<none>** appears, indicating that no galaxies have been selected.

3. Next, in the **File** field, specify the XML file containing the commands to execute. Use the **Browse** button provided to browse to the proper directory containing the command file.

4. If security is turned on within the Galaxy, provide a username and password in the **UserName** and **Password** fields. Otherwise, leave these fields blank.

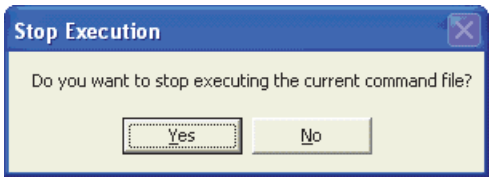
5. If desired, select **Show XML**. If this option is selected, the lines of the input file appear in the output window as they are executed. Enabling this option is helpful for debugging.

Once the node, Galaxy, and XML file have been specified, click **Execute** to actually begin processing the commands in the file to manipulate objects

within the Galaxy Repository. Syntax problems found within the XML file will be displayed in the text window directly underneath the file name.

If **Show XML** is selected, actions taken by Direct Access during processing of the XML file will be displayed in the text window directly underneath the file name. When Direct Access has completed processing the file, it displays the word **Done**. To execute the commands contained in a different XML file, simply enter the new file name in the **File** field and click **Execute** again. There is no need to exit the program.

**Note** While executing Direct Access, the **Close** button changes to a **Stop** button. At any time during execution of the program, you can press the **Stop** button, and you will be prompted with:




Answering **Yes** to the Stop Execution dialog box will stop the execution of the current command file. Keep in mind that after you click **Stop**, the current command will finish executing before the dialog box above is displayed.

To print the contents of the commands text window:

- Click **Page Setup** to initialize the page setup to the desired settings, and then either click **Print Preview** or **Print**, as desired.

To save the contents of the commands text window:

- Click **Save**, which will prompt you for the location to which you want to save the contents. You can save the contents as either a plain text file (*.txt) or a rich text formatted (*.rtf) file.

# Executing the Command Line Version of Direct Access

To execute the command-line version of Direct Access, issue the following command from a DOS command prompt:

`DirectAccess_Cmd` <**GalaxyNode**> <**GalaxyName**> <**CommandFile**> <**UserName**> <**Password**>

or:

`DirectAccess_Cmd NoGalaxy` <**CommandFile**>

from the same directory that DirectAccess_Cmd.exe is located in, or have that directory somewhere in your **PATH** system environment variable. DirectAccess_Cmd.exe is located in the D:\Program Files\Archestra\framework\bin\Invensys directory by default. However, if the Wonderware Industrial Application Server (IAS) software is installed in a folder other than the default, the executable will be located in *<IAS install folder>*\framework\bin\Invensys.

In the command above:

- **GalaxyNode** refers to the name of the Galaxy node of where you wish to open the Galaxy. This may be the name of any valid platform to which **DirectAccess_Cmd** has network access, or the name of the local platform. The **GalaxyNode** may also be specified as **NoGalaxy** indicating that the command file will be creating (or deleting) a named Galaxy during program execution.
- **GalaxyName** refers to the name of the Galaxy on the platform you wish to open.
- **CommandFile** refers to the name of the XML command file that will be processed during this session of Direct Access. It must be a fully qualified name, that is, if the file is not in your immediate local directory, you must specify its full location beginning with the drive letter.
- **UserName** refers to the username of the user running DirectAccess_Cmd. This is needed only if security is turned on within the Galaxy.
- **Password** refers to the password of the user logging in under the username. A password is needed only if security is turned on.

As DirectAccess_Cmd executes, the program's progress will be displayed on the screen.

Following is an example of using the command line version of Direct Access:

**DirectAccess_Cmd USFOX0246 MyGalaxy**
**C:\DirectAccess\TestCommands.xml**

# Multi-User Support

Other users can be accessing the Galaxy at the same time this program runs, including having the IDE run on the same platform. However, if a command within the XML relies on having update access to an object that is checked out by another user, an error will result.

**Note** If this utility runs on a platform with IDE running on it, an object that you checked out in the IDE will be able to be updated from within Direct Access, as long as the object's editor is not open within the IDE.

**Note** Since Direct Access commands are executed wholly with GRAccess and everything runs client-side, once a Direct Access script begins to run, it will dominate the entire Galaxy network until finished. Other clients will be operational, but will experience severe performance slowdown.

# Conventions of Direct Access XML Files

A Direct Access XML file must begin with a valid XML declaration, as in:

```
<?xml version="1.0" encoding="utf-8"?>
```

**Note** If the XML files are edited manually, they must be saved with UTF-8 encoding. This option exists in most text editors.

Immediately following the XML declaration, the XML commands must be bracketed with the following XML root:

```
<DirectAccess>
…
</DirectAccess>
```

Each command contained within a Direct Access command file begins with an *action*. The action command is followed by one or more keywords and keyword values providing additional data needed to perform the indicated action.

The action, keywords and associated keyword values are all enclosed within angle brackets, as follows:

```
<Action Keyword1="Value1" Keyword2="Value2"..../>
```

Successive keywords and their values must be separated by at least one space. All keyword values must be enclosed within quotes. The ending angle bracket (‘>’) must be immediately preceded by a forward slash (‘/’), indicating the action specification is complete, or an error reading the XML file will occur.

**Note** Direct Access is case-sensitive. When creating a Direct Access XML file, every *action* and *keyword* must have the exact case listed for their associated commands in this document. If an action or keyword does not have the proper case, Direct Access will not recognize it.
This pertains to keyword values as well. For example, if a command is meant to create an object for a specific controller but the controller’s name in the command does not match the correct case for the controller’s name, then the object may be created under a new controller which has the incorrect name.

# Object Name Resolution

Generally, Direct Access finds objects in the Galaxy by treating each object name as a tagname. However, there are a few exceptions.

1. When the name specified contains a period (‘.’), it is assumed to be the hierarchical name of the object. An example of this is a strategy contained within a compound, for example, **COMPND_001.MyStrategy**, or an FBM assigned to a controller, for example, **CP2701.F00002**. This hierarchical name applies only for containment relationships, **not** hosting relationships. The following relationships are containment relationships within Control Editors software, so these objects may be referred to by either their tagname or their hierarchical name.

- FCMs are contained by a controller
- FBMs are contained by a controller (or FCM)
- Devices are contained by an FBM
- Equipment units are contained by another equipment unit
- Plant units are contained by another plant unit
- Strategies are contained by compounds.

When specifying a hierarchical name, the name must refer to the *contained* name of the object, such as **MYCP00.FBM255.MY_DEV**, versus the tagname (for example, **DEV001**).

**Note** The contained name of a device must take into account the **CP** > **FBM** > **Device** hierarchy, not the **Plant Unit** > **Device** hierarchy.

The following are hosting relationships, in which hierarchical names do not apply. In these cases, the object **must** be referred to by its tagname.

- Controllers, workstations, and switches are hosted by an equipment unit
- Compounds are hosted by controllers.

Blocks and ECBs are not true ArchestrA objects, so they must be referenced by their name in the context of their "owning" entity: that is, a compound "owns" an ECB, and a strategy "owns" a block.

2. Object names beginning with a dollar sign ('**$**') are assumed to be ArchestrA template definitions, for example, **$MY_AIN**.
3. A control block is not an ArchestrA object, and therefore needs another level of reference in order to find the correct block. This is done by providing the block's strategy name (either tagname or hierarchical name).

   Example:

   ```
   <CreateBlock Template="$AIN" Block="MY_AIN"
   Strategy="COMPND_001.MyStrategy1"/>
   ```

   Will create the block **MY_AIN** in the strategy **MyStrategy1**, contained within the compound **COMPND_001**.
4. Many actions support the establishment of a relationship between two entities, one of which is typically thought of as an "owner", or "container", of the other. Examples of such relationships exist between a strategy and a compound, or a control block and a strategy. When specifying actions of this type, you have the option of specifying both entities explicitly, or allowing the "owning" entity to default to the one most recently referenced within the XML data file.

   Example:

   ```
   <CreateBlock Template="$AIN" Block="MY_AIN"/>
   ```

   In this example, a block cannot be created without a strategy, so Direct Access will create the block specified in the most recent strategy that was referenced within that XML data file. If a strategy had not yet been referenced, an error will result.

CHAPTER 2

# SQL Table Operations

This chapter describes SQL table operations using Microsoft Access® database software and Microsoft Excel® spreadsheet software.

All Direct Access exports and reports support the creation of one or more SQL tables, and the subsequent storage of export or report data into those tables. Any SQL tables created from Direct Access are stored in the SQL Server® database called **InFusion_Data**.

You can specify your own SQL table names for report contents, but for export purposes, the table names are determined by Direct Access. In either case, the current Windows® operating system user ID will automatically be prepended to the tablename, to differentiate it from SQL tables created by other users of Direct Access.

However you decide to access any table within the InFusion_Data database, you must access the data with the user/login:

- **User name:** InFusion_Reader
- **Password:** InFusion

**Note** Database, user name, and password are automatically created during the first export or report of type SQLTable.

**Note** SQL table exports are only supported when run from the Galaxy Server.

## Contents



# Accessing InFusion_Data via Microsoft Access Database Software

To access your tables within the InFusion_Data database using Microsoft Access database software, perform the following steps:

1. Open the Access software project that you want to connect to a Microsoft SQL Server.

2. Select **File > Connection** to invoke the Data Link Properties dialog box, shown below.

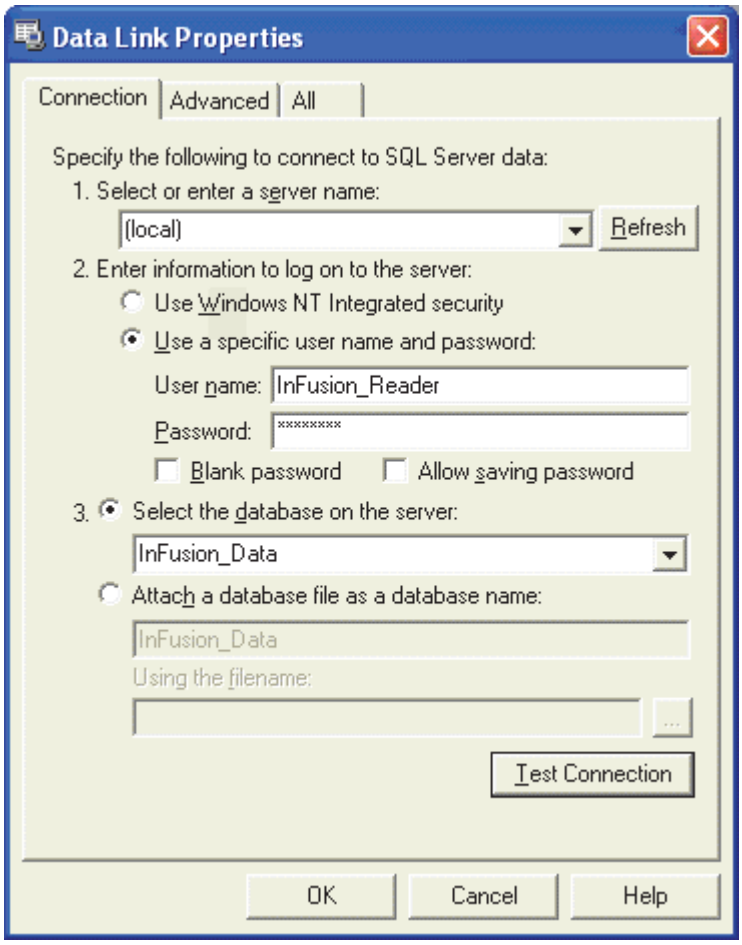



**Figure 2-1. Data Link Properties Dialog Box**

3. Fill in the Data Link Properties dialog box with the following information:
   a. Server name: **(local)**, or enter the actual server name if not on your local machine (for example, **USFOX0249**).
   b. Select **Use a specific user name and password**:
      - For **User name**, enter **InFusion_Reader**
      - For **Password**, enter **InFusion**
   c. Select the **InFusion_Data** database on the specified server from the pull-down provided, or enter the name directly.
   d. You can click **Test Connection** to make sure the connection is valid.
   e. Click **OK**.
4. Microsoft Access software will now populate the open project with all the SQL tables found within the InFusion_Data database, as depicted below.

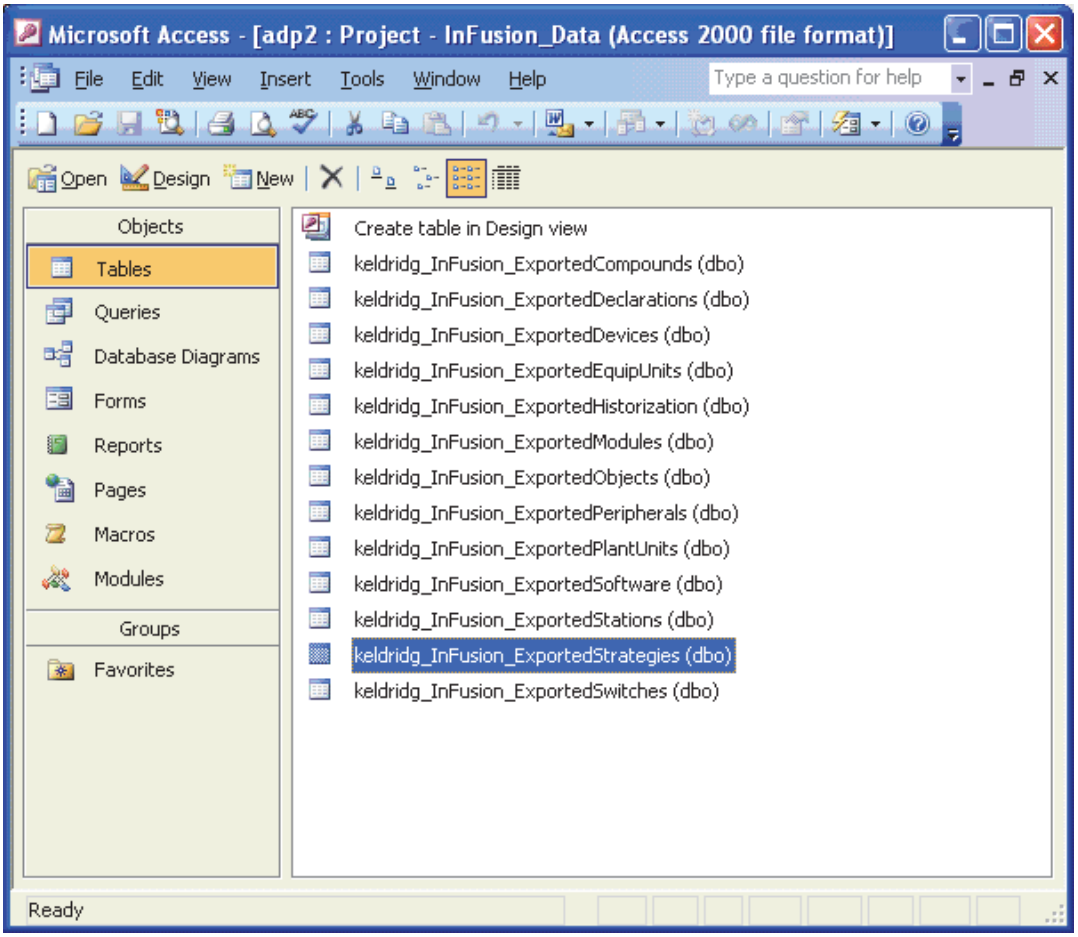



**Figure 2-2. Viewing InFusion_Data Tables with Microsoft Access Software**

5. Please note that this will include not only your tables, but SQL tables created by other users of Direct Access as well. You should work only with those tables that are prepended with your Windows user ID.

**Note** You can also access SQL data tables within InFusion_Data with Microsoft Access software by selecting **File > Get External Data > Import**, but the procedure just described is less complicated. Refer to Microsoft Access software documentation to learn about importing external data.

6. You can double-click one of the tables to open it for viewing (this document will not explain the inner workings of Microsoft Access software, but rather offers this as an example to quickly see results):

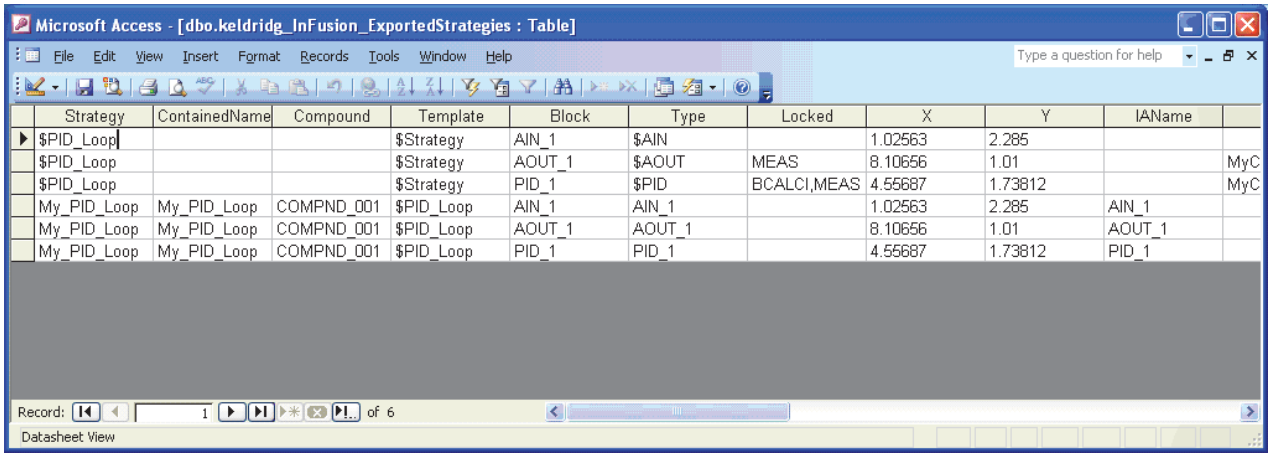



**Figure 2-3. Viewing Details of InFusion_Data Tables with Microsoft Access Software**

# Accessing InFusion_Data via Microsoft Excel Spreadsheet Software

To access your SQL tables within the InFusion_Data database using Microsoft Excel® software:

1. From an open worksheet in Microsoft Excel software, select **Data** > **Import External Data** > **Import Data**.

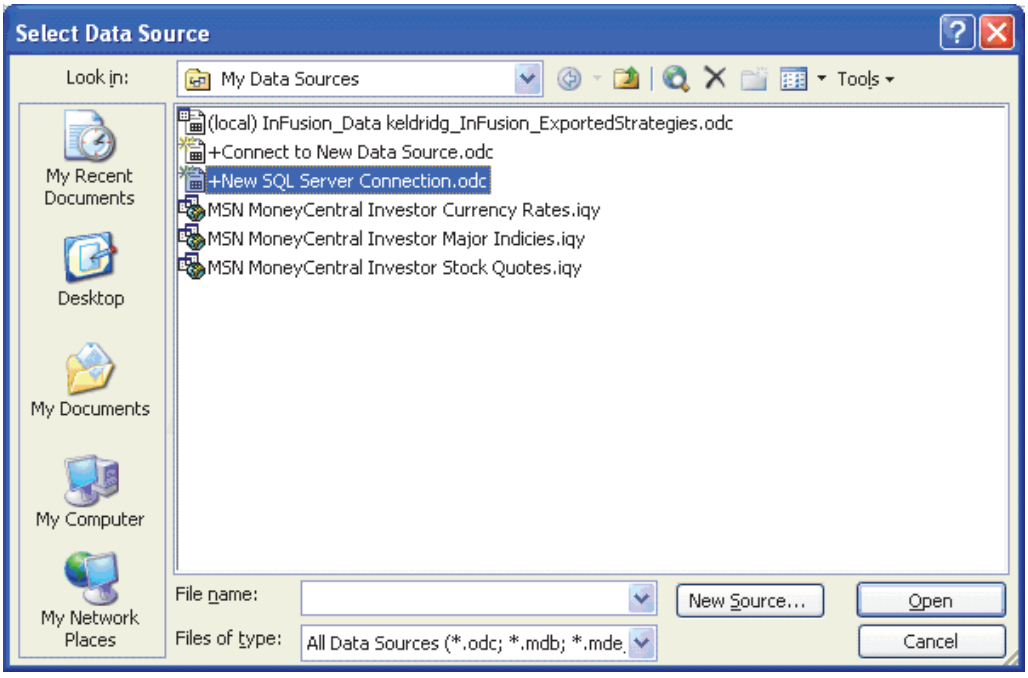



**Figure 2-4. Selecting a Data Source to Import in Microsoft Excel Software**

2. On the Select Data Source dialog box, select **+New SQL Server Connection.odc**, and click **Open**.

3. On the Data Connection Wizard dialog box, supply the server name for the database server (use **(local)** if the database is on your local platform), and use the user name **InFusion_Reader** with the appropriate password. Click **Next**.

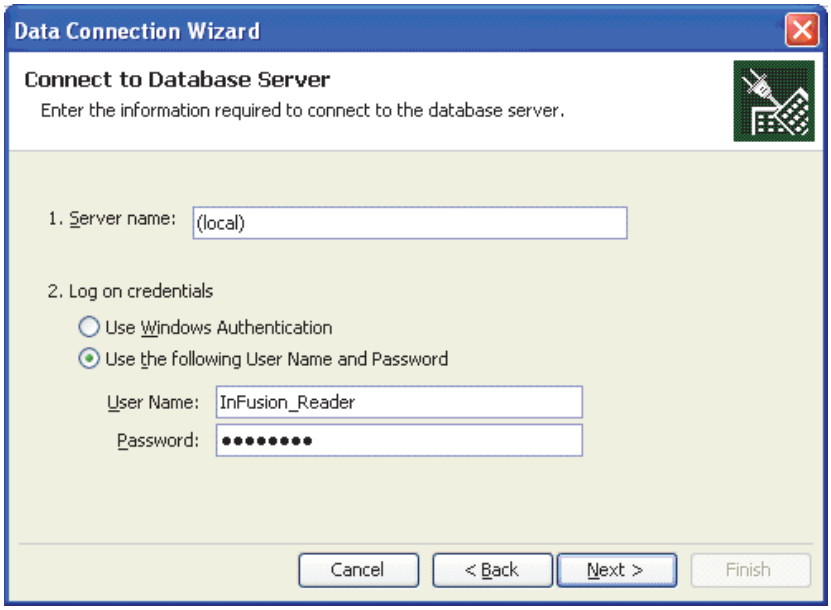



**Figure 2-5. Microsoft Excel Software Data Connection Wizard**

4. Specify the **InFusion_Data** database, by either selecting it from the pull-down menu or entering it manually, and select the appropriate SQL table to import. You should work only with those tables that are prepended with your Windows user ID. Click **Finish**.

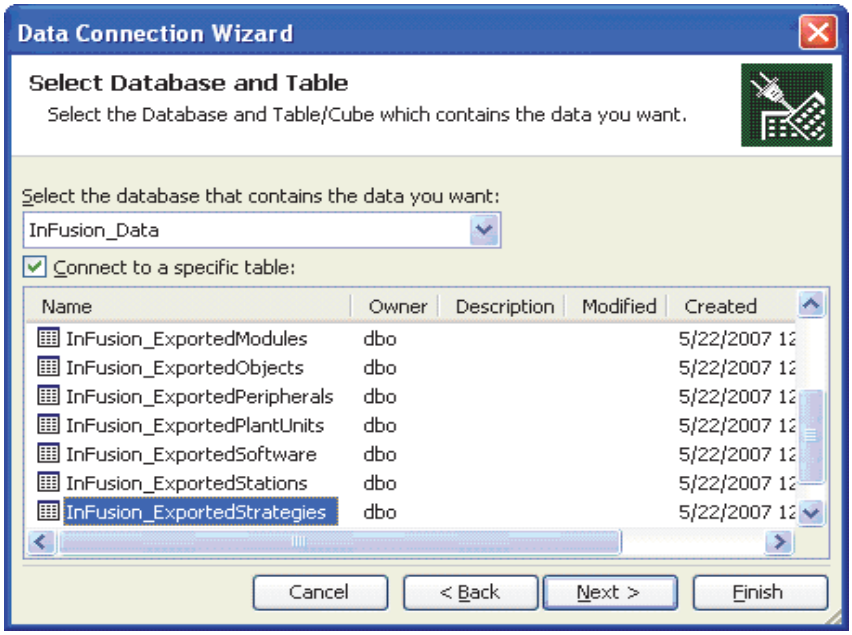



**Figure 2-6. Selecting a Database and Table with Microsoft Excel Software**

5. Select whether or not you want the imported data to appear on the existing worksheet (and specify a location), or a new worksheet. Click **OK**.

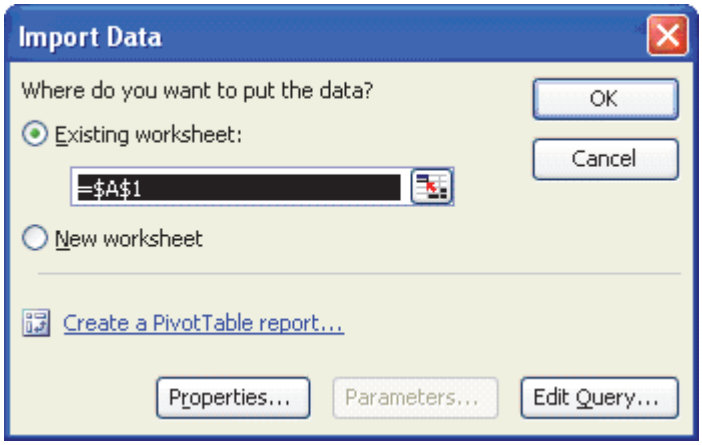



**Figure 2-7. Selecting a Location for Imported Data with Microsoft Excel Software**

6. Specify the final **Password** check to access the data, and then click **OK**.

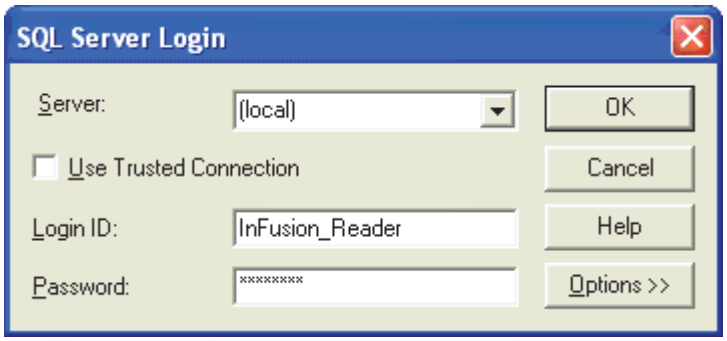
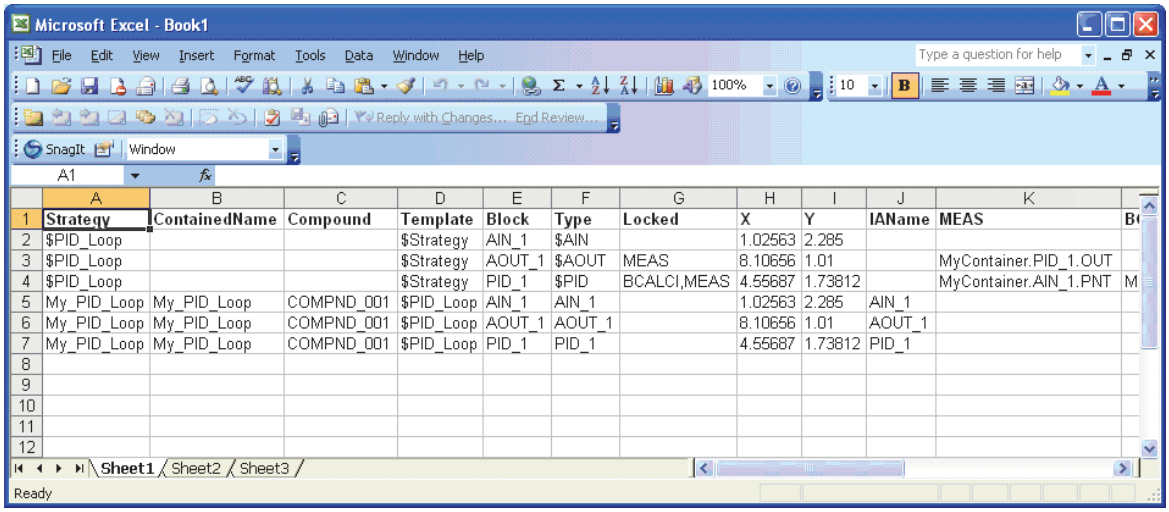



**Figure 2-8. Specifying a Password for Data Access with Microsoft Excel Software**

7. The data from the specified table within the InFusion_Data database is now imported into the spreadsheet.




| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Strategy | ContainedName | Compound | Template | Block | Type | Locked | X | Y | IAName | MEAS |
| 2 | $PID_Loop | | | $Strategy | AIN_1 | $AIN | | 1.02563 | 2.285 | | |
| 3 | $PID_Loop | | | $Strategy | AOUT_1 | $AOUT | MEAS | 8.10656 | 1.01 | | MyContainer.PID_1.OUT |
| 4 | $PID_Loop | | | $Strategy | PID_1 | $PID | BCALCI,MEAS | 4.55687 | 1.73812 | | MyContainer.AIN_1.PNT |
| 5 | My_PID_Loop | My_PID_Loop | COMPND_001 | $PID_Loop | AIN_1 | AIN_1 | | 1.02563 | 2.285 | AIN_1 | |
| 6 | My_PID_Loop | My_PID_Loop | COMPND_001 | $PID_Loop | AOUT_1 | AOUT_1 | | 8.10656 | 1.01 | AOUT_1 | |
| 7 | My_PID_Loop | My_PID_Loop | COMPND_001 | $PID_Loop | PID_1 | PID_1 | | 4.55687 | 1.73812 | PID_1 | |
| 8 | | | | | | | | | | | |
| 9 | | | | | | | | | | | |
| 10 | | | | | | | | | | | |
| 11 | | | | | | | | | | | |
| 12 | | | | | | | | | | | |

**Figure 2-9. Table Imported into Excel Software**

CHAPTER 3

# Support Operations

This chapter describes operations that allow you to create, delete, or back up a Galaxy, bulk compile control strategy templates or instances, specify a command file from which one or more additional commands are to be executed, insert an informational log message into the SMC, check in objects, or perform string substitution in any command by specifying a variable and associated value.

## Contents



# BackupGalaxy Command

The **BackupGalaxy** command allows you to back up the current Galaxy.

**Syntax:**

```
<BackupGalaxy      Name="GalaxyBackupName"/>
```

| Attribute | Description |
|---|---|
| GalaxyBackupName | The name of the galaxy backup to create.<br>The currently open galaxy will be backed up into the standard ArchestrA BackupGalaxies location. If no galaxy is open at the time the command is issued, an error will result. |

**Example:**

Back up the currently open galaxy to **My_Galaxy.cab**:

```
<BackupGalaxy Name="My_Galaxy.cab"/>
```

# BulkCompile Command

The **BulkCompile** command allows you to bulk compile one or more control strategy templates or instances.

**Syntax:**

```
<BulkCompile  Strategy="StrategyName"/>
              Or
<BulkCompile  Filter="FilterName"/>
```

**Note** For the **BulkCompile** command to work, you must run Direct Access on a computer that has the Control Core Services compilers installed for the Control Software. Any messages that would normally appear in the output window of the Strategy Editor within the Control Editors will appear in the Direct Access output stream.

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy to be bulk compiled.<br>You can specify either a template or instance. This command will also bulk compile any child strategies that may be contained within the named strategy.<br>**StrategyName** should not be specified if **FilterName** is given. |
| FilterName | The name of the filter that will be used to determine what strategies get bulk compiled.<br>**FilterName** should not be specified if **StrategyName** is given. |

**Examples:**

Bulk compile strategy instance **My_Strategy1** that is contained within compound **CMPND_001**:

```
<BulkCompile Strategy="CMPND_001.My_Strategy1"/>
```

Bulk compile strategy template **$MY_STRAT** and any strategy templates that may be contained within **$MY_STRAT**:

```
<BulkCompile Strategy="$MY_STRAT"/>
```

Bulk compile child strategy instance **INNER_LOOP** contained within the strategy **OUTER_LOOP** in compound **CMP_001**:

```
<BulkCompile Strategy="CMP_001.OUTER_LOOP.INNER_LOOP"/>
```

Bulk compile all strategies contained in compound **COMPND_002** that begin with **STR** and bulk compile any child strategies which may be contained within those strategies as well:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_002"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="STR%"/>
<BulkCompile Filter="Filter1"/>
```

# CheckInObject Command

The **CheckInObject** command allows you to check in an object that is checked out.

### Syntax:

```
<CheckInObject      Object="ObjectName"/>
```

**Note** This is one way to check in an object that was left in a checked out state and needs to be checked in. This is especially useful for checking in "hidden" objects that were left checked out. Use the **Find** dialog box within the IDE to find all checked out objects, then you can use this command to check them in.

| Attribute | Description |
|---|---|
| ObjectName | The tagname of the object to check in.<br>If security is turned off, and the object is not being edited by someone else, or by you in another session, it will be checked in.<br>If security is turned on, the object will only be checked in if it was checked out by you, and is not being edited in another session. |

### Example:

Check in the compound **COMPND5**:

```
<CheckInObject Object="COMPND5"/>
```

# CommandFile Command

The **CommandFile** command specifies a command file from which one or more additional commands are to be executed.

**Syntax:**

```
<CommandFile  Name="FileName"/>
```

| Attribute | Description |
|---|---|
| FileName | The fully qualified filename. |

This command specifies a command file from which one or more additional commands are to be executed, effectively allowing you to specify nested command files. There is no limit to the number of nested files you can specify.

Care should be taken to be sure not to specify a command file that is already in the "chain" of executing command, or an endless loop will result.

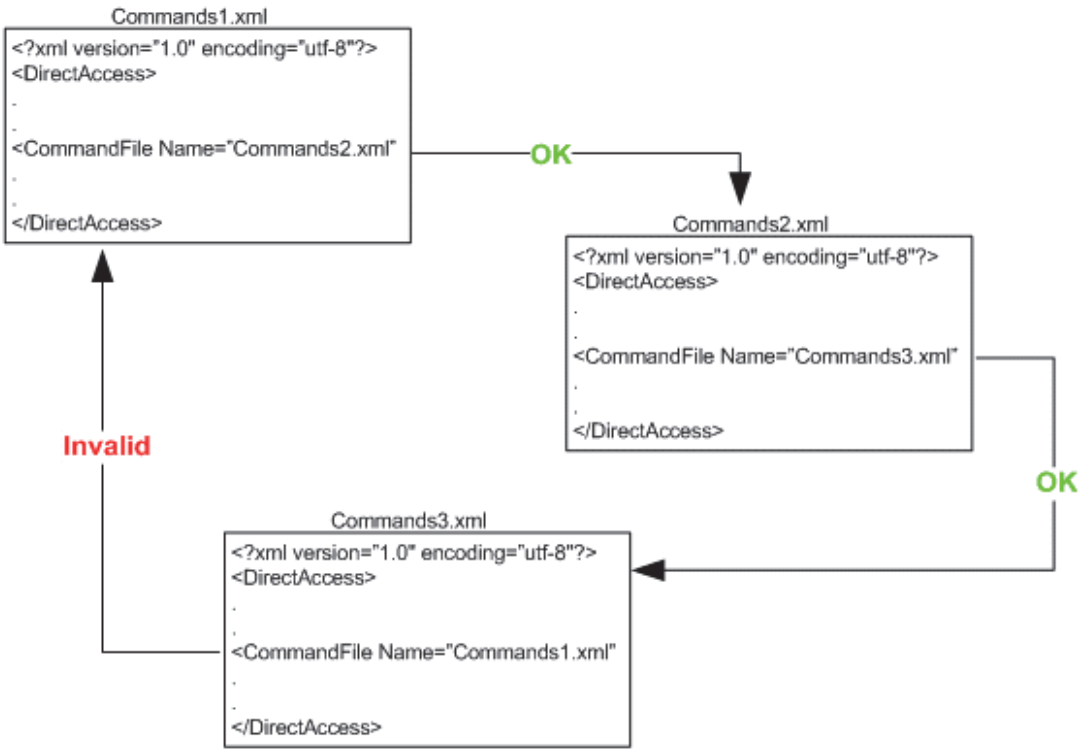



**Figure 3-1. Using the CommandFile Command**

There is also no restriction to the commands that can be embedded within any of the nested command files. For example, any command file, whether it be a top-level command file or a nested one, can use the **PerformOperation** command, allowing looping within the command file.

### Examples:

Execute the commands found in the local file TestCommands.xml:

```
<CommandFile Name="TestCommands.xml"/>
```

Execute the commands found in the file C:\DirectAccess\TestCommands.xml:

```
<CommandFile Name="C:\DirectAccess\TestCommands.xml"/>
```

# CreateGalaxy Command

The **CreateGalaxy** command allows you to create a Galaxy.

### Syntax:

```
<CreateGalaxy      Name="GalaxyName"
                   Flags="FlagOptions"
                   CABFileName="CABFile"/>
```

| Attribute | Description |
|---|---|
| GalaxyName | The name of the Galaxy to create.<br>The Galaxy name must be unique for the platform – that is, if a name is provided for a Galaxy that already exists, an error message will result.<br>No mechanism is provided for determining which version of ArchestrA to use when creating this Galaxy.<br>By default, the Galaxy will be created with security not enabled. |
| FlagOptions | Determines whether or not a unique string should be appended to the string provided by **GalaxyName** in order to determine the full name of the Galaxy.<br>Valid values are:<br>• **Date**. Indicates that the current time and date should be appended to the name specified by **GalaxyName**<br>• **GUID**. Indicates that a unique GUID string should be appended to the name specified by **GalaxyName**. |
| CABFile | The name of the Galaxy backup (CAB) file that is to be used as a basis for the created Galaxy. Standard Control Software Galaxies are built from "AquariusGalaxyBackup.cab". |

### Examples:

Create a Galaxy called **Test**:

```
<CreateGalaxy Name="Test"
CABFileName="AquariusGalaxyBackup.cab"/>
```

Create a Galaxy called **Test** and append the current time and date:

```
<CreateGalaxy Name="Test" Flags="Date"
CABFileName="AquariusGalaxyBackup.cab"/>
```

(for example, using the syntax above would result in a name similar to **Test_02082006_122252PM**)

Create a Galaxy called **Test** and append a unique GUID:

```
<CreateGalaxy Name="Test" Flags="GUID"
CABFileName="AquariusGalaxyBackup.cab"/>
```

(for example, using the syntax above would result in a Galaxy name similar to **Test_85480958_7306_4B75_A335_1E3BAF584C07**)

# DeleteGalaxy Command

The **DeleteGalaxy** command allows you to delete a Galaxy.

### Syntax:

```
<DeleteGalaxy Name="GalaxyName"/>
```

| Attribute | Description |
|---|---|
| GalaxyName | The name of the Galaxy to delete.<br>The Galaxy must exist, or an error message will result.<br>If the name specified matches the name of a Galaxy that is already open within the Direct Access session, the program will log the user out of the Galaxy and close it prior to attempting the delete operation.<br>The Galaxy being deleted should not contain any deployed objects. If any objects are deployed, an error message will result.<br>**Note** This command cannot be undone. Once the command to delete a Galaxy executes, that Galaxy will no longer be found on the platform. |

### Examples:

Delete a Galaxy called **My_Galaxy**:

```
<DeleteGalaxy Name="My_Galaxy”/>
```

# DeleteGalaxyBackup Command

The **DeleteGalaxyBackup** command allows you to delete one or more Galaxy backup .cab files.

### Syntax:

```
<DeleteGalaxyBackup Name="GalaxyBackupName"/>
```

| Attribute | Description |
|---|---|
| GalaxyBackupName | The name of the galaxy backup to delete.<br>The galaxy backup .cab file must exist, or an error will result.<br>If **GalaxyBackupName** = asterisk ("*"), then all galaxy backup files on the platform will be removed.<br><br>**Note** This command cannot be undone. Once the command to delete a Galaxy backup executes, that Galaxy backup will no longer be found on the platform. |

**Examples:**

Delete a galaxy backup file called **My_Galaxy.cab**:

```
<DeleteGalaxyBackup Name="My_Galaxy.cab"/>
```

Delete all galaxy backup files on the platform:

```
<DeleteGalaxyBackup Name="*"/>
```

# LogMessage Command

The **LogMessage** command inserts an informational log message into the ArchestrA logger, viewable from the ArchestrA SMC.

**Syntax:**

```
<LogMessage Message="LogMessage"/>
```

| Attribute | Description |
|---|---|
| LogMessage | The string to insert into the ArchestrA logger.<br>Viewable from SMC, this log message will be inserted with the following characteristics:<br>• Log Flag: **Info**<br>• Component: **DirectAccess**<br><br>**Note** To see messages put into the logger from Direct Access, you must select **Action** > **Log Flags** from within SMC, and ensure that **Info** is selected under **Global Log Flags**. |

**Examples:**

Insert a log message stating that a specific operation is about to take place:

```
<LogMessage Message="Assign compound MT07 to controller FCP207"/>
```

# MigrateFBM Command

The **MigrateFBM** command allows you to migrate the existing 100 Series FBMs to the appropriate 200 Series FBMs in the Galaxy.

The list of the 200 Series FBMs which replace the 100 Series FBMs is provided in *100 Series Fieldbus Module Migration User's Guide* (B0700BQ).

### Syntax:

```
<MigrateFBM        FBM100="FBMXYZ"
                   FBM200="FBMABC"/>
```

| Attribute | Description |
|---|---|
| FBMXYZ | The ArchestrA tagname of the FBM that is to be migrated.<br>To determine an FBM's ArchestrA tagname, right-click on the FBM in Control Editors and select **Properties**. The tagname is listed adjacent to the yellow tag icon, and in the Attributes tab of the Properties table for the FBM. |
| FBMABC | The letterbug name to be assigned to the 200 Series FBM which will replace the 100 Series FBM specified in the "FBM100" field in this command.<br>This attribute is subject to the naming rules applicable to that 200 Series FBM. |

### Example:

The following example shows an FBM-level migration. An 100 Series FBM which has the ArchestrA tagname **F00002** will be migrated to the 200 Series FBM which will have the letterbug **F00201**.

```
<MigrateFBM FBM100="F00002" FBM200="F00201" />
```

**Notes:**

You must follow these notes during the 100 Series migration process:

- During the 100 Series migration, all users must not operate on any of the affected control processors, FCMs, FBMs, compounds or blocks (including ECBs) in Control Editors; for example, opening any of the ECBs in editors, or viewing their properties in assignment tabs, etc. Such operations may result in unexpected behaviors in Control Editors.
- Control Editors will identify any situations in which the name of a 200 Series FBM created as part of migration has a base naming conflict with any of the other FBMs in the Galaxy. Any instances of this situation will stop the migration process for the affected FBMs.

While migrating a 100 Series FBM to a 200 Series FBM, the 200 Series FBM instance name should not end with "00". This may lead to an ECB error after deployment.

# MigrateWorkstationAW70P Command

The **MigrateWorkstationAW70P** command allows you to migrate AW70P instances to the appropriate WSTA70/WSVR70 workstation types. The command takes care of mapping the hardware attributes, software packages, and peripherals of AW70P to corresponding WSTA70/WSVR70 supported types. Warnings are displayed if there are any peripherals or software packages configured for AW70P that are not supported by the target workstation. The warnings only display information and do not abort the migration.

### Syntax:

```
<MigrateWorkstation AW70P  Workstation="WorkstationName"
                           TargetWorkstationType="TargetWorkstationType"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The ArchestrA tagname of the AW70P instance to be migrated. |
| TargetWorkstation Type | The target workstation type can be either WSTA70 or WSVR70. |

**Note** The migration deletes the original AW70P and creates a new WSTA70/WSVR70. It is recommended that a backup of the entire Galaxy be kept before starting the migration process.

### Example:

The following example describes the migration of AW70P instance to WSTA70. An AW70P instance which has ArchestrA letterbug **S00001** will be migrated to WSTA70 workstation type with the letterbug **S00001**.

```
<MigrateWorkstationAW70P Workstation="S00001"
TargetWorkstationType="WSTA70" />
```

### Software package mapping of AW70P to WSTA70/WSVR70

| AW70P | WSTA70/WSVR70 |
|---|---|
| AADM7 | AADM7 |
| ADM7 | IAMESH, IASVCS |
| ASMDW7 | ASMDW7 |
| AXCEED7 | IASVCS |
| OS7AW1 | ADDISP, IAMESH, IASVCS, IACTRL |
| ACSA7 | IAMESH, IACTRL, IASVCS, ACSA7 |
| AHIST7 | Need not be configured. |
| AMSGM7 | AMSGM7 |
| ANSOF7 | ANSOF7 |
| ASMON7 | ASMON7 |

| AW70P | WSTA70/WSVR70 |
|---|---|
| ADDE7 | Need not be configured. |
| AMCMAP | Need not be configured. |
| AMCSRV | Need not be configured. |
| AICC7 | Need not be configured as this is not supported for WSTA70/WSVR70. |

**Peripheral mapping of AW70P to WSTA70/WSVR70**

| Peripheral selection in AW70P | Peripheral selection that will be made in WSTA70/WSVR70 |
|---|---|
| GCIO | Serial-Port GCIO |
| Terminal / Modem / P132S / P136S / P80BWS | These peripherals will not be available in WSTA70/WSVR70. Hence, selecting these in AW70P will not have any effect on new workstations. |
| | USB Printers and USB Annunciators will not be selected after migration because AW70P does not support these. |
| Any peripheral other than the ones mentioned above | Same peripheral (as that of AW70P) being selected. |

# SetDeployOption Command

The **SetDeployOption** command allows you to set the sequence block deployment options in the Deploy Options dialog box using Direct Access with the following command:

```
<SetDeployOption Option="Value" CodeGrowthFactor="Factor"/>
```

**Option** is a required attribute whose value “Value” should be one of the following:

- CP
- USER_CONFIGURED
- IEE_CALCULATED

**CodeGrowthFactor** is a required attribute when **Option=IEE_CALCULATED**. The value of the attribute “Factor” must be within the range of 0 to 30000.
This command can be executed in Direct Access only when the user specified in the Direct Access application has administrator privileges in the Control Software.

# SetVar Command

The **SetVar** command allows you to specify a variable and associated value to be used for string substitution in any command. You can use the variable in subsequent commands by bracketing its name with a set of percent signs (%). Whenever such a string is encountered on any command, the value of the variable will be used to replace the variable name, resulting in string substitution.

### Syntax:

```
<SetVar          Variable="VariableName" Value="VariableValue"/>
```

**Note** Use caution when combining the use of variable substitution and command loops. It is strongly suggested to avoid the use of percent signs ('%') when declaring string substitution patterns in a **PerformOperation** command.

**Note** The variables declared by the **SetVar** command are in memory only for the duration of one "run" of Direct Access. Each time a new command file is executed, the table(s) storing the variable names and values are reinitialized to an empty state, even if you never leave the GUI.

| Attribute | Description |
|---|---|
| VariableName | The name of the variable to set. Variable names are case sensitive, so variable **MyVar** is different from variable **MyVaR**.<br>There is no limit to the number of variables that can be stored, and no restrictions on their names. |
| VariableValue | The value of the variable. If a variable was set by a previous **SetVar** command, the old value will be replaced by the new value.<br>There are no restrictions on what the variable value can consist of. However, the use of special characters in the value may result in unforeseen results if specified within a **PerformOperation** command.<br>The value specified by **VariableValue** for any **VariableName** will remain in effect for the duration of the "run" of Direct Access, unless another **SetVar** command on that same variable name is encountered. |

**Example:**

The following example shows how to set a variable value called **MyVar** to the characters **ABCD**. Whenever the presence of the variable is encountered in subsequent commands (indicated by the characters "**%MyVar%**"), substitute the characters **ABCD**.

```
<SetVar Variable="MyVar" Value="ABCD"/>

<CreateCompound Compound="%MyVar%CMPD"
Template="$COMPND"/>
```

The syntax of the example above will result in the creation of the compound **ABCDCMPD**.

CHAPTER 4

# Query Filters

This chapter describes query filters, which allow you to specify criteria when searching for Control Editors software objects and/or attributes. You can specify filters with standard expressions supported by the ArchestrA environment. Query filters can be used to extend a command that would normally operate on a single object.

With Direct Access querying capabilities, you can use the results of a filter in another operation such as generating a report, returning a value, updating an attribute, or performing an action on a Control Software object or attribute.

## Contents



# Types of Query Filters

There are three types of query filters for Direct Access:

- **QueryFilter** – original filter restricted to looking for the Control Software and ArchestrA objects, including block and ECB derived templates
- **BlockQueryFilter** – query filter used to define which block and ECB instances to return
- **AttributeQueryFilter** – query filter used to define which attributes to return.

Using query filters, you can extend a command that normally operates on a single object, and perform that same activity on any number of objects matching the query.

In order for a **BlockQueryFilter** and/or an **AttributeQueryFilter** to be used in conjunction with a **QueryFilter**, the same filter name must be used (for example, **Filter1**), as shown in later examples.

Query filters are stored and retrieved by name. Each named filter can actually be composed of many “sub-queries”, depending on how they are specified using the **QueryFilter**, **BlockQueryFilter**, or **AttributeQueryFilter** commands. Each time a query filter command is issued, Direct Access checks to see if a filter by that name already exists. If it does, the filter condition and value are added to the existing filter. If it does not, then a new filter by that name is created.

Any “sub-query” condition specified for a query filter may be negated by placing an exclamation point in front of it (for example, Condition="!BasedOn" equates to “not based on”).

You can reset an existing filter via the **Reset** command. This allows a new filter to be specified using a filter name that was previously used in the command file. The **Reset** command effectively removes any existing filter conditions and values from the named filter.

With Direct Access query filters, you can find all Control Software objects in the Galaxy that satisfy the specified search criteria. For example, you can find:

1. The full containment hierarchy for a specified Control Software object tagname
2. The set of FBM letterbugs and channel numbers that are assigned to specified strategy and block names
3. All strategies with blocks that are assigned to a specific FBM
4. All the spare channels of all FBMs that are of a specific type (for example, FBM203) and/or assigned to a specific controller (for example, CP0100)
5. All members of a containment hierarchy starting from a user specified container such as an equipment unit, controller, compound, or strategy
6. All Control Software objects in the Galaxy that reference a specified Control Software object or object attribute
7. All block attributes in the Galaxy that reference a specified set of block attributes indirectly via strategy declarations
8. All Control Software objects in the Galaxy that are referenced by a specified Control Software object or object attribute
9. All block attributes in the Galaxy that are referenced by a specified set of block attributes indirectly via strategy declarations
10. All attribute names and references of a specified strategy (for example, declarations and user attributes).

# QueryFilter Command

The **QueryFilter** command allows you to specify filter criteria that will be used to determine what objects are to be processed. Multiple **QueryFilter** commands can be issued against the same **FilterName**, resulting in a filter with an implied **AND** between the multiple dissimilar conditions, and an implied **OR** between multiple identical conditions.

**Syntax:**

```
<QueryFilter    Filter="FilterName"
                Condition="FilterCondition"
                Value="FilterValue"
                Type="TypeValue"/>
```

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter. |
| FilterCondition | The filter condition used to construct the query that ultimately determines what objects actually get processed. To negate the filter condition, immediately precede it with an exclamation point. Valid values of **FilterCondition** are listed below. |
| FilterValue | The value to be used in conjunction with the **FilterCondition** to determine what actually gets processed. Refer to the description for **FilterCondition** for allowable values. |
| TypeValue | Specifies whether instances or templates are to be queried. Valid values are **Instances**, **Templates**, or **Both**. **TypeValue** defaults to **Both** if not specified. |

Valid values of FilterCondition are:

- **AssignedTo** – All objects satisfying this filter are assigned to the object specified by this tagname.
- **BasedOn** – All objects satisfying this filter are based on the object specified by this base template.
- **CheckedOutBy** – All objects satisfying this filter are checked out by this user ID.
- **CheckoutStatusIs** – All objects satisfying this filter have this checkout status. Valid values are:
    - **CheckedOutToMe**
    - **CheckedOutToSomeoneElse**
    - **NotCheckedOut**.
- **ContainedBy** – All objects satisfying this filter are contained by the object specified by this tagname.
- **CreatedBy** – All objects satisfying this filter are created by this user-id.
- **DeploymentStatusIs** – All objects satisfying this filter have this deployment status. Valid values are:
    - **Deployed**
    - **DeployedWithPendingChanges**
    - **NotDeployed**
- **DerivedOrInstantiatedFrom** – All objects satisfying this filter are derived or instantiated from the template specified by this tagname.

- **DiagramStatusIs** – All objects satisfying this filter have strategies that are up-to-date.Valid Values are:
    - **UpToDate**
- **HierarchicalNameLike** – All objects satisfying this filter have a hierarchical name that satisfies the specified LIKE clause.
- **HostEngineIs** – All objects satisfying this filter have a host engine specified by this tagname.
- **LastModifiedBy** – All objects satisfying this filter were last modified by this user-id.
- **NamedLike** – All objects satisfying this filter have a tagname that satisfies the specified LIKE clause.
- **NameEquals** – All objects satisfying this filter have a tagname exactly matching the specified value.
- **ValidationStatusIs** – All objects satisfying this filter have this validation status. Valid values are:
    - **Bad**
    - **Good**
    - **Unknown**
    - **Warning**
- **WithinSecurityGroup** – All objects satisfying this filter belong to the specified security group.

### Example:

For example, the query:

```
<DirectAccess>
<QueryFilter Filter="Filter1" Condition="ContainedBy"Value="COMPN
D_001" />
<QueryFilter Filter="Filter1" Condition="ContainedBy" Value="COMPN
D_002" />
<QueryFilter Filter="Filter1" Condition="ContainedBy" Value="COMPN
D_003" />
<QueryFilter Filter="Filter1" Condition="DiagramStatusIs" Value="!UpT
oDate" />
```

results in a query returning all the strategies that are in COMPND_001, COMPND_002, or COMPND_003 and that are not up-to-date.

### Example:

For example, the query:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280"/>
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270"/>
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$AW70P"/>
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="XYZ%"/>
<QueryFilter Filter="Filter1" Condition="!AssignedTo"
Value="Equip_Unit_001"/>
```

Results in a query returning all objects that are:

1. Based on **$FCP280**, **$FCP270**, **$AW70P,** or **$ZCP270**, and;
2. Named like XYZ% (that is, letterbugs beginning with the characters **XYZ**), and;
3. NOT assigned to equipment unit **Equip_Unit_001**.

# BlockQueryFilter Command

The **BlockQueryFilter** command allows you to specify filter criteria that will be used to determine what blocks or ECBs are to be processed. Multiple **BlockQueryFilter** commands can be issued against the same **FilterName**, resulting in a filter with an implied **AND** between multiple dissimilar conditions, and an implied **OR** between multiple identical conditions. Except for the **WhereUsed** report, a **BlockQueryFilter** is used in conjunction with a **QueryFilter** to determine which blocks are processed, and should specify the same filter name as the **QueryFilter**.

### Syntax:

```
<BlockQueryFilter   Filter="FilterName"
                    Condition="FilterCondition"
                    Value="FilterValue"/>
```

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter. |
| FilterCondition | The filter condition used to construct the query that ultimately determines what objects actually get processed. Valid values of **FilterCondition** for a block query are:<br>• **BasedOn** – All blocks/ECBs satisfying this filter are based on the base template specified by this base template.<br>• **ContainedBy** – All blocks satisfying this filter are contained by the strategy specified by this tagname.<br>**Note** The **ContainedBy** condition can be used only for the **WhereUsed** report.<br>• **DerivedOrInstantiatedFrom** – All blocks/ECBs satisfying this filter are derived or instantiated from the template specified by this tagname.<br>• **NamedLike** – All blocks/ECBs satisfying this filter have a tagname that satisfies the specified LIKE clause.<br>• **NameEquals** – All block/ECBs satisfying this filter have a tagname exactly matching the specified value.<br>• **HierarchicalNameLike –** All blocks/ECBs satisfying this filter have a contained name that satisfies the specified LIKE clause. |
| FilterValue | The value to be used in conjunction with the **FilterCondition** to determine what actually gets processed. Refer to the description for **FilterCondition** for allowable values. |

### Example:

For example, the query:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy_001"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AOUT"/>
```

Results in a query returning all blocks from strategy **Strategy_001** that are derived from **$AIN** or **$AOUT**.

Refer to the **ProduceReport** or **Export** commands for further examples.

# AttributeQueryFilter Command

The **AttributeQueryFilter** command allows you to specify filter criteria that will be used to determine what attributes are to be processed. Multiple **AttributeQueryFilter** commands can be issued against the same **FilterName**, resulting in a filter with an implied **AND** between multiple dissimilar conditions, and an implied **OR** between multiple identical conditions, with one exception. The Value="NonDefaultsOnly" condition is ANDed with the other AttributeType conditions.

An **AttributeQueryFilter** is meant to be used in conjunction with a **QueryFilter** and/or a **BlockQueryFilter** to determine which attributes get processed, and should specify the same filter name.

Attribute query filters are used to determine what attributes are reported on during execution of a **ProduceReport** command, or what attributes are exported during execution of an **Export** command.

### Syntax:

```
<AttributeQueryFilter    Filter="FilterName"
                         Condition="FilterCondition"
                         Value="FilterValue"/>
```

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter. |
| FilterCondition | The filter condition used to construct the query that determines what attributes to return. Valid values for an attribute filter condition are:<br>• **AttributeType** – The type of attributes returned will be determined by the attribute type specified by **FilterValue**.<br>• **AttributeName** – The attributes returned will be determined by attribute names matching that specified by **FilterValue**. |

| Attribute | Description |
|---|---|
| FilterValue | The filter value used to construct the query that ultimately determines what attributes actually get processed.<br>If an **AttributeType** filter is specified, valid values for **FilterValue** are:<br>• **Configurable** – All configurable attributes will be returned. Not necessary if the value **NotYetPropagated** is used.<br>• **Settable** – All settable attributes will be returned.<br>• **DataStore** – All data store attributes will be returned.<br>• **NonDefaultsOnly** – Only those attributes with values that are different from their default values will be returned.<br><br>(Valid values for FilterValue are continued on the next page.) |

| Attribute | Description |
|---|---|
| FilterValue (Cont.) | If an **AttributeType** filter is specified, valid values for **FilterValue** are:<br>(Continued from above.)<br>• **NotYetPropagated** – All attributes of a deployed block(s) that were not marked as "dirty" (that is, blocks which require redeployment to the controller) when a related attribute's value was changed. This could happen in two cases:<br>**1)** The attribute is a sink of a regular controller connection that was not properly marked dirty as a result of other strategy declaration changes.<br>**2)** The attribute is a sink of a data propagation connection and value of the source attribute has been changed.<br>An attribute is only returned by the use of **NotYetPropagated** if its block matches the following conditions:<br>• The block is deployed.<br>• The attribute is not marked as "dirty".<br>• The attribute is the sink of a regular controller connection or data propagation connection.<br>• If the attribute is the sink of a regular controller connection and the deployed C:B.P value is different from C:B.P value currently specified for the attribute.<br>• If the attribute is the sink of a data propagation connection and the deployed value is different from the value of the source of the connection.<br>**Configurable** is implied by **NotYetPropagated**, so if **NotYetPropagated** is used, **Configurable** is not necessary.<br>If an **AttributeName** filter is specified, the name of an attribute must exactly match the value specified for **FilterValue** for a match to occur. |

**Example:**

For example, the query:

```
<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="HSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="LSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="EO1"/>
```

Results in an attribute query returning all attributes that are named HSCO1, LSCO1, or EO1.

Refer to the **ProduceReport** or **Export** commands for further examples.

# Discussion of the LIKE Clause

Two filter conditions support the use of a LIKE clause for the filter value. These conditions are:

- **HierarchicalNameLike**
- **NamedLike**

The filter value for these two conditions uses a standard SQL LIKE syntax. Briefly, the string specified for the filter value can contain the following special characters:

- **%** – any string of zero or more characters.
    - NamedLike="AW%" will return all objects with tagnames that begin with the characters “AW”.
    - NamedLike="%AW%" will return all object instances that have the characters “AW” anywhere in the tagname.
    - NamedLike="$%" will return all templates with tagnames of any length.
- _ (underscore) – any single character.
    - NamedLike="AW000_" will return all objects with tagnames that begin with the characters “AW000” followed by any other single character.
- **[ ]** – any single character in the specified range.
    - NamedLike="[ACS]%" will return all objects with tagnames that begin with the characters “A”, “C” or “S”.
- **[^]** – any single character NOT in the specified range.
    - NamedLike="[^ACS]%" will return all objects with tagnames that do not begin with the characters “A”, “C”, or “S”.

A filter value using the LIKE syntax is not case sensitive (for example, **AW%** is the same as **aw%**).

# Query Filter Usage

Throughout this document, a mention is made wherever the use of query filters is supported. To optimize performance, you should take every precaution to construct query filters that result in the least number of objects returned by the query. This is especially true of any type of action resulting in permanent, unrecoverable activity, such as any of the **Delete** commands.

Taking the **AssignCompound** command as an example, the proper use of the query filter would look something like the following:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>
```

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%003"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0001"/>

<AssignCompound Filter="Filter1" Controller="CP0002"/>
```

In this example, all compounds ending with the characters **003** currently assigned to controller CP0001 would be assigned to controller CP0002. If the **BasedOn** condition were left off, it might result in a large number of objects being returned in the query that otherwise would not need to be. While Direct Access will process only compounds with the **AssignCompound** command, not specifying the **BasedOn** or **DerivedOrInstantiatedFrom** condition will cause extra processing and use memory that will adversely affect performance.

# Using Query Filters with Blocks and ECBs

Instances of blocks and ECBs are not ArchestrA objects, which places restrictions on how query filters can be used when querying for these types of objects. There are no restrictions when dealing with block or ECB derived templates, since they are true ArchestrA objects, so query filters can operate properly against them.

For example, when performing an operation such as **UpdateBlockAttribute**, all blocks within a strategy will be examined to see if they satisfy the query filter specified for the block. You should use the appropriate **BlockQueryFilter** query conditions to limit the types of blocks that are examined, as well as **NamedLike**, **NameEquals**, or **HierarchicalNameLike** conditions. The proper use of these five conditions should allow you to adequately indicate which blocks you wish the command to be executed against.

**Note** Only the **BasedOn**, **DerivedOrInstantiatedFrom**, **NamedLike**, **NameEquals**, and **HierarchicalNameLike** conditions are valid for block and/or ECB instances. All other conditions specified in a block or ECB instance query filter will result in an error message, and further processing for that command will halt.

**Tip** When querying for blocks and/or ECBs, the **NamedLike** and **NameEquals** condition applies to the block's tagname (or controller name) while the **HierarchicalNameLike** applies to the block's contained name.

### Examples:

Update the **HSCO1** attribute on all blocks derived from **$AIN** on all strategies derived from **$Strategy**, and contained within compound **MY_COMPND**:

```
<QueryFilter Filter="Filter1" Condition="DerivedOrInstantiatedFrom"
Value="$Strategy"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="MY_COMPND"/>

<BlockQueryFilter Filter="Filter1"
Condition="DerivedOrInstantiatedFrom" Value="$AIN"/>
```

```
<UpdateBlockAttribute Filter="Filter1" ParmName="HSCO1"
ParmValue="105.3"/>
```

Update the historization settings for the **ACHNGE** attribute on all **AIN** blocks with names beginning with the character **A** on all strategies derived from **$Strategy**, and contained within compound **MY_COMPND**:

```
<QueryFilter Filter="Filter1" Condition="DerivedOrInstantiatedFrom"
Value="$Strategy"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="MY_COMPND"/>

<BlockQueryFilter Filter="Filter1"
Condition="DerivedOrInstantiatedFrom" Value="$AIN"/>

<BlockQueryFilter Filter="Filter1" Condition="NamedLike"
Value="A%"/>

<UpdateBlockHistory Filter="Filter1" ParmName="ACHNGE"
Description="Test description" Deadband="2" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" Period="10" ScanRate="100"
OnMessage="Turned On" OffMessage="Turned Off"/>
```

Update the **DESCRP** attribute on all ECBs derived from **$ECB1_001** that are in compounds with names ending with **ECB** that are currently assigned to controller **FCP100**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="FCP100"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%ECB"/>

<BlockQueryFilter Filter="Filter2"
Condition="DerivedOrInstantiatedFrom" Value="$ECB1_001"/>

<UpdateECBAttribute Filter="Filter1" ParmName="DESCRP"
ParmValue="PUMP_ECB"/>
```

CHAPTER 5

# Timer Functions

This chapter describes timer functions, which allow you to manipulate timers when executing a command script. You can start and stop a named timer, view the time elapsed for a specific timer, and temporarily suspend the execution of Direct Access for a specified period of time.

## Contents



# Timer CSV File

A timer .csv file is created each time a **StartTimer** command is encountered within the command file. The name of this file is:

[*TimerName*].csv

and it is automatically placed into the C:\temp directory. If the directory C:\temp does not exist, it will be created for you.

This file contains an entry for each **ElapsedTime** command encountered within the command file for that timer. Each entry consists of:

- Current wall time (for correlation to the ArchestrA log file entries) in hh:mm:ss format.
- Total elapsed time since start of timer in hh:mm:ss format.
- Amount of memory (private bytes) used by the Direct Access application.
- Amount of memory (private bytes) used by the aaGR.exe application.
- Amount of memory (private bytes) used by the sqlservr.exe application.

The memory amounts are snapshots of memory taken at the time the **ElapsedTime** command was encountered.

The format of the file is such that you can easily open it within Microsoft Excel software and:

- Plot elapsed time versus amount of memory the three named processes are utilizing, or
- Insert a new column, perform a **FillDown > Series**, and plot entry versus elapsed time. The entry would obviously be highly dependent for that particular run of Direct Access, and would only be meaningful to you. It might mean the time it took to create a single specific type of object, or series of objects.

If the file already exists, it will be overwritten whenever a **StartTimer** command is encountered using a timer by that name.

Care should be taken not to use two or more **StartTimer** commands in the same command file with the same timer name, as the associated file will ultimately only contain the results of the most recent **StartTimer** and subsequent **ElapsedTime** commands.

# ElapsedTime Command

The **ElapsedTime** command allows you to view the time elapsed for a specific timer.

### Syntax:

```
<ElapsedTime     Name="TimerName"/>
```

| Attribute | Description |
|---|---|
| TimerName | The name of the timer for which the elapsed time is to be reported.<br>**TimerName** must be the name of a timer that has been started with the **StartTimer** command, or an error will result.<br>Reports on the elapsed time since the last **ElapsedTIme** command was issued, and the total time elapsed for this specific timer. |

### Examples:

Display the elapsed time for timer **MYTIMER**:

```
<ElapsedTime Name="MYTIMER"/>
```

### Examples with Output:

Script:

```
<StartTimer Name="MyTimer"/>
<Sleep Sec="10"/>
<ElapsedTime Name="MyTimer"/>
<Sleep Sec="10"/>
<ElapsedTime Name="MyTimer"/>
```

```
<Sleep Sec="10"/>
<ElapsedTime Name="MyTimer"/>
```

Output:

```
Starting timer MyTimer...
Sleeping for 0 min 10 sec...
Checking timer MyTimer...00:00:10.0000640 (00:00:10.0000640 total)
Sleeping for 0 min 10 sec...
Checking timer MyTimer...00:00:10.0000640 (00:00:20.0001280 total)
Sleeping for 0 min 10 sec...
Checking timer MyTimer...00:00:10.0000640 (00:00:30.0001920 total)
```

# Sleep Command

The **Sleep** command allows you to temporarily suspend the execution of Direct Access for a specified period of time. This can prove useful, for example, following a deployment, if you want to allow a few scan cycles to go by before proceeding.

### Syntax:

```
<Sleep  Min="Minutes" Sec="Seconds"/>
```

| Attribute | Description |
|---|---|
| Minutes | The number of minutes to suspend Direct Access. |
| Seconds | The number of seconds to suspend Direct Access. |

**Note** If neither **Minutes** nor **Seconds** is specified, or both are set to zero, the time will default to 1 second.

### Examples:

Suspend execution of Direct Access for 10 seconds:

```
<Sleep Min="0" Sec="10"/>
```

Suspend execution of Direct Access for 1 second:

```
<Sleep/>
```

# StartTimer Command

The **StartTimer** command allows you to start a named timer.

### Syntax:

```
<StartTimer  Name="TimerName"/>
```

| Attribute | Description |
|---|---|
| TimerName | The name of the timer to start.<br>If the name specified is not the name of an existing timer, an error will result. |

**Examples:**

Start a timer called **MYTIMER**:

```
<StartTimer Name="MYTIMER"/>
```

# StopTimer Command

The **StopTimer** command allows you to stop a specified timer.

**Syntax:**

```
<StopTimer        Name="TimerName"/>
```

| Attribute | Description |
|---|---|
| TimerName | The name of the timer to stop.<br>If the name specified is not the name of an existing timer, an error will result.<br>Once stopped, the timer will no longer be available for **ElapsedTime** commands. |

**Example:**

Stop the timer called **MYTIMER**:

```
<StopTimer Name="MYTIMER"/>
```

CHAPTER 6

# Assign Operations

This chapter describes assign operations, which allow you to assign a system component to another system components, for example, you can assign a compound to a controller, a controller to an equipment unit or Ethernet switch, and so forth.

## Contents

# AssignCompound Command

The **AssignCompound** command allows you to assign one or more compounds to a controller. This command is not valid for templates.

### Syntax:

```
<AssignCompound    Compound="CompoundName"
                   Controller="ControllerName"/>

                   Or

<AssignCompound    Filter="FilterName"
                   Controller="ControllerName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound being assigned. **CompoundName** must be the name of a compound instance, and should not be specified if **FilterName** is given. |
| ControllerName | The name of the controller to which the compound is being assigned. If **ControllerName** is missing or blank, the most recently referenced controller will be used. If no controller had been previously referenced within the XML data, an error will result. |
| FilterName | The name of the filter that will be used to determine what compounds get assigned to **ControllerName**. **FilterName** should not be specified if **CompoundName** is given. |

### Examples:

Assign compound **COMPND_001** to controller **CP2801**:

```
<AssignCompound Compound="COMPND_001"
Controller="CP2801"/>
```

Assign compound **CMP_ABC** to the most recently referenced controller:

```
<AssignCompound Compound="CMP_ABC"/>
```

Assign all compounds currently assigned to **CP0001** ending with the characters **003** to **CP0002**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%003"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0001"/>

<AssignCompound Filter="Filter1" Controller="CP0002"/>
```

# AssignController Command

The **AssignController** command allows you to assign one or more controllers to an equipment unit and/or switch. This command is not valid for templates.

**Syntax:**

```
<AssignController   Controller="ControllerName"
                    EquipUnit="EquipUnitName"
                    Node="NodeName"
                    Switch="SwitchName"
                    Port="PortNumber"/>

                    Or

<AssignController   Filter="FilterName"
                    EquipUnit="EquipUnitName"
                    Node="NodeName"
                    Switch="SwitchName"
                    Port="PortNumber"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller that is to be assigned to the equipment unit.<br>**ControllerName** should not be specified if **FilterName** is given. |
| EquipUnitName | The name of the equipment unit to which the controller is to be assigned.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced within the XML data, then a value for **SwitchName** or **NodeName** must be specified. |
| NodeName | The name of the node to which the controller is to be assigned.<br>**Note** AssignController command cannot accept Equip Name and node name together. It will accept either of the two. |
| SwitchName | The name of the switch with which the controller is to be associated.<br>If this attribute is missing or blank, then a value for **EquipUnitName** must be specified, or an error will result. |
| PortNumber | The number of the switch port to which the controller is to be associated.<br>If this attribute is missing or blank, the port number will not be set. |
| FilterName | The name of the filter that will be used to determine what controllers get assigned to **EquipUnitName** or **SwitchName**. **FilterName** should not be specified if **ControllerName** is given. |

**Examples:**

Assign controller **CPXY00** to equipment unit **EQUIP_UNIT_B** and switch **SW0001** and port **7**:

```
<AssignController Controller="CPXY00" EquipUnit="EQUIP_UNIT_B"
Switch="SW0001"Port="7" />
```

Assign controller **CPXY00** to nested equipment unit **EQUIP_UNIT_G** contained within equipment unit **EQUIP_UNIT_B**:

```
<AssignController Controller="CPXY00"
EquipUnit="EQUIP_UNIT_B.EQUIP_UNIT_G"/>
```

Assign all the controllers based on **$FCP280** in equipment unit **EquipUnit_1** to equipment unit **EquipUnit_2**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_1"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280"/>

<AssignController Filter="Filter1" EquipUnit="EquipUnit_2"/>
```

Assign controller **CP0001** to node **N00005**:

```
<AssignController Controller="CP0001" Node="N00005"/>
```

Assign same controller **CP0001** to node **N00005** and equipment unit **EQUIP_UNIT_B**:

```
<AssignController Controller="CP0001" EquipUnit="EQUIP_UNIT_B
"Node="N00005"/>
```

**Note** This command fails displaying the following message “Assigning Controller...cannot specify both Equipment and Node”.

# AssignDevice Command

The **AssignDevice** command allows you to assign one or more device instances to a controller and/or FBM/FCM.

You can assign a device directly to $TACM, $AB3B, $MG3B, and $FD3B. You cannot assign a device directly to $CP60 and $CP4B. They can be assigned through FCM/FBM.

**Syntax:**

```
<AssignDevice   Device="DeviceName"
                FBM="FBMName"
                Controller="ControllerName"
                Segment="SegmentNumber"
                ContainedName="ContainedName"
                PlantUnit="PlantUnitName"/>
```

Or

```
<AssignDevice   Filter="FilterName"
                FBM="FBMName"
                Controller="ControllerName"
                Segment="SegmentNumber"
                ContainedName="ContainedName"
                PlantUnit="PlantUnitName"/>
```

| Attribute | Description |
|---|---|
| DeviceName | The name of the device to assign. **DeviceName** is valid only for device instances.<br>This attribute should not be specified if **FilterName** is given. |
| FBMName | The name of the FBM to which this device is to be assigned (device instances only). **FBMName** may be left blank if **PlantUnitName** is specified.<br><br>**Note** When the device is assigned to an FBM, an ECB of the appropriate type (for example, ECB201) will be created automatically, and named the same as **DeviceName**. If the FBM is attached to a controller, then the ECB will also be placed automatically into the controller's ECB compound. |
| ControllerName | The name of the controller to which this device is to be assigned (device instances only). |
| SegmentNumber | The channel number corresponding to the channel on the FBM to which the device is to be assigned. |
| ContainedName | The contained name of the device (if different from the tagname) if being assigned to an FBM. If the contained name of the device is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other devices assigned to the same FBM, and should not be specified unless **FBMName** is specified.<br>For HART instances, the contained name and the tagname are the same.<br>This attribute will be ignored for HART devices.<br><br>**Note** If a filter is specified, then this value will be ignored. |
| PlantUnitName | The name of the plant unit to which this device is to be assigned. **PlantUnitName** may be left blank if **FBMName** is specified. |
| FilterName | The name of the filter that will be used to determine what devices get assigned to **FBMName** or **PlantUnitName**.<br>This attribute should not be specified if *DeviceName* is given. |

**Examples:**

Assign device **MY_DEVICE** to FBM **F02701** contained in controller **MYCP00**:

```
<AssignDevice Device="MY_DEVICE" FBM="MYCP00.F02701"/>
```

Assign Profibus device **MY_DEV** to segment **2** on the FBM222 instance **XYZ001** contained in controller **CP0100**, and at the same time, assign **MY_DEV** to plant unit **EAST_PLANT**:

```
<AssignDevice Device="MY_DEV" FBM="CP0100.XYZ001"
Segment="2" PlantUnit="EAST_PLANT" />
```

Assign Profibus device **MY_DEV** to plant unit **WEST_PLANT**, leaving its current FBM and segment assignment unchanged:

```
<AssignDevice Device="MY_DEV" PlantUnit="WEST_PLANT" />
```

Assign device **DEV001** to controller **CP0001**:

```
<AssignDevice Device="DEV001" Controller="CP0001"/>
```

Assign same device to **FBM001** and controller:

```
<AssignDevice Device="FBM001" Controller="CP0001"/>
```

**Note** This command fails displaying the following message “Assigning device...cannot specify both FBM and Controller”.

# AssignEquipUnit Command

The **AssignEquipUnit** command allows you to assign one or more equipment units to another, resulting in nested equipment units. This command is not valid for templates.

**Syntax:**

```
<AssignEquipUnit    EquipUnit="EquipUnitName"
                    Parent="ParentName"
                    ContainedName="ContainedName"/>
                    Or
<AssignEquipUnit    Filter="FilterName"
                    Parent="ParentName"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit being assigned. This attribute must be the name of an equipment unit instance, and should not be specified if **FilterName** is given. |
| ParentName | The name of the equipment unit that is to contain the equipment unit being assigned.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit had been previously referenced within the XML data, an error will result. |
| ContainedName | The contained name of the equipment unit (if different from the tagname) if being assigned to another equipment unit. If the contained name of the equipment unit is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other equipment units assigned to the same parent equipment unit.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what equipment units get assigned to **ParentName**. This attribute should not be specified if **EquipUnitName** is given. |

**Examples:**

Assign equipment unit **EQUIP_UNIT_B** to equipment unit **EQUIP_UNIT_A**:

```
<AssignEquipUnit EquipUnit="EQUIP_UNIT_B"
Parent="EQUIP_UNIT_A"/>
```

Assign equipment unit **EQUIP_UNIT_C** to the equipment unit **EQUIP_UNIT_B** which is already contained within **EQUIP_UNIT_A**:

```
<AssignEquipUnit EquipUnit="EQUIP_UNIT_C"
Parent="EQUIP_UNIT_A.EQUIP_UNIT_B"/>
```

Assign equipment unit **EQUIP_UNIT_Z** to the equipment unit **EQUIP_UNIT_B** and change its contained name to **EQUIP_UNIT_X**:

```
<AssignEquipUnit EquipUnit="EQUIP_UNIT_Z"
Parent="EQUIP_UNIT_B" ContainedName="EQUIP_UNIT_X"/>
```

Assign all equipment units with names beginning with the characters **EQUIP** that are currently contained within **EQUIP_002** to the equipment unit **EQUIP_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Equip_Unit"/>
```

```
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="EQUIP_002"/>

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="EQUIP%"/>

<AssignEquipUnit Filter="Filter1" Parent="EQUIP_001"/>
```

# AssignFBM Command

The **AssignFBM** command allows you to assign one or more FBMs to a controller, FCM or ISCM. This command is not valid for templates.

**Syntax:**

```
<AssignFBM      FBM="FBMName"
                Controller="ControllerName"
                FCM="FCMName"
                ISCM="ISCMName"
                ContainedName="ContainedName"/>

                Or

<AssignFBM      Filter="FilterName"
                Controller="ControllerName"
                FCM="FCMName"
                ISCM="ISCMName"/>
```

| Attribute | Description |
|---|---|
| FBMName | The name of the FBM being assigned. This attribute must be the name of an FBM instance, and should not be specified if **FilterName** is given. |
| ControllerName | The name of the controller to which the FBM is being assigned.<br>For 200 Series FBMs, the **ControllerName** must be the name of an FCP280 or FCP270. To assign a 200 Series FBM to a ZCP270, it must be associated with an FCM100.<br>If **ControllerName** is specified, **FCMName** must not be specified.<br>If neither **ControllerName** nor **FCMName** is specified, the FBM will be assigned to the most recently referenced controller. If no controller has been previously referenced, the FBM will be created, but will not be assigned to anything. |
| FCMName | The name of the FCM to which the FBM is being assigned.<br>If **FCMName** is specified, **ControllerName** must not be specified. |

| Attribute | Description |
|---|---|
| ISCMName | The name of the ISCM to which the FBM is being assigned. An FBM can only be assigned to a controller, an FCM or an ISCM, but no other combination of that assignment is allowed (i.e., cannot be assigned to two or more at the same time).<br>If **ISCMName** is specified, **FCMName** and **Controller Name** must not be specified. |
| ContainedName | The contained name of the FBM (if different from the tagname) if being assigned to a controller or FCM. If the contained name of the FBM is to be the same as the tagname, this attribute does not need to be specified. The contained name must be unique from all other modules assigned to the same controller or FCM. This attribute should not be specified unless **ControllerName** or **FCMName** is specified.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what FBMs get assigned to **ParentName**. This attribute should not be specified if **FBMName** is given. |

**Examples:**

Assign FBM **FBM001** to controller **CP2801**:

```
<AssignFBM FBM="FBM001" Controller="CP2801"/>
```

Assign FBM **F00123** to the most recently referenced controller:

```
<AssignFBM FBM="F00123"/>
```

Assign FBM **F00002** to the FCM **FC0001**, which has already assigned to controller **ZCP345**. Change its contained name to **F02011**:

```
<AssignFBM FBM="F00002" FCM="ZCP345.FC0001"
ContainedName=”F02011”/>
```

Assign all FBMs based on **$FBM01** that are currently contained by controller **CP0200**, that do NOT have letterbugs beginning with the letter **F** to controller **CP0300**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FBM01"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0200"/>

<QueryFilter Filter="Filter1" Condition="!NamedLike" Value="F%"/>

<AssignFBM Filter="Filter1" Controller="CP0300"/>
```

# AssignFCM Command

The **AssignFCM** command allows you to assign one or more FCMs to a controller. This can also be used to assign FBI and DCI to NonMesh Controllers. This command is not valid for templates.

**Note** This command enables you to assign DCM10, FBI10, FCM10, and FCM100.

### Syntax:

```
<AssignFCM    FCM="FCMName"
              Controller="ControllerName"
              ContainedName="ContainedName"/>
              Or
              Filter="FilterName"
              Controller="ControllerName"/>
```

| Attribute | Description |
|---|---|
| FCMName | The name of the FCM being assigned. This attribute must be the name of an FCM instance, and should not be specified if **FilterName** is given. |
| ControllerName | The name of the controller to which the FCM is being assigned.<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller had been previously referenced within the XML data, an error will result. |
| ContainedName | The contained name of the FCM (if different from the tagname) if being assigned to a controller. If the contained name of the FCM is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other modules assigned to the same controller.<br>This attribute should not be specified unless **ControllerName** is specified.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what FCMs get assigned to **ControllerName**. This attribute should not be specified if **FCMName** is given. |

### Examples:

Assign FCM **FC1200** to controller **ZCP271**:

```
<AssignFCM FCM="FC1200" Controller="ZCP271"/>
```

Assign FCM **FC1100** to the most recently referenced controller:

```
<AssignFCM FCM="FC1100"/>
```

Assign FCM **FC1700** to controller ZCP **ZCP345**. Change its contained name to **FC1800**:

```
<AssignFCM FCM="FC17070" Controller="ZCP345"
ContainedName=”FC1800”/>
```

Assign all FCM100s beginning with the letter **F** that are currently assigned to controller **CP0003** to controller **CP0004**.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$FCM100"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0003"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="F%"/>

<AssignFCM Filter="Filter1" Controller="CP0004"/>
```

Assign FCM **FBI100** of type FBI10 to controller **CP6000**.

```
<AssignFCM FCM="FBI100" Controller="CP6000"/>
```

Assign FCM **DCM200** of type DCM10 to controller **CP6000**.

```
<AssignFCM FCM="DCM200" Controller="CP6000"/>
```

# AssignFoxCTS Command

The **AssignFoxCTS** command allows you to assign one or more workstations to be hosted by FoxCTS Server.

### Syntax:

```
<AssignFoxCTS      Workstation="WorkstationName"
                   FoxCTSServer="FoxCTSServerName"/>
                   Or
<AssignFoxCTS      Filter="FilterName"
                   FoxCTSServer="FoxCTSServerName"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The name of the workstation to be associated with the indicated FoxCTS Server. This attribute must be the name of an instance of a workstation, and should not be specified if **FilterName** is given. |
| FoxCTSServer Name | The name of the FoxCTS Server which will be assigned for the workstation given in **WorkstationName**. If the workstation represented by **WorkstationName** is already assigned to this FoxCTS Server, the user will be informed. If the workstation represented by **WorkstationName** is already assigned to a different FoxCTS Server, the relationship will be removed from the old FoxCTS Server and added to the new one represented by **FoxCTSServerName**. If the FoxCTS Server named by **FoxCTSServerName** is not suitable to be a FoxCTS Server, an error will result. |
| FilterName | The name of the filter that will be used to determine what workstations get assigned to FoxCTSServer. This attribute should not be specified if **WorkstationName** is given. |

**Examples:**

Assign workstation **AW70P1** to be hosted by FoxCTSServer **AW70P2**:

```
<AssignFoxCTS Workstation="AW70P1" FoxCTSServer="AW70P2" />
```

Assign workstation **AW70P1** to be hosted by workstation **AW70P3** (previously hosted by **AW70P2**):

```
<AssignFoxCTS Workstation="AW70P1" FoxCTSServer="AW70P3"/>
```

Assign any workstations that are assigned to **EQUIP_UNIT_002** to be hosted by workstation **AW70P2**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_002"/>

<AssignFoxCTS Filter="Filter1" FoxCTSServer="AW70P2" />
```

Assign **DCM10** device to **CP60** controller:

```
<AssignFCM FCM="DCM200" Controller="CP6000" />
```

# AssignISCM Command

The **AssignISCM** command allows you to assign one or more ISCMs/ISCMRs to a controller or FCM100. This command is not valid for templates.

**Syntax:**

```
<AssignISCM    ISCM="ISCMName"
               Controller="ControllerName"
               ParentFCM="FCMName"
               ContainedName="ContainedName"/>
               Or
               Filter="FilterName"
               Controller="ControllerName"
               ParentFCM="FCMName"/>
```

| Attribute | Description |
|---|---|
| ISCMName | The name to assign this ISCM/ISCMR. |
| ControllerName | The name of the controller to which this ISCM/ISCMR is to be assigned (ISCM/ISCMR instances only).<br>If **ControllerName** is specified, **FCMName** must not be specified.<br>If neither **ControllerName** nor **FCMName** are specified, the ISCM/ISCMR will be assigned to the most recently referenced controller. If no controller has been previously referenced, the ISCM/ISCMR will be created, but will not be assigned to anything. |
| FCMName | The name of the FCM to which the ISCM/ISCMR is being assigned.<br>If **FCMName** is specified, **ControllerName** must not be specified. |
| ContainedName | The contained name of the ISCM/ISCMR (if different from the tagname) if being assigned to a controller or FCM. If the contained name of the ISCM/ISCMR is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other modules assigned to the same controller or FCM.<br>This attribute should not be specified unless **ControllerName** or **FCMName** is specified.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what ISCMs/ISCMRs get assigned to **ParentName**. This attribute should not be specified if **ISCMName** is given. |

**Examples:**

Assign ISCM **ISC5M1** to controller **FCP272**:

```
<AssignISCM ISCM="ISC5M1" Controller="FCP272"/>
```

Assign ISCM **ISC5MA** to the most recently referenced controller:

```
<AssignISCM ISCM="ISC5MA"/>
```

Assign ISCM **ISC6M1** to the FCM **FC0001**, which has already assigned to controller **ZCP272**. Change its contained name to **FC06M1**:

```
<AssignISCM ISCM="ISC6M1" ParentFCM="ZCP272.FC0001"
ContainedName="FC06M1"/>
```

Assign all ISCMs based on **$ISCM** that are currently contained by controller **CP0200**, that do NOT have letterbugs beginning with the letter **F** to controller **CP0300**.

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ISCM"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0200"/>

<QueryFilter Filter="Filter1" Condition="!NamedLike" Value="F%"/>

<AssignISCM Filter="Filter1" Controller="CP0300"/>
```

# AssignNetworkPrinter Command

The **AssignNetworkPrinter** command allows you to assign network printer objects to equipment units. This command is not valid for templates.

### Syntax:

```
<AssignNetwork Printer="PrinterName"
Printer EquipUnit="EquipUnitName"
Switch="SwitchName"/>

Or

<AssignNetwork Filter="FilterName"
Printer EquipUnit="EquipUnitName"
Switch="SwitchName"/>
```

| Attribute | Description |
|---|---|
| PrinterName | The name of the network printer to be assigned to the equipment unit or switch. This attribute should not be specified if **FilterName** is given. |
| EquipUnitName | The optional name of the equipment unit to which this network printer is to be assigned. If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced within the XML data, a value for **SwitchName** must be specified. |
| SwitchName | The name of the switch to which the network printer is to be assigned. If this attribute is missing or blank, a value for **EquipUnitName** must be specified, or an error will result. |
| FilterName | The name of the filter that will be used to determine what network printers get assigned to **EquipUnitName** and/or **SwitchName**. This attribute should not be specified if **PrinterName** is given. |

### Examples:

Assign network printer **PR0001** to equipment unit **EQUIP_UNIT_B** and switch **SW0001**:

```
<AssignNetworkPrinter Printer="PR0001"
EquipUnit="EQUIP_UNIT_B" Switch="SW0001"/>
```

Assign network printer **PR0001** to nested equipment unit **EQUIP_UNIT_G** contained within equipment unit **EQUIP_UNIT_B**:

```
<AssignNetworkPrinter Printer="PR0001"
EquipUnit="EQUIP_UNIT_B.EQUIP_UNIT_G"/>
```

Assign all network printers with letterbugs beginning with **PR** that are currently assigned to **EQUIP_UNIT_001** to equipment unit **EQUIP_UNIT_002**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$PRTNET"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="PR%"/>

<AssignNetworkPrinter Filter="Filter1"
EquipUnit="EQUIP_UNIT_002"/>
```

# AssignNode Command

The **AssignNode** command allows you to assign one or more nodes to the equipment unit. It does not allow nested Node assignment. This command is not valid for templates.

### Syntax:

```
<AssignNode     Node="NodeName "
                Parent="ParentName"
                ContainedName="ContainedName"/>

                Or

<AssignNode     Filter="FilterName"
                Parent="ParentName"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node being assigned.<br>This attribute must be the name of a node instance, and should not be specified if **FilterName** is given. |
| ParentName | The name of the equipment unit that should contain the node being assigned.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit had been previously referenced within the XML data, an error occurs. |

| Attribute | Description |
|---|---|
| ContainedName | The contained name of the node if it is being assigned to the equipment unit.<br>If the contained name of the node is the same as the tagname, this attribute does not need to be specified. The contained name must be unique from all other nodes assigned to the same equipment unit. This attribute should not be specified unless **ParentName** is specified.<br>**Note** If a filter is specified, this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what nodes get assigned to **ParentName**. This attribute should not be specified if **NodeName** is given. |

**Examples:**

Assign node **NODE_B** to equipment unit **EQUIP_UNIT_A**:

```
<AssignNode Node="NODE_B" Parent="EQUIP_UNIT_A"/>
```

Assign Node **NODE_Z** to the equipment unit **EQUIP_UNIT_B** and change its contained name to **NODE_X**:

```
<AssignNode Node="NODE _Z" Parent="EQUIP_UNIT_B"
ContainedName="NODE_X"/>
```

Assign all nodes with names beginning with the characters **N00** that are currently contained within **EQUIP_002** to equipment unit **EQUIP_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Node"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="EQUIP_002"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="N00%"/>

<AssignNode Filter="Filter1" Parent="EQUIP_001"/>
```

# AssignObject Command

The **AssignObject** command allows you to assign one or more objects to another object. This command works for either a hosting or a containment relationship.

**Note** Although it is possible to assign derived templates and instances of Control Software objects (that is, strategies, compounds, controllers, and so forth), it is not advisable to use this command for that purpose. Use the appropriate **Assign** command to actually assign the Control Software objects.

**Note** For templates, a containment relationship will always be established. For instances, an attempt will first be made to establish a hosting relationship. If that fails, then a containment relationship will be attempted.

### Syntax:

```
<AssignObject   Object="ObjectName"
                Parent="ParentName"
                ContainedName="ContainedName"/>

                Or

<AssignObject   Filter="FilterName"
                Parent="ParentName"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object being assigned. This attribute may be an object template or instance, and should not be specified if **FilterName** is given. |
| ParentName | The name of the parent object to which the original object is being assigned. |
| ContainedName | The contained name of the object (if different from the tagname). If the contained name of the object is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other objects assigned to the same parent. This attribute should not be specified unless **ParentName** is specified.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what objects get assigned to **ParentName**. This attribute should not be specified if **ObjectName** is given. |

### Examples:

Assign Boolean template **$Boolean_001** to template **$Boolean_002**. In this case, since they are templates, the association will be one of containment.

```
<AssignObject Object="$Boolean_001" Parent="$Boolean_002"/>
```

Assign all instances of ArchestrA application objects based on **$Double** with names beginning with the characters **MyNum** to the **$Float** instance **Float_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Double"/>

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="MyNum%"/>

<AssignObject Filter="Filter1" Parent="Float_001"/>
```

# AssignPlantUnit Command

The **AssignPlantUnit** command allows you to assign one or more plant units to another, resulting in nested plant units. This command is not valid for templates.

### Syntax:

```
<AssignPlantUnit    PlantUnit="PlantUnitName"
                    Parent="ParentName"
                    ContainedName="ContainedName"/>

                    Or

<AssignPlantUnit    Filter="FilterName"
                    Parent="ParentName"/>
```

| Attribute | Description |
|---|---|
| PlantUnitName | The name of the plant unit being assigned. This attribute must be the name of an plant unit instance. |
| ParentName | The name of the plant unit that is to contain the plant unit being assigned.<br>If this attribute is missing or blank, the most recently referenced plant unit will be used. If no plant unit had been previously referenced within the XML data, an error will result. |
| ContainedName | The contained name of the plant unit (if different from the tagname) if being assigned to another plant unit. If the contained name of the plant unit is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other plant units assigned to the same parent plant unit, and should not be specified unless **ParentName** is specified.<br>**Note** If a filter is specified, then this value will be ignored. |
| FilterName | The name of the filter that will be used to determine what plant units get assigned to **ParentName**. This attribute should not be specified if **PlantUnitName** is given. |

### Examples:

Assign plant unit **PLANT_UNIT2** to plant unit **PLANT_UNIT1**:

```
<AssignPlantUnit PlantUnit="PLANT_UNIT2"
Parent="PLANT_UNIT1"/>
```

Assign plant unit **PLANT_UNIT3** to the equipment unit **PLANT_UNIT2** which is already contained within **PLANT_UNIT1**:

```
<AssignPlantUnit PlantUnit="PLANT_UNIT3"
Parent="PLANT_UNIT1.PLANT_UNIT2"/>
```

Assign plant unit **PLANT_UNIT6** to plant unit **PLANT_UNIT5**, change its contained name to **MY_UNIT**:

```
<AssignPlantUnit PlantUnit="PLANT_UNIT6"
Parent="PLANT_UNIT5" ContainedName="MY_UNIT”/>
```

Assign all plant units beginning with the characters **MY** to the plant unit **PLANT_UNIT_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Plant_Unit"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="MY%"/>

<AssignPlantUnit Filter="Filter1" Parent="PLANT_UNIT_001"/>
```

# AssignSoftwareHost Command

The **AssignSoftwareHost** command allows you to assign one or more hardware objects to be hosted by a software host (workstation).

### Syntax:

```
<AssignSoftwareHost     HardwareToHost="HardwareName"
                        Host="HostName"/>
                        Or
<AssignSoftwareHost     Filter="FilterName"
                        Host="HostName"/>
```

| Attribute | Description |
|---|---|
| HardwareName | The name of the hardware object to be associated with the indicated software host. This attribute must be the name of an instance of a controller or network printer, and should not be specified if **FilterName** is given. |
| HostName | The name of the workstation that will be responsible for hosting the hardware named in **HardwareName**.<br>If the hardware represented by **HardwareName** is already hosted by this workstation, the user will be informed.<br>If the hardware represented by **HardwareName** is already hosted by a different workstation, the software host relationship will be removed from the old workstation and added to the new one represented by **HostName**.<br>If the workstation named by **HostName** is not suitable to be a software host, an error will occur.<br>If the **HardwareName** is an instance of the **PRTNET**, the **HostName** must be an instance of **WSTA70/WSVR70**, otherwise an error will be reported. |
| FilterName | The name of the filter that will be used to determine what hardware gets assigned to **HostName**. This attribute should not be specified if **HardwareName** is given. |

**Examples:**

Assign controller **CPXY00** to be hosted by workstation **AW0001**:

```
<AssignSoftwareHost HardwareToHost="CPXY00" Host="AW0001"/>
```

Assign controller **CPXY00** to be hosted by workstation **AW0002** (previously hosted by **AW0001**):

```
<AssignSoftwareHost HardwareToHost="CPXY00" Host="AW0002"/>
```

Assign any controllers that are assigned to **EQUIP_UNIT_002** to be hosted by workstation **AW0001**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_002"/>

<AssignSoftwareHost Filter="Filter1" Host="AW0001"/>
```

# AssignStrategy Command

The **AssignStrategy** command allows you to assign one or more strategies to a compound or a parent strategy.

**Syntax:**

```
<AssignStrategy     Strategy="StrategyName"
                    Compound="CompoundName"
                    ParentStrategy="ParentName"
                    X="XPosition”
                    Y="YPosition"/>

                    Or

<AssignStrategy     Filter="FilterName"
                    Compound="CompoundName"
                    ParentStrategy="ParentName”/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being assigned. This attribute should not be specified if **FilterName** is given. |
| ParentName | The name of the parent strategy to which this strategy is to be assigned. This attribute is valid for both instances and templates.<br>If **ParentName** is specified, **CompoundName** must not be specified (that is, a strategy can belong to another strategy or a compound, but cannot belong to both). |

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to which this strategy is to be assigned (instances only).<br>Only top-level strategies may be assigned to a compound, that is, a child strategy cannot be assigned to a compound.<br>If **CompoundName** is specified, **ParentName** must not be specified.<br>If neither **ParentName** nor **CompoundName** is specified, **CompoundName** will default to the name of the most recently referenced compound. |
| XPosition | The X coordinate of the strategy appearance object if the strategy being created is a child strategy. This attribute is valid only if **ParentName** is specified without **FilterName**; otherwise, it is ignored. |
| YPosition | The Y coordinate of the strategy appearance object if the strategy being created is a child strategy. This attribute is valid only if **ParentName** is specified without **FilterName**; otherwise, it is ignored. |
| FilterName | The name of the filter that will be used to determine what strategies get assigned to **ParentName** or **CompoundName**. This attribute should not be specified if **StrategyName** is given. |

**Examples:**

Assign strategy **OUTER_LOOP** to compound **COMPND_001**:

```
<AssignStrategy Strategy="OUTER_LOOP"
Compound="COMPND_001"/>
```

Assign strategy **INNER_LOOP** to strategy **OUTER_LOOP**:

```
<AssignStrategy Strategy="INNER_LOOP"
ParentStrategy="OUTER_LOOP"/>
```

Assign strategy **CASCADE** to child strategy **OUTER_LOOP.INNER_LOOP**:

```
<AssignStrategy Strategy="CASCADE"
ParentStrategy="OUTER_LOOP.INNER_LOOP"/>
```

Assign strategy **Strategy_2** to strategy **Strategy_1**. Locate the strategy at **X=2.5**, **Y=3.0** within the canvas of **Strategy_1**:

```
<AssignStrategy Strategy="Strategy_2" ParentStrategy="Strategy_1"
X="2.5" Y="3.0" />
```

Assign all strategies contained in compound **COMPND_002** beginning with **STR** to **COMPND_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_002"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="STR%"/>

<AssignStrategy Filter="Filter1" Compound="COMPND_001"/>
```

# AssignSwitch Command

The **AssignSwitch** command allows you to assign one or more switches to an equipment unit and/or another switch. This command is not valid for templates.

### Syntax:

```
<AssignSwitch   Switch="SwitchName "
                EquipUnit="EquipUnitName"
                Switch2="Switch2Name"/>

                Or

<AssignSwitch   Filter="FilterName "
                EquipUnit="EquipUnitName"
                Switch2="Switch2Name"/>
```

| Attribute | Description |
|---|---|
| SwitchName | The name of the switch that is to be assigned to the equipment unit and/or another switch.<br>This attribute should not be specified if **FilterName** is given. |
| EquipUnitName | The name of the equipment unit to which the switch is to be assigned.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced within the XML data, then a value for **Switch2Name** must be specified. |
| Switch2Name | The name of the other switch to which the switch is to be assigned.<br>If this attribute is missing or blank, then a value for **EquipUnitName** must be specified, or an error will result. |
| FilterName | The name of the filter that will be used to determine what switches get assigned to **EquipUnitName** and/or **Switch2Name**. This attribute should not be specified if **SwitchName** is given. |

### Examples:

Assign switch **SWCH01** to equipment unit **EQUIP_UNIT_B** and switch **SW0001**:

```
<AssignSwitch Switch="SWCH01" EquipUnit="EQUIP_UNIT_B"
Switch2="SW0001" />
```

Assign switch **SWCH01** to nested equipment unit **EQUIP_UNIT_G** contained within equipment unit **EQUIP_UNIT_B**:

```
<AssignSwitch Switch=" SWCH01"
EquipUnit="EQUIP_UNIT_B.EQUIP_UNIT_G"/>
```

Assign all switches beginning with the characters **SW** to the equipment unit **EQUIP_UNIT_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$IA_SWITCH"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="SW%"/>

<AssignSwitch Filter="Filter1" EquipUnit="EQUIP_UNIT_001"/>
```

# AssignSystemMonitor Command

The **AssignSystemMonitor** command allows you to assign one or more hardware objects to be monitored by a system monitor.

### Syntax:

```
<AssignSystemMonitor   HardwareToMonitor="HardwareName"
                       Monitor="MonitorName"/>

                       Or

<AssignSystemMonitor   Filter="FilterName"
                       Monitor="MonitorName"/>
```

| Attribute | Description |
|---|---|
| HardwareName | The name of the hardware object to be associated with the indicated system monitor. This attribute must be the name of an instance of a workstation, controller, network printer (PRTNET) or switch, and should not be specified if **FilterName** is given. |
| MonitorName | The name of the system monitor that will be responsible for monitoring the hardware named in **HardwareName**. If the hardware represented by **HardwareName** is already monitored by this monitor, the user will be informed. If the hardware represented by **HardwareName** is already monitored by a different monitor, the system monitor relationship will be removed from the old monitor and added to the new one represented by **MonitorName**. |
| FilterName | The name of the filter that will be used to determine what hardware gets assigned to **MonitorName**. This attribute should not be specified if **HardwareName** is given. |

### Examples:

Assign controller **CPXY00** to be monitored by **SYSMO1**:

```
<AssignSystemMonitor HardwareToMonitor="CPXY00"
Monitor="SYSMO1"/>
```

Assign controller **CPXY00** to be monitored by **SYSMO2** (previously monitored by **SYSMO1**):

```
<AssignSystemMonitor HardwareToMonitor="CPXY00"
Monitor="SYSMO2"/>
```

Assign all FCP280s currently assigned to **EQUIP_UNIT_001** to be monitored by **SYSMO4**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280"/>
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>
<AssignSystemMonitor Filter="Filter1" Monitor="SYSMO4"/>
```

# AssignWorkstation Command

The **AssignWorkstation** command allows you to assign one or more workstations to an equipment unit and/or switch. The command also allows you to assign an ATS to a Node. This command is not valid for templates.

**Note** This command also enables you to assign an ATS.

### Syntax:

```
<AssignWorkstation  Workstation="WorkstationName "
                    EquipUnit="EquipUnitName"
                    Node="NodeName"
                    Switch="SwitchName"/>
                    Or
<AssignWorkstation  Filter="FilterName"
                    EquipUnit="EquipUnitName"
                    Node="NodeName"
                    Switch="SwitchName"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The name of the workstation that is to be assigned to the equipment unit or switch. It is also the name of the ATS to be assigned to the node.<br>This attribute should not be specified if **FilterName** is given. |
| EquipUnitName | The name of the equipment unit to which the workstation is to be assigned.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced within the XML data, then a value for **SwitchName** or **NodeName** must be specified. This will not be used for assigning ATSs. |
| NodeName | The name of the node to which the workstation is to be assigned. It is also the name of the node to which the ATS will be assigned. |

| Attribute | Description |
|---|---|
| SwitchName | The name of the switch to which the workstation is to be assigned.<br>If this attribute is missing or blank, then a value for **EquipUnitName** must be specified, or an error will occur.<br>**SwitchName** should not be used to assign an ATS to a node. When you try to assign an ATS to a Switch, an error occurs. |
| FilterName | The name of the filter that will be used to determine what workstations get assigned to **EquipUnitName** and/or **SwitchName** and/or Node. This attribute should not be specified if **WorkstationName** is given. |

**Note** The AssignWorkstation command cannot accept both the **EquipUnitName** and the **NodeName** together. It will accept either one of the two.

### Examples:

Assign workstation **STA001** to equipment unit **EQUIP_UNIT_B** and switch **SW0001**:

```
<AssignWorkstation Workstation="STA001"
EquipUnit="EQUIP_UNIT_B" Switch="SW0001" />
```

Assign workstation **STA001** to nested equipment unit **EQUIP_UNIT_G** contained within equipment unit **EQUIP_UNIT_B**:

```
<AssignWorkstation Workstation="STA001"
EquipUnit="EQUIP_UNIT_B.EQUIP_UNIT_G"/>
```

Assign all workstations with letterbugs beginning with **AW** that are currently assigned to **EQUIP_UNIT_001** to equipment unit **EQUIP_UNIT_002**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$AW70P"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="AW%"/>

<AssignWorkstation Filter="Filter1" EquipUnit="EQUIP_UNIT_002"/>
```

Assign all workstations with letterbugs beginning with **ATS** to node **NODE01**:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="ATS%"/>

<AssignWorkstation Filter="Filter1" Node="NODE01"/>
```

Assign workstation **ATS001** to node **N00005**:

```
<AssignWorkstation Workstation="ATS001" Node="N00005"/>
```

CHAPTER 7

# Create Operations

This chapter describes create operations, which allow you to create new derived Control Software objects, instances, or templates.

## Contents

# CopyStrategy Command

The **CopyStrategy** command allows you to make an exact copy of a strategy. The new strategy will be derived from the same strategy template as the original. All nested strategies will also be copied.

### Syntax:

```
<CopyStrategy      OldStrategy="OldStrategyName"
                   NewStrategy="NewStrategyName"/>
```

| Attribute | Description |
|---|---|
| OldStrategyName | The Tagname of the strategy to be copied. Strategy templates may also be specified. |
| NewStrategyName | The Tagname of the copy. If the **NewStrategyName** begins with '$', then a strategy template will be created. |

### Examples:

Copy strategy **Strategy1** to **Strategy2**:

```
<CopyStrategy OldStrategy="Strategy1" NewStrategy="Strategy2"/>
```

# CreateBlock Command

The **CreateBlock** command allows you to create a control block within a control strategy, or a block template.

### Syntax:

```
<CreateBlock       Template="TemplateName"
                   Block="BlockName"
                   Strategy="StrategyName"
                   X="XPosition"
                   Y="YPosition"
                   Source="BlockSource"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the block will be derived (for example, **$MY_AIN**). |
| BlockName | The name of the block to be created. This attribute may specify the name of an instance or template.<br>If **StrategyName** is specified, then this should reflect the name of a block instance. If **StrategyName** is not specified, and no strategy was previously referenced within the XML, then this should reflect the name of a block template. |

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy to which this block is to be assigned. This attribute may be the name of an existing template (for example, **$My_Strategy**) or instance.<br>This attribute should not be specified if creating a block template.<br>For a block instance, if this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| XPosition | The X coordinate of the block appearance object (defaults to 0.0). |
| YPosition | The Y coordinate of the block appearance object (defaults to 0.0). |
| BlockSource | If provided, this attribute provides the full path to the file(s) containing the source for the block to be created. The value of this attribute must be the full path to the source file(s), including base filename without any file extension. Direct Access software will append the appropriate extension, depending on the block type being created, according to the following list:<br><br>Sequence Blocks:<br>.s HLBL or SFC source<br>.g SFC graphics<br>.k SFC graphics<br>.vsd SFC graphics<br>PLB Blocks:<br>.p Ladder source<br>.vsd Ladder graphics<br>Logic Blocks:<br>.rpn Step text<br>.vsd Graphical logic |

**Examples:**

Create a block called **MY_AIN** derived from the template **$AIN** in strategy instance **MyStrategy1**:

```
<CreateBlock Template="$AIN" Block="MY_AIN"
Strategy="MyStrategy1"/>
```

Create a block called **MY_AOUT** derived from the template **$AOUT** in strategy template **$MyStrategy1**:

```
<CreateBlock Template="$AOUT" Block="MY_AOUT"
Strategy="$MyStrategy1"/>
```

Create a block called **MY_PID** derived from the template **$MY_PID**, and assign it to the strategy most recently referenced within the XML command file:

```
<CreateBlock Template="$MY_PID" Block="MY_PID"/>
```

Create a block template called **$MY_AIN**, derived from **$AIN**:

```
<CreateBlock Template="$AIN" Block="$MY_AIN"/>
```

Create a block called **MY_DEP** derived from the template **$DEP** in strategy template **$MyStrategy1**, with HLBL source in the file C:\Temp\InFusionSupport\BLOCK01.s. Note that the extension ".s" is not included in the Source attribute value.

```
<CreateBlock Template="$DEP" Block="MY_DEP
Strategy="$MyStrategy1"
Source="C:\Temp\InFusionSupport\BLOCK01"/>
```

# CreateBlockAddressCxn Command

The **CreateBlockAddressCxn** command allows you to create a sink address connection, which will ultimately be resolved to a C:B.P address at deployment time.

### Syntax:

```
<CreateBlockAddressCxn    Strategy="StrategyName"
                          Sink="SinkBlockName"
                          SinkParm="SinkParmName"
                          SinkValue="SinkValue"/>
```

| Attribute | Description |
|---|---|
| Strategy | The name of the strategy in which the sink block resides. This attribute may be a template or instance.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| Sink | The name of the block in which the sink attribute exists. This name should reflect the contained name of the block within the strategy.<br>If the block is not found, an error will result. |
| SinkParm | The name of the attribute to be updated in the sink block. This attribute must be the name of a valid control attribute. If not found, an error will result. |
| SinkValue | The value to which the sink attribute is to be updated. This attribute needs to be a valid connection for that block. When written to the sink attribute, the relative reference "**MyContainer.**" will be prepended to the value specified.<br>**Note** Although similar in nature to the ability to update an attribute value via **UpdateAttribute**, the value placed into an attribute value using the **UpdateAttribute** command will not resolve properly to a C:B.P address at deployment time. |

**Examples:**

Create a block address connection on the **MEAS** attribute of block **MY_PIDA** contained in strategy **MyStrategy1**. Set the connection reference to point to the **PNT** attribute of the block **MY_AIN1** in the same strategy.

```
<CreateBlockAddressCxn Strategy="MyStrategy1" Sink="MY_PIDA"
SinkParm="MEAS"
SinkValue="MY_AIN1.PNT"/>
```

**Note** After finding the sink block and attribute, the program will create a connection reference **MyContainer.MY_AIN1.PNT**, and the attribute will be flagged as having an address connection which will be resolved to a C:B.P reference at deployment time (vs. a data value, which will not be resolved at deployment time).

**Note** Setting Block Address Connections for Block Attributes within Strategy Templates using Direct Access does not behave the same as when this operation is done from within the IDE. When Connections are made in Strategy Templates via the IDE, the sink parameter is automatically locked. This is not the case when a connection is made to a block in a Strategy Template via Direct Access. The lock status of the sink parameter is not changed automatically. If this behavior is required, it will be necessary to add a LockBlockAttribute command.

# CreateCompound Command

The **CreateCompound** command allows you to create a compound within the Galaxy, and optionally assign it to a controller.

**Syntax:**

```
<CreateCompound   Template="TemplateName"
                  Compound="CompoundName"
                  Controller="ControllerName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the compound will be derived (for example, **$MY_COMPND**).<br>If this attribute is missing or blank, the compound template will default to $**COMPND**. |

| Attribute | Description |
|---|---|
| CompoundName | The name to assign this compound.<br>If the name begins with a dollar sign (**$**), a compound template will be created. |
| Controller | The name of the controller by which this compound will be hosted (compound instances only).<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller has been previously referenced, the compound will be created, but not assigned to anything.<br>If specified, the controller name may not reference a controller template.<br>This attribute should not be specified if a compound template is being created. |

**Examples:**

Create a compound called **COMPND_001** derived from **$MY_COMPND**:

```
<CreateCompound Template="$MY_COMPND"
Compound="COMPND_001"/>
```

Create a compound called **COMPND_001** derived from **$COMPND**, and assign it to controller **CP2801**:

```
<CreateCompound Compound="COMPND_001"
Controller="CP2801"/>
```

Create a compound template called **$MY_COMPND**, derived from **$COMPND**:

```
<CreateCompound Compound="$MY_COMPND"/>
```

# CreateCompoundAddressCxn Command

The **CreateCompoundAddressCxn** command allows you to create a sink address connection, which will ultimately be resolved to a C:B.P address at deployment time. This command is not supported for Compound Templates.

**Syntax:**

```
<CreateCompound    Sink="SinkCompoundName"
AddressCxn         SinkParm="SinkParmName"
                   SinkValue="SinkValue"/>
```

| Attribute | Description |
|---|---|
| SinkCompoundName | The name of the compound in which the sink attribute exists. If the compound is not found, an error will occur. |
| SinkParm | The name of the attribute to be updated in the sink compound.This attribute must be the name of a valid control attribute. If not found, an error will occur. |
| SinkValue | The value to which the sink attribute is to be updated. This attribute needs to be a valid connection for that compound.<br><br>**Note** Although similar to the **UpdateCompoundAttribute** command, a connection value placed into an attribute value using the **UpdateCompoundAttribute** command will not be resolved properly to a C:B.P address at deployment time. |

**Examples:**

Create an address connection on the **CINHIB** attribute of compound **MYCOMP**. Set the connection reference to point to the **CINHIB** attribute of the compound **MYOTHERCOMP**.

```
<CreateCompoundAddressCxn Sink="MYCOMP" SinkParm="CINHIB"
SinkValue="MYOTHERCOMP.CINHIB"/>
```

Create an address connection on the **CINHIB** attribute of compound **MYCOMP**. Set the connection reference to point to the **Out output declaration** of the strategy **Strategy1 in compound MYOTHERCOMP**.

```
<CreateCompoundAddressCxn Sink="MYCOMP" SinkParm="CINHIB"
SinkValue="MYOTHERCOMP.Strategy1.Out"/>
```

**Note** After finding the sink compound and attribute, the program creates a connection reference to the sink values specified above and the attribute is flagged as having an address connection which will be resolved to a C:B.P reference at deployment time (vs. a data value, which will not be resolved at deployment time).

# CreateController Command

The **CreateController** command allows you to create a controller instance.

**Syntax:**

```
<CreateController    Template="TemplateName"
                     Controller="ControllerName"
                     Node="NodeName"
                     EquipUnit="EquipUnitName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the controller will be derived (for example, **$FCP280**). |
| Controller | The name of the controller to create. This attribute must be the name of an instance; the creation of controller templates is not supported.<br>Although allowed, it is not recommended that you create an FCP280 or FCP270 instance with a name that does not end in '00'. |
| NodeName | The optional name of the node to which this controller is to be assigned. If specified, the attribute must be the name of an existing instance.<br><br>**Note** The CreateController command cannot accept both the **EquipUnitName** and the **NodeName** together. It accepts either one of the two. |
| EquipUnitName | The optional name of the equipment unit to which this controller is to be assigned. If specified, this attribute must be the name of an existing instance; the creation of controllers within equipment unit templates is not supported. |

**Examples:**

Create a controller derived from **$FCP280**, called **CP2800**:

```
<CreateController Template="$FCP280" Controller="CP2800"/>
```

Create a controller derived from **$ZCP270** called **MY_ZCP**, and assign it to the equipment unit **EQUIP_UNIT_A**:

```
<CreateController Template="$ZCP270" Controller="MY_ZCP"
EquipUnit="EQUIP_UNIT_A"/>
```

Create a controller derived from **$CP60** called **CP0001**, and assign it to the node **N00005**:

```
<CreateController Template="$CP60" Controller="CP0001"
Node="N00005"/>
```

# CreateDeclaration Command

The **CreateDeclaration** command allows you to create a declaration within a strategy, and optionally update the appearance object for the strategy to reflect the declaration.

**Syntax:**

```
<CreateDeclaration    Decl="DeclName"
                      Type="DeclType"
                      Strategy="StrategyName"
                      RefreshAO="RefreshAO"/>
```

| Attribute | Description |
|---|---|
| DeclName | The name of the declaration to create within the strategy. |
| DeclType | The type of declaration to create.<br>Allowable values are **Input** or **Output**. |
| StrategyName | The name of the strategy in which this declaration is to be created. This attribute may be the name of an existing template (for example, **$My_Strategy**) or instance.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |
| RefreshAO | Specifies whether or not the appearance object for the strategy should be updated to reflect the new declarations. Allowable values are **True** or **False**. If not specified, **False** is assumed.<br><br>**Note** This is a very expensive operation, and if you are adding several declarations to a strategy, you should specify it as **True** only on the last declaration. |

**Examples:**

Create an input declaration called **My_Input** on strategy **Strategy_001**:

```
<CreateDeclaration Decl="My_Input" Type="Input"
Strategy="Strategy_001"/>
```

Create an output declaration called **My_Output** on strategy **My_Strategy1** and when done, update the appearance object for the strategy:

```
<CreateDeclaration Decl="My_Output" Type="Output"
Strategy="My_Strategy1" RefreshAO="True"/>
```

# CreateDevice Command

The **CreateDevice** Command allows you to create device instances in the Network view and device templates, that are immediately derived from $HART/$FF_Device/$Profibus, if a DD/EDD/GSD file is provided.

It allows you to create a device within the Galaxy, and optionally assign it to an FBM and/or associate it with a plant unit. CreateDevice command does not support creation of DTM templates and HART/FF/Profibus/DTM instances in the DTM Network view.

**Syntax:**

```
<CreateDevice   Template="TemplateName"
                Device="DeviceName"
                ContainedName="ContainedName"
                DDFile="DDFileName"
                FBM="FBMName"
                Controller="ControllerName"
                Segment="SegmentNumber"
                PlantUnit="PlantUnitName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template the device will be derived from (for example, **$My_Device**). |
| DeviceName | The name of the device to create. This will become the tagname of the device.<br>If the name begins with a dollar sign (**$**), a device template will be created derived from the template specified by **TemplateName**.<br>If a template immediately derived from **$Profibus** or **$HART** is being created and **DeviceName** is set to **$Default**, the template's name will be determined by data contained within the GSD or HART DD file. Otherwise, the template will be named whatever is specified by **DeviceName**.<br>If an instance based on **$Profibus or $HART** is created, and **DeviceName** is set to **Default**, the name of the device will be that as specified by its naming mask. Otherwise, the naming mask will be overridden and the device instance will be renamed to **DeviceName**.<br>If a first level template is derived from **$FF_Device**, the templates's name will be determined by data contained in the DD file.<br>If a first level template is derived from **$DeviceNet**, the templates's name will be determined by data contained in the EDS file.<br>While assigning the device to an FBM, **DeviceName** must be unique from all other devices assigned to the same FBM. |
| ContainedName | The contained name of the device if being assigned to an FBM. If the contained name of the device is to be the same as the tagname, this attribute does not need to be specified. The contained name must be unique from all other devices assigned to the same FBM, and should not be specified unless **FBMName** is specified.<br>For HART instances, the contained name and the tagname are the same. This attribute will be ignored for HART devices. |

| Attribute | Description |
|---|---|
| DDFileName | The fully qualified name (path and file name with extension) of the DD, EDD or GSD file that will be used to bind to this device. This attribute is valid for device templates immediately derived from **$HART**. If such a template is created, and DDFileName is missing, an error is reported. This attribute is also used for Profibus and **DeviceNet** devices. For Profibus devices, this is the GSD file which defines the device type. For DeviceNet, this is the EDS file that defines the device type. This attribute is used only if the fieldbus base template (**$Profibus** for Profibus and **$DeviceNet** for DeviceNet) is specified in the **TemplateName** attribute. It is ignored for all other templates when device instances are being created.<br><br>**Note** The HART object implementation allows you to create one or more templates using the same DD file for a device. The DD association process also allows for creating templates using a DD for the same device but with a higher or lower DD revision number. In such cases, you are asked to decide how to proceed, because it might affect the already defined templates and instances.<br><br>The CreateDevice command in DirectAccess only supports the creation of templates:<br>• Using a new DD file for a new device template<br>• Using the same DD file as already used for another template<br>Any changes in the DD template association which affect other templates are not supported and result in an error. |
| FBMName | The name of the FBM to which this device is to be assigned (device instances only).<br>If left blank, and a device instance is being created, then the device will be created and assigned to segment or port number 1 of the most recently referenced FBM. If no FBM has been previously referenced, then the device will be created, but not assigned to anything.<br>If specified, then **FBMName** may not reference an FBM template.<br><br>**Note** When the device is assigned to an FBM, an ECB of the appropriate type (for example, ECB201) will be created automatically, and named the same as **DeviceName**. If the FBM is attached to a controller, then the ECB will also be placed automatically into the controller's ECB compound. |

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller to which this device is to be assigned (device instances only).<br>If left blank and a device instance is being created, the device will be created and assigned to the most recently referenced controller. If no FBM has been previously referenced, the device will be created, but not assigned to anything.<br><br>**Note** When the device is assigned to a controller, an ECB of the appropriate type (for example, ECB21) will be created automatically, and named the same as the **DeviceName**. The ECB will also be placed automatically into the controller's ECB compound. |
| SegmentNumber | The segment/port number to which the device will be assigned after it has been created. This attribute is valid for Profibus device instances only if **FBMName** is specified. It is also the channel number on a HART FBM corresponding to the channel on the HART FBM to which the device is to be assigned after being created.<br>If no SegmentNumber is defined, then the device will be created but it will be located in the Unassigned Hardware folder. |
| PlantUnitName | The name of the plant unit with which this device is to be associated.<br>This attribute is valid for Profibus device instances. |
| Address | The device address. Used for Profibus and DeviceNet devices. Valid values for Profibus devices are 0-126. Valid values for DeviceNet devices are 0-63. |

### Generic Device Examples:

Create a device called **MY_DEV** derived from **$DEV_ECB18** and assign it to the **FBM FBM001**:

```
<CreateDevice Template="$DEV_ECB18" Device="MY_DEV"
FBM="FBM001"/>
```

Create a device template called **$MY_DEVICE** derived from **$DEV_ECB41**:

```
<CreateDevice Template="$MY_DEVICE" Device="$DEV_ECB41"/>
```

### Profibus Device Examples:

Create a Profibus device template and bind it to GSD file C:\GSDFiles\37_ABB_9521.gsd. Allow Control Editors to name the template based on data obtained from the GSD file:

```
<CreateDevice Device="$Default" Template="$Profibus"
DDFile="C:\GSDFiles\37_ABB_9521.gsd" />
```

Create a Profibus device template and bind it to GSD file C:\GSDFiles\37_ABB_9521.gsd. Override the Control Editors name generated from the GSD file with **$MY_PROFIBUS_DEV**:

```
<CreateDevice Device="$MY_PROFIBUS_DEV" Template="$Profibus"
DDFile=" C:\GSDFiles\37_ABB_9521.gsd”/>
```

Create a Profibus device instance derived from **$ABB_DRIVES_ABB#B** and override its default naming mask to name the device **MY_DEV**. Assign the device to Port **2** on FBM **XYZ001** (an FBM222), which has been assigned to controller **CP0100**:

```
<CreateDevice Device="MY_DEV"
Template=”$ABB_DRIVES_ABB#B" FBM="CP0100.XYZ001"
Segment="2"/>
```

Create a Profibus device instance derived from **$ABB_DRIVES_ABB#B**. Allow Control Editors to name the new object according to its pre-defined naming mask. Assign the device to Port **2** on FBM **XYZ001** (an FBM222), which has been assigned to controller **CP0100**:

```
<CreateDevice Device="Default" Template=”$ABB_DRIVES_ABB#B"
FBM="CP0100.XYZ001" Segment="2"/>
```

Create a device called **MY_DEV** derived from **$DEV_ECB18** and assign it to the controller **CP0001**:

```
<CreateDevice Template="$DEV_ECB18" Device=”MY_DEV"
Controller="CP0001"/>
```

### HART Device Examples:

Create a HART device template and bind it to a DD file C:\DDFiles\000000\0007\0101.fm8. Allow Control Editors to name the template based on the data obtained from the DD file:

```
<CreateDevice Device="$Default" Template="$HART"
DDFile="C:\DDFiles\000000\0007\0101.fm8"/>
```

Create a HART device template and bind it to a DD file C:\DDFiles\000000\0007\0101.fm8. Override the Control Editors name generated from the DD file with **$MY_HART_DEV**:

```
<CreateDevice Device="$MY_HART_DEVICE" Template="$HART"
DDFile="C:\DDFiles\000000\0007\0101.fm8"/>
```

Create a HART device instance derived from **$PR_Electronics_Pretop#01** and override its default naming mask to name the device **MY_DEV**. Assign the device to Segment 2 on FBM **XYZ001** (an FBM214) that has been assigned to controller **CP0100**:

```
<CreateDevice Device="MY_DEV"
Template="="$PR_Electronics_Pretop#01" FBM="CP0100.XYZ001"
Segment="2"/>
```

Create a HART device instance derived from **$PR_Electronics_Pretop#01**. Allow Control Editors to name the new object according to its predefined naming mask. Assign the device to Segment 2 on FBM **XYZ001** (an FBM214) that has been assigned to controller **CP0100**:

```
<CreateDevice Device="Default"
Template="="$PR_Electronics_Pretop#01" FBM="CP0100.XYZ001"
Segment="2"/>
```

# CreateECB Command

The **CreateECB** command allows you to create a user-defined ECB template. This command is used only for creating ECB templates. ECB instances cannot be created using this command. ECB instances are only created when their corresponding hardware is created and assigned to a controller.

### Syntax:

```
<CreateECB      Template="TemplateName"
                ECB="ECBName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining ECB template from which the newly created ECB template will be derived. |
| ECBName | The name of the ECB template to be created. The name supplied must begin with "$". |

### Examples:

Create an ECB template named **ECB200_NEW**, derived from **$ECB200**.

```
<CreateECB Template="$ECB200" ECB="$ECB200_NEW" />
```

# CreateECBAddressCxn Command

The **CreateECBAddressCxn** command allows you to create a sink address connection, which will ultimately be resolved to a C:B.P address at deployment time. This command is not supported for ECB Templates

### Syntax:

```
<CreateECB      Compound="SinkCompoundName"
AddressCxn      Sink="SinkECBName"
                SinkParm="SinkParmName"
                SinkValue="SinkValue"/>
```

| Attribute | Description |
|---|---|
| SinkCompoundName | The name of the compound in which the sink ECB exists. If this attribute is not supplied, the most recently referenced compound will be used. |
| SinkECBName | The name of the ECB in which the sink attribute exists. If the ECB is not found, an error will result. |

| Attribute | Description |
|---|---|
| SinkParm | The name of the attribute to be updated in the sink ECB. This attribute must be the name of a valid control attribute. If not found, an error will result. |
| SinkValue | The value to which the sink attribute is to be updated. This attribute needs to be a valid connection for that block.<br><br>**Note** Although similar to the **UpdateECBAttribute** command, a connection value placed into an attribute value using the **UpdateECBAttribute** command will not resolve properly to a C:B.P address at deployment time. |

**Example:**

Create an address connection on the **SSMOHL** attribute of ECB **CP0100_ECB:MYECB70**. Set the connection reference to point to the **BCALCO** attribute of the block **MYCOMP.Strategy1.BLOCK1**.

```
<CreateECBAddressCxnCompound="CP0100_ECB" Sink="MYECB70"
SinkParm="SSMOHL"
SinkValue="MYCOMP.Strategy1.BLOCK1.BCALCO/>
```

**Note** After finding the sink compound, ECB and attribute, the program will create a connection reference to **MYCOMP.Strategy1.BLOCK1.BCALCO**, and the attribute will be flagged as having an address connection which will be resolved to a C:B.P reference at deployment time (vs. a data value, which will not be resolved at deployment time).

# CreateEquipUnit Command

The **CreateEquipUnit** command allows you to create an equipment unit within the Galaxy, and optionally assign it to another equipment unit. Only equipment unit instances can be created. The creation of equipment unit templates is not supported.

**Syntax:**

```
<CreateEquipUnit   EquipUnit="EquipUnitName"
                   Parent="ParentName"
                   ContainedName="ContainedName"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit to create. |
| ParentName | The optional name of the equipment unit that the equipment unit being created is to be assigned to. This allows nested equipment units to be created.<br>If specified, it must be the name of an existing instance; the creation of equipment units within equipment unit templates is not supported.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced, the equipment unit will be created, but not assigned to anything. |
| ContainedName | The contained name of the equipment unit if being assigned to another equipment unit. If the contained name of the equipment unit is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other equipment units assigned to the same equipment unit.<br>This attribute should not be specified unless **ParentName** is specified. |

**Examples:**

Create an equipment unit called **EQUIP_UNIT_A**:

```
<CreateEquipUnit EquipUnit="EQUIP_UNIT_A"/>
```

Create an equipment unit called **EQUIP_UNIT_B** and assign it to **EQUIP_UNIT_A**:

```
<CreateEquipUnit EquipUnit="EQUIP_UNIT_B"
Parent="EQUIP_UNIT_A"/>
```

Create an equipment unit called **EQUIP_UNIT_C** and assign it to **EQUIP_UNIT_B** which is already contained within **EQUIP_UNIT_A**:

```
<CreateEquipUnit EquipUnit="EQUIP_UNIT_C"
Parent="EQUIP_UNIT_A.EQUIP_UNIT_B"/>
```

Create an equipment unit called **EQUIP_UNIT_Z** and assign it to **EQUIP_UNIT_X**. Rename the contained name **EQUIP_UNIT_Z** to **MY_EQUIP_UNIT**:

```
<CreateEquipUnit EquipUnit="EQUIP_UNIT_Z"
Parent="EQUIP_UNIT_X" ContainedName="MY_EQUIP_UNIT"/>
```

# CreateFBM Command

The **CreateFBM** command allows you to create an FBM within the Galaxy, and optionally assign it to a controller, FCM or ISCM.

### Syntax:

```
<CreateFBM      Template="TemplateName"
                FBM="FBMName"
                FCM="FCMName"
                ISCM="ISCMName"
                Controller="ControllerName"
                ContainedName="ContainedName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the FBM will be derived (for example, **$FBM228**). |
| FBMName | The tagname of the FBM to create. |
| FCMName | The name of the FCM to which this FBM is to be assigned.<br>This should be specified only if associating the FBM with a ZCP270. If the FBM is to be associated with an FCP280 or FCP270, then **FCMName** should not be specified.<br>If specified, the FCM name may not reference an FCM template. |
| ISCMName | The name of the ISCM to which the FBM is being assigned. An FBM can only be assigned to a controller, an FCM or an ISCM, but no other combination of that assignment is allowed (i.e., cannot be assigned to two or more at the same time).<br>If **ISCMName** is specified, **FCMName** and **Controller Name** must not be specified. |

| Attribute | Description |
|---|---|
| Controller | The name of the controller to which this FBM is to be assigned.<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller has been previously referenced, the FBM will be created, but not assigned to anything.<br><br>**Note** When the FBM is assigned to an FCM or controller, an ECB of the appropriate type (for example, ECB200) will be created automatically, and named the same as **FBMName**. The ECB will also automatically be placed into the controller's ECB compound. |
| ContainedName | The contained name of the FBM. If the contained name of the FBM is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other FBMs assigned to that controller or FCM, and should not be specified unless **Controller** is specified. |

### Examples:

Create an FBM201 called **F00001** and assign it to the FCP280 controller **CP2800**:

```
<CreateFBM Template="$FBM201" FBM="F00001"
Controller="CP2800"/>
```

Create an FBM228 called **F00003** and assign it to FCM **FC0003**:

```
<CreateFBM Template="$FBM228" FBM="F00003" FCM="FC0003"/>
```

Create an FBM214 called **F00005** and assign it to ISCM **ISC0M1**:

```
<CreateFBM Template="$FBM214" FBM="F00005"
ISCM="ISC0M1"/>
```

Create an FBM205 called **F00010** and assign it to the most recently referenced controller.

```
<CreateFBM Template="$FBM205" FBM="F00010"/>
```

Create an FBM201 called **F00023** and assign it to the FCP280 controller **CP2800**. Give it a contained name of **F00101**:

```
<CreateFBM Template="$FBM201" FBM="F00023"
Controller="CP2800" ContainedName="F00101"/>
```

# CreateFCM Command

The **CreateFCM** command allows you to create an FCM within the Galaxy, and optionally assign it to a controller.

**Note** This command enables you to create DCM and FBI as well as FCM.

### Syntax:

```
<CreateFCM      Template="TemplateName"
                FCM="FCMName"
                ContainedName="ContainedName"
                Controller="ControllerName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the FCM will be derived (for example, **$FCM100**). |
| FCMName | The name to assign this FCM. |
| Controller | The name of the controller to which this FCM is to be assigned.<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller has been previously referenced, the FCM will be created, but not assigned to anything.<br>**Note** When the FCM is assigned to a controller, an ECB of the appropriate type (for example, ECB210) will be created automatically, and named the same as **FCMName**. The ECB will also be placed automatically into the controller's ECB compound. |
| ContainedName | The contained name of the FCM. If the contained name of the FCM is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other FCMs and FBMs assigned to that controller, and should not be specified unless **Controller** is specified |

### Examples:

Create an FCM100 called **FC0000** and assign it to controller **CP2801**:

```
<CreateFCM Template="$FCM100" FCM="FC0000"
Controller="CP2801"/>
```

Create an FCM100 called **FC3000** and assign it to the most recently referenced controller:

```
<CreateFCM Template="$FCM100" FCM="FC3000"/>
```

Create an FCM100 called **FC2000** and assign it controller **CPXY00** and give it a contained name of **FC1010**:

```
<CreateFCM Template="$FCM100" FCM="FC2000"
Controller="CPXY00" ContainedName="FC1010"/>
```

Create a DCM10 called **DC0000** and assign it controller **CP6001**.

```
<CreateFCM Template="$DCM100" FCM="DC0000"
Controller="CP6001"/>
```

# CreateISCM Command

The **CreateISCM** command allows you to create an ISCM within the Galaxy, and optionally assign it to an FCP280 or FCP270 control processor or FCM100.

### Syntax:

```
<CreateISCM     Template="TemplateName"
                ISCM="ISCMName"
                ContainedName="ContainedName"
                Controller="ControllerName"
                ParentFCM="FCMName" />
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the ISCM will be derived (for example, **$ISCM**/**$ISCMR**). |
| ISCMName | The name to assign this ISCM/ISCMR. |
| ControllerName | The name of the controller to which this ISCM/ISCMR is to be assigned (ISCM/ISCMR instances only).<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller has been previously referenced, the ISCM/ISCMR will be created, but not assigned to anything.<br>If specified, the controller name may not reference a controller template.<br>**Note** When the ISCM/ISCMR is assigned to a controller, an ECB of the appropriate type (for example, ECB200/ECB202) will be created automatically, and named the same as **ISCMName**. The ECB will also be placed automatically into the controller's ECB compound. |

| Attribute | Description |
|---|---|
| FCMName | The name of the FCM to which this ISCM/ISCMR is to be assigned (ISCM/ISCMR instances only).<br>This should be specified only if associating the ISCM/ISCMR with a ZCP270. If the ISCM/ISCMR is to be associated with an FCP280 or FCP270, then **FCMName** should not be specified.<br>If specified, the FCM name may not reference an FCM template. |
| ContainedName | The contained name of the ISCM/ISCMR (if different from the tagname). If the contained name of the ISCM/ISCMR is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other ISCMs and FBMs assigned to that controller or FCM, and should not be specified unless **Controller** or **FCM** is specified. |

**Examples:**

Create an ISCM/ISCMR called **ISC0M1/ISCM0MA** and assign it to controller **CP2801**:

```
<CreateISCM Template="$ISCM" ISCM="ISC0M1"
Controller="CP2801"/>

<CreateISCM Template="$ISCMR" ISCM="ISC0MA"
Controller="CP2801"/>
```

Create an ISCM/ISCMR called **ISC1M1/ISC1MA** assign it to communication module F00100/F00200:

```
<CreateISCM Template="$ISCM" ISCM="ISC1M1"
ParentFCM="F00100"/>

<CreateISCM Template="$ISCMR" ISCM="ISC1MA"
ParentFCM="F00200"/>
```

Create an ISCM/ISCMR called **ISC2M1/ISC2MA** and assign it to the most recently referenced controller:

```
<CreateISCM Template="$ISCM" ISCM="ISC2M1"/>

<CreateISCM Template="$ISCMR" ISCM="ISC2MA"/>
```

Create an ISCM called **ISCM3M1** and assign it Controller **CP2801** and give it a contained name of **CPM3M1**:

```
<CreateISCM Template="$ISCM" ISCM="ISC3M1"
Controller="CP2801" ContainedName="CPM3M1"/>
```

Create an ISCMR called **ISCM4M1** and assign it FCM100 **F00400** and give it a contained name of **FCM4M1**:

```
<CreateISCM Template="$ISCMR" ISCM="ISC4M1"
ParentFCM="F00400" ContainedName="FCM4M1"/>
```

# CreateNetworkPrinter Command

The **CreateNetworkPrinter** command allows you to create the network printer object PRTNET.

### Syntax:

```
<CreateNetwork   Template="PRTNET"
Printer          Printer="NetworkPrinterName"
                 EquipUnit="EquipUnitName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the network printer will be derived. It must be **$PRTNET**. |
| NetworkPrinter Name | The name of the network printer to be created. Creation of network printer within templates is not supported. |
| EquipUnitName | The optional name of the equipment unit to which this network printer is to be assigned. If specified, the attribute must be the name of an existing instance; the creation of network printers within equipment unit templates is not supported. |

### Examples:

Create a network printer called **PR0001** derived from **$$PRTNET**:

```
<CreateNetworkPrinter Template="$PRTNET" Printer="PR0001"/>
```

Create a network printer called **PR0001** derived from **$PRTNET**, and assign it to equipment unit **EQUIP_UNIT_A**:

```
<CreateNetworkPrinter Template="$PRTNET" Printer="PR0001"
EquipUnit="EQUIP_UNIT_A"/>
```

# CreateNode Command

The **CreateNode** command allows you to create a node within the Galaxy and assign it to an equipment unit. Only node instances can be created. The creation of node templates is not supported.

### Syntax:

```
<CreateNode      Node="NodeName"
                 Parent="ParentName"
                 ContainedName="ContainedName"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node to create. The name of the node must be entered in uppercase. If it is not, Direct Access converts it to uppercase automatically. |
| ParentName | The optional name of the equipment unit to which the node being created is to be assigned.<br>If specified, it must be the name of an existing instance. The creation of nodes within equipment unit templates is not supported.<br>If this attribute is missing or blank, the most recently referenced equipment unit will be used. If no equipment unit has been previously referenced, the node will be created, but not assigned to anything. |
| ContainedName | The contained name of the node, if it is assigned to the equipment unit. If the contained name of the node is the same as the tagname, this attribute need not be specified.<br>The contained name must be unique from all other nodes assigned to the same equipment unit. This attribute should not be specified unless **ParentName** is specified. |

**Examples:**

Create a node called **NODE_A**:

```
<CreateNode Node="NODE_A"/>
```

Create a node called **NODE_A** and assign it to the Equipment unit called **EQUIP_UNIT_A**:

```
<CreateNode Node="NODE_A"Parent="EQUIP_UNIT_A"/>
```

Create a node called **NODE_Z** and assign it to the Equipment unit called **EQUIP_UNIT_Z**. Contained name of the node will change from **NODE_Z** to **NODE_T**:

```
<CreateNode Node = "NODE_Z"Parent="EQUIP_UNIT_X"
ContainedName="NODE_T"/>
```

# CreateObject Command

The **CreateObject** command allows you to create an ArchestrA object within the Galaxy, and optionally assign it to an Area.

**Note** Although it is possible to create derived templates and instances of the Control Software objects (that is, strategies, compounds, controllers, and so forth), it is not advisable to use this command for that purpose. Use the appropriate **Create** command to actually create the Control Software objects.

**Syntax:**

```
<CreateObject   Template="TemplateName"
                Object="ObjectName"
                Area="AreaName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the object will be derived (for example, **$Integer**). |
| ObjectName | The name to assign this object.<br>If the name begins with a dollar sign (**$**), an object template will be created. |
| AreaName | The name of the ArchestrA area to which this object is assigned.<br>If left blank, the new object will be assigned to either the **Unassigned Area**, or to the default Area, if one is set. |

**Examples:**

Create an object called **Integer01** derived from **$Integer**. In this example, **Integer01** will be assigned to the default area object within ArchestrA.

```
<CreateObject Template="$Integer" Object="Integer01"/>
```

Create an object called **MyFloat** derived from **$Float**, and assign it to area **Area_001**:

```
<CreateObject Template="$Float" Object="MyFloat"
Area="Area_001"/>
```

# CreatePlantUnit Command

The **CreatePlantUnit** command allows you to create a plant unit within the Galaxy, and optionally assign it to another plant unit. Only plant unit instances can be created. The creation of plant unit templates is not supported.

**Syntax:**

```
<CreatePlantUnit   PlantUnit="PlantUnitName"
                   Parent="ParentName"
                   ContainedName="ContainedName"/>
```

| Attribute | Description |
|---|---|
| PlantUnitName | The name of the plant unit to create. |
| ParentName | The optional name of the plant unit that the plant unit being created is to be assigned to. This allows nested plant units to be created.<br>If specified, this attribute must be the name of an existing instance; the creation of plant units within plant unit templates is not supported.<br>If this attribute is missing or blank, the most recently referenced plant unit will be used. If no plant unit has been previously referenced, the plant unit will be created, but not assigned to anything. |
| ContainedName | The contained name of the plant unit. If the contained name of the plant unit is to be the same as the tagname, this attribute does not need to be specified.<br>The contained name must be unique from all other and plant units assigned to the same parent plant unit, and should not be specified unless **ParentName** is specified |

**Examples:**

Create a plant unit called **Plant_Unit_A**:

```
<CreatePlantUnit PlantUnit="Plant_Unit_A"/>
```

Create a plant unit called **Plant_Unit_B** and assign it to **Plant_Unit_A**:

```
< CreatePlantUnit PlantUnit="Plant_Unit_B" Parent="Plant_Unit_A"/>
```

Create a plant unit called **Plant_Unit_C** and assign it to **Plant_Unit_B** which is already contained within **Plant_Unit_A**:

```
< CreatePlantUnit PlantUnit="Plant_Unit_C"
Parent="Plant_Unit_A.Plant_Unit_B"/>
```

# CreateSoftwarePackage Command

The **CreateSoftwarePackage** command allows you to create a software package on an instance of a workstation.

**Syntax:**

```
<CreateSoftwarePackage  SoftwarePackage="PackageName"
                        Workstation="StationName"/>
```

| Attribute | Description |
|---|---|
| PackageName | The name of the software package to create (for example, **ASMON7**.<br>This attribute must be the name of an optional software package, or an error will result (required software packages are already pre-created when the workstation was created).<br>Must be the name of a software package capable of running on that workstation, or an error will result. |
| StationName | The name of the workstation for which the new software package is to be created.<br>This attribute must be the name of a workstation instance. The creation of software packages on a workstation template is not supported. |

**Examples:**

Create a software package called **ASMON7** (system monitor) for the workstation **SW0001**:

```
<CreateSoftwarePackage SoftwarePackage="ASMON7"
Workstation="SW0001"/>
```

# CreateStrategy Command

The **CreateStrategy** command allows you to create a strategy within the Galaxy, and optionally assign it to a compound, or in the case of a child strategy, assign it to another strategy acting as a parent.

**Syntax:**

```
<CreateStrategy     Template="TemplateName"
                    Strategy="StrategyName"
                    Compound="CompoundName"
                    ParentStrategy="ParentName"
                    X="XPosition”
                    Y="YPosition"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the strategy will be derived (for example, **$My_Strategy**).<br>If this attribute is missing or blank, the strategy template will default to **$Strategy**. |
| StrategyName | The name to assign this strategy.<br>If the name begins with a dollar sign (**$**), a strategy template will be created. |

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to which this strategy is to be assigned (instance only).<br>If **CompoundName** is specified, **ParentName** must not be specified.<br>If neither **ParentName** nor **CompoundName** is specified, **CompoundName** will default to the name of the most recently referenced compound. If no compound has been previously referenced, the strategy will be created, but not assigned to anything |
| ParentName | The name of the parent strategy to which this strategy is to be assigned.This attribute is valid for both instances and templates.<br>If **ParentName** is specified, **CompoundName** must not be specified (that is, a strategy can belong only to another strategy or to a compound, but cannot belong to both). |
| XPosition | The X coordinate of the strategy appearance object if the strategy being created is a child strategy.<br>This attribute is valid only if **ParentName** is specified; otherwise, it is ignored. |
| YPosition | The Y coordinate of the strategy appearance object if the strategy being created is a child strategy.<br>This attribute is valid only if **ParentName** is specified; otherwise, it is ignored. |

**Examples:**

Create a strategy called **MY_STRAT1** derived from **$Strategy**:

```
<CreateStrategy Strategy="MY_STRAT1"/>
```

Create a strategy called **Strategy_001** derived from **$My_Strategy**, and assign it to compound **COMPND_001**:

```
<CreateStrategy Template="$My_Strategy" Strategy="Strategy_001"
Compound="COMPND_001" />
```

Create a strategy template called **$MY_STRAT2**, derived from **$MY_STRAT1**:

```
<CreateStrategy Template="$MY_STRAT1"
Strategy="$MY_STRAT2"/>
```

Create a child strategy called **Inner_Loop** derived from **$Cascade** which will be parented by strategy **Outer_Loop**:

```
<CreateStrategy Template="$Cascade" Strategy="Inner_Loop"
ParentStrategy="Outer _Loop"/>
```

Create strategy **Strategy_2** and assign it to strategy **Strategy_1**. Locate the strategy at **X=2.5**, **Y=3.0** within the canvas of **Strategy_1**:

```
< CreateStrategy Strategy="Strategy_2" ParentStrategy="Strategy_1"
X="2.5" Y="3.0" />
```

# CreateSwitch Command

The **CreateSwitch** command allows you to create a switch instance and optionally assign it to an equipment unit.

### Syntax:

```
<CreateSwitch        Template="TemplateName"
                     Switch="SwitchName"
                     EquipUnit="EquipUnitName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template from which the switch will be derived (for example, **$IA_Switch**). |
| SwitchName | The name of the switch to create. This attribute must be the name of an instance; the creation of switch templates is not supported. |
| EquipUnitName | The optional name of the equipment unit to which this switch is to be assigned. If specified, this attribute must be the name of an existing instance; the creation of switches within equipment unit templates is not supported. |

### Examples:

Create a switch named **SW0001**, assigned to equipment unit **UNIT1**:

```
<CreateSwitch Template="$IA_SWITCH" Switch="SW0001"
EquipUnit="UNIT1"/>
```

# CreateUserAttribute Command

The **CreateUserAttribute** command allows you to create a user attribute within a strategy, and optionally assign a value to it. User attributes may be used to supplement the information that appears on print templates when printing a strategy.

### Syntax:

```
<CreateUserAttribute   Attribute="AttributeName"
                       Value="AttributeValue"
                       Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| AttributeName | The name of the user attribute to create within the strategy. |

| Attribute | Description |
|---|---|
| AttributeValue | The value to assign to the user attribute |
| StrategyName | The name of the strategy in which this user attribute is to be created. This attribute may be the name of an existing template (for example, **$My_Strategy**) or instance.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

**Example:**

Create a user attribute called **Project** on strategy **Strategy_001**:

```
<CreateUserAttribute Attribute="Project " Value="Refinery"
Strategy="Strategy_001"/>
```

# CreateWorkstation Command

The **CreateWorkstation** command allows you to create a workstation instance.

**Note** This command also enables you to create ATS.

**Syntax:**

```
<CreateWorkstation  Template="TemplateName"
                    Workstation="WorkstationName"
                    Node="NodeName"
                    EquipUnitName="EquipUnitName"/>
```

| Attribute | Description |
|---|---|
| TemplateName | The name of the defining template the workstation will be derived from (for example, **$AW70P**). |
| WorkstationName | The name of the workstation to create. This attribute must be the name of an instance; the creation of workstation templates is not supported. |
| NodeName | The optional name of the node to which this workstation is to be assigned. If specified, the attribute must be the name of an existing instance.<br>**Note** The CreateWorkstation command cannot accept both the **EquipUnitName** and the **NodeName** together. It accepts either one of the two. |
| EquipUnitName | The optional name of the equipment unit to which this workstation is to be assigned. If specified, the attribute must be the name of an existing instance; the creation of workstations within equipment unit templates is not supported. |

**Note** **NodeName** should be used while creating ATS.

### Examples:

Create a workstation derived from **$AW70P**, called **STA001**:

```
<CreateWorkstation Template="$AW70P" Workstation="STA001"/>
```

Create a workstation derived from **$AW70P** called **STA001**, and assign it to the equipment unit **EQUIP_UNIT_A**:

```
<CreateWorkstation Template="$AW70P" Workstation="STA001"
EquipUnit="EQUIP_UNIT_A"/>
```

Create a workstation derived from **$ATS** called **ATS001**, and assign it to node **N00005**:

```
<CreateWorkstation Template="$ATS" Workstation="ATS001"
Node="N00005"/>
```

C H A P T E R 8

# Delete Operations

This chapter describes delete operations, which allow you to delete derived Control Software objects.

## Contents



# DeleteBlock Command

The **DeleteBlock** command allows you to delete one or more control blocks within a control strategy instance or a strategy template.

**Syntax:**

```
<DeleteBlock    Block="BlockName"
                Strategy="StrategyName"
                TypeName="TypeName"/>
                Or
<DeleteBlock    Filter="FilterName"/>
```

| Attribute | Description |
|---|---|
| BlockName | The name of the block to be deleted. This attribute may specify either a template or an instance.<br>If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| StrategyName | The name of the strategy from which this block is to be deleted. This attribute may be the name of an existing template (for example, **$My_Strategy**) or instance.<br>This attribute should not be specified if deleting a block template, or if **FilterName** is specified.<br>For a block instance, if this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| TypeName | Specifies the type of block name signified by **BlockName**. Valid values are **ContainedName** or **Tagname**.<br>**TypeName** defaults to **ContainedName** if not specified.<br>**Tagname** signifies the name by which the block is known to the Control Core Services. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block.<br>This attribute is ignored if **FilterName** is specified. |
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be specified.<br>**Note** If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either block templates or strategies.<br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates, or instances of strategies, compounds, controllers and/or equipment units. The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |

**Examples:**

Delete block with tagname of **MY_AIN** from strategy **MyStrategy1**:

```
<DeleteBlock Block="MY_AIN" Strategy="Strategy_001"
TypeName="Tagname"/>
```

Delete the derived template block **$MY_AIN**:

```
<DeleteBlock Block="$MY_AIN"/>
```

Delete block with contained name of **SWCH_2** from the strategy instance **Strategy_002**:

```
<DeleteBlock Block="SWCH_2" Strategy="Strategy_002" />
```

Delete block with tagname of **AOUT_001** from the child strategy **INNER_LOOP** contained within the parent strategy **OUTER_LOOP**:

```
<DeleteBlock Block="AOUT_001"
Strategy="OUTER_LOOP.INNER_LOOP" TypeName="Tagname"/>
```

Delete all blocks based on **$AOUT** from strategy template **$Strategy_001**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="$Strategy_001"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AOUT"/>

<DeleteBlock Filter="Filter1"/>
```

# DeleteCompound Command

The **DeleteCompound** command allows you to delete one or more compound templates or instances.

**Syntax:**

```
<DeleteCompound  Compound="CompoundName"
                 Cascade="CascadeValue"/>
                 Or
<DeleteCompound  Filter="FilterName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, the default value is **No**.<br>If the compound is a template, and **Yes** is specified, all templates and instances derived from that compound template will be deleted as well. If **No** is specified, then the template will not be deleted until all templates and instances derived from it are deleted first.<br>If the compound is an instance, and **Yes** is specified, then all strategies associated with that compound will be deleted as well. If **No** is specified, then the compound instance will not be deleted until the contained strategies are deleted first. |
| FilterName | The name of the filter that will be used to determine what compounds get deleted. This attribute should not be specified if **CompoundName** is given. |

**Examples:**

Delete compound instance **CMPND_001**. Do not delete the compound if it has strategies associated with it.

```
<DeleteCompound Compound="CMPND_001"/>
```

Delete compound instance **CMPND_001**, and delete any strategies that may be associated with it. Note that this will delete all strategies recursively associated with **CMPND_001**:

```
<DeleteCompound Compound="CMPND_001" Cascade="Yes"/>
```

Delete all compounds with names ending in "**003**" assigned to controller **CP0001**. For each compound matching the filter, also delete any strategies that may be associated with it.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%003"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0001"/>

<DeleteCompound Filter="Filter1" Cascade="Yes"/>
```

# DeleteController Command

The **DeleteController** command allows you to delete one or more controller instances.

**Syntax:**

```
<DeleteController  Controller="ControllerName"
                   Cascade="CascadeValue"/>

                   Or

<DeleteController  Filter="FilterName"
                   Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is No.<br>If **Yes** is specified, then all compounds, FCMs, ISCMs and FBMs associated with that controller will be deleted as well. If **No** is specified, then the controller instance will not be deleted until the associated objects are deleted first. |
| FilterName | The name of the filter that will be used to determine what controllers get deleted. This attribute should not be specified if **ControllerName** is given. |

**Examples:**

Delete controller **CP2701**. Do not delete the controller if it has any FCMs, ISCMs or FBMs associated with it, or compounds assigned to it.

```
<DeleteController Controller="CP2701"/>
```

Delete controller instance **ZCP001**, and delete any compounds, FBMs, ISCMs and/or FCMs that may be associated with it:

```
<DeleteController Controller="ZCP001" Cascade="Yes"/>
```

Delete all controllers with letterbugs beginning with the characters **FCP** found in **EquipUnit_1**, and delete any compounds, FBMs, ISCMs and/or FCMs that may be associated with it:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_1"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="FCP%"/>

<DeleteController Filter="Filter1" Cascade="Yes"/>
```

# DeleteDeclaration Command

The **DeleteDeclaration** command allows you to delete a declaration within a control strategy instance, or a strategy template.

**Syntax:**

```
<DeleteDeclaration    Decl="DeclName"
                      Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| DeclName | The name of the declaration to be deleted. |
| StrategyName | The name of the strategy from which this declaration is to be deleted.<br>If this attribute is missing, or blank, the declaration will be deleted from the most recently referenced strategy. If no strategy had been previously referenced, an error will result. |

**Examples:**

Delete declaration **Input1** from strategy **MyStrategy1**:

```
<DeleteDeclaration Decl="Input1" Strategy="MyStrategy1"/>
```

Delete declaration **Output1** from the most recently referenced strategy:

```
<DeleteDeclaration Decl="Output1"/>
```

Delete declaration **My_Input** from the strategy template **$STRATEGY_001**:

```
<DeleteDeclaration Decl="My_Input" Strategy="$STRATEGY_001"/>
```

Delete declaration **My_Output** from the child strategy **INNER_LOOP** contained within the parent strategy **OUTER_LOOP**:

```
<DeleteDeclaration Decl="My_Output"
Strategy="OUTER_LOOP.INNER_LOOP"/>
```

# DeleteDevice Command

The **DeleteDevice** command allows you to delete one or more device instances or templates.

**Syntax:**

```
<DeleteDevice    Device="DeviceName"
                 Cascade="CascadeValue"/>
                 Or
<DeleteDevice    Filter="FilterName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| DeviceName | The name of the device to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed (meaningful only if a device template is being deleted).<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **No** is specified for a device template, then that template will not be deleted if there are any derived templates or instances.<br>If **Yes** is specified, then all child devices associated with that device will be deleted as well. If **No** is specified, then the device will not be deleted until the child devices are deleted first. |
| FilterName | The name of the filter that will be used to determine what devices get deleted. This attribute should not be specified if **DeviceName** is given. |

**Examples:**

Delete device **MY_IAP** which is assigned to FBM **F00001** on controller **CP2701**.

```
<DeleteDevice Device="CP2701.F00001.MY_IAP"/>
```

Delete device template **$IAP020** and delete any devices (templates or instances) that may be derived from it:

```
<DeleteDevice Device="$IAP020" Cascade=”Yes”/>
```

Delete Profibus device instance **MY_DEV**:

```
<DeleteDevice Device="MY_DEV”/>
```

# DeleteEquipUnit Command

The **DeleteEquipUnit** command allows you to delete one or more equipment unit instances.

**Syntax:**

```
<DeleteEquipUnit  EquipUnit="EquipUnitName"
                  Cascade="CascadeValue"/>

                  Or

<DeleteEquipUnit  Filter="FilterName"
                  Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **Yes** is specified, then all objects associated with that equipment will be deleted as well. If **No** is specified, then the equipment unit instance will not be deleted until the associated objects are deleted first. |
| FilterName | The name of the filter that will be used to determine what equipment units get deleted. This attribute should not be specified if **EquipUnitName** is given. |

### Examples:

Delete equipment unit **EQUIP_UNIT1**. Do not delete the equipment unit if it has any other objects associated with it, or compounds assigned to it.

```
<DeleteEquipUnit EquipUnit="EQUIP_UNIT1"/>
```

Delete equipment unit **EQUIP_UNIT2**, which is parented by **EQUIP_UNIT1**. Do not delete the equipment unit if it has any other objects associated with it, or compounds assigned to it.

```
<DeleteEquipUnit EquipUnit="EQUIP_UNIT1.EQUIP_UNIT2"/>
```

Delete equipment unit instance **MY_EQUIP_UNIT**, and delete any objects that may be associated with it:

```
<DeleteEquipUnit EquipUnit="MY_EQUIP_UNIT" Cascade=”Yes”/>
```

Delete all equipment units with names starting with the characters **EQUIP** that are contained within the equipment unit **EQUIP_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Equip_Unit"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="EQUIP_001"/>

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="EQUIP%"/>

<DeleteEquipUnit Filter="Filter1"/>
```

# DeleteFBM Command

The **DeleteFBM** command allows you to delete one or more FBM instances.

### Syntax:

```
<DeleteFBM      FBM="FBMName"
                Cascade="CascadeValue"/>
                Or
<DeleteFBM      Filter="FilterName"
                Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| FBMName | The Tag Name of the FBM to be deleted (not its hierarchical name). This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, the default value is **No**.<br>If **Yes** is specified, then all child devices associated with that FBM will be deleted as well. If **No** is specified, then the FBM instance will not be deleted until the child devices are deleted first. |
| FilterName | The name of the filter that will be used to determine what FBMs get deleted. This attribute should not be specified if **FBMName** is given. |

### Examples:

Delete FBM instance **F00001**. Do not delete the FBM if it has any child devices associated with it.

```
<DeleteFBM FBM="F00001"/>
```

Delete FBM instance **F00001**, and delete any child devices that may be associated with it:

```
<DeleteFBM FBM="F00001" Cascade=”Yes”/>
```

Delete all FBMs assigned to controller **CP0300** with letterbugs beginning with the character **F**, and devices that may be associated with it:

```
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0300"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="F%"/>

<DeleteFBM Filter="Filter1" Cascade="Yes"/>
```

# DeleteFCM Command

The **DeleteFCM** command allows you to delete one or more FCM instances.

**Note** This command also enables you to delete DCM and FBI.

### Syntax:

```
<DeleteFCM      FCM="FCMName"
                Cascade="CascadeValue"/>
                Or
<DeleteFCM      Filter="FilterName"
                Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| FCMName | The name of the FCM to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**.<br>If not specified, default value is **No**.<br>If **Yes** is specified, then all ISCMs and/or FBMs associated with that FCM will be deleted as well. If **No** is specified, then the FCM instance will not be deleted until the ISCMs and FBMs are deleted first. |
| FilterName | The name of the filter that will be used to determine what objects get deleted. This attribute should not be specified if **FCMName** is given. |

### Examples:

Delete FCM **FC0001**. Do not delete the FCM if it has any ISCMs or child FBMs associated with it.

```
<DeleteFCM FCM="FC0001"/>
```

Delete FCM instance **FC0001**, and delete any ISCMs or child FBMs associated with it:

```
<DeleteFCM FCM="FC0001" Cascade="Yes"/>
```

Delete all FCM100s beginning with the characters **FC** currently assigned to controller **CP0003**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$FCM100"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0003"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="FC%"/>

<DeleteFCM Filter="Filter1"/>
```

# DeleteISCM Command

The **DeleteISCM** command allows you to delete one or more ISCM instances.

### Syntax:

```
<DeleteISCM     ISCM="ISCMName"
                Cascade="CascadeValue"/>
                Or
<DeleteISCM     Filter="FilterName"
                Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ISCMName | The name of the ISCM to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is No.<br>If **Yes** is specified, then all FBMs associated with that ISCM will be deleted as well. If **No** is specified, then the ISCM instance will not be deleted. |
| FilterName | The name of the filter that will be used to determine what objects get deleted. This attribute should not be specified if **ISCMName** is given. |

### Examples:

Delete ISCM **ISC0M1**. Do not delete the ISCM if it has FBMs associated with it.

```
<DeleteISCM ISCM="ISC0M1" />
```

Delete ISCM instance **ISC0M1**, and delete any FBMs associated with it:

```
<DeleteISCM ISCM="ISC0M1" Cascade="Yes" >
```

Delete all ISCMs beginning with the characters **IS** currently assigned to controller **CP0003**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ISCM"/>
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="CP0003"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="IS%"/>
<DeleteISCM Filter="Filter1"/>
```

# DeleteNetworkPrinter Command

The **DeleteNetworkPrinter** command allows you to delete one or more network printer instances.

### Syntax:

```
<DeleteNetwork Printer  Printer="PrinterName"/>

                        Or

<DeleteNetwork Printer  Filter="FilterName"/>
```

| Attribute | Description |
|---|---|
| PrinterName | The name of the network printer to be deleted. |
| FilterName | The name of the filter that will be used to determine what printers get deleted. This attribute should not be specified if the **PrinterName** is given. |

### Examples:

Delete network printer PRTNET **PR0001**. Do not delete the network printer if it has any other objects associated with it (which should not happen).

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="PR%"/>

<DeleteNetworkPrinter Printer="PR0001"/>
```

Delete all network printers with letterbugs that begin with **PR** and are currently assigned to equipment unit **EQUIP_UNIT_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$PRTNET"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>

<DeleteNetworkPrinter Filter="Filter1"/>
```

# DeleteNode Command

The **DeleteNode** command allows you to delete one or more node instances.

### Syntax:

```
<DeleteNode   Node="NodeName"
              Cascade="CascadeValue"/>

              Or

<DeleteNode   Filter="FilterName"
              Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, the default value is **No**. If **Yes** is specified, all objects associated with that node will be deleted. If **No** is specified, the node instance will not be deleted until the associated objects are deleted first. |
| FilterName | The name of the filter that will be used to determine what nodes get deleted. This attribute should not be specified if the **NodeName** is given. |

**Examples:**

Delete NODE1. Do not delete NODE1 if it has any other objects associated with it, or compounds assigned to it.

```
<DeleteNode Node="NODE1"/>
```

Delete the node instance called NODE_T and delete all the objects that may be associated with it:

```
<DeleteNode Node="NODE_T" Cascade="Yes"/>
```

Delete all nodes with names starting with the characters **N00** that are contained within the equipment unit **EQUIP_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Node"/>
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="EQUIP_001"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="N00 %"/>
<DeleteNode Filter="Filter1"/>
```

# DeleteObject Command

The **DeleteObject** command allows you to delete one or more object instances or templates.

**Note** Although it is possible to delete derived templates and instances of the Control Software objects (that is, strategies, compounds, controllers, and so forth), it is not advisable to use this command for that purpose. Use the appropriate **Delete** command to actually delete the Control Software objects.

**Syntax:**

```
<DeleteObject      Object="ObjectName"
                   Cascade="CascadeValue"/>
```

```
                    Or
<DeleteObject       Filter="FilterName"
                    Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If the object is a template, and **Yes** is specified, all templates and instances derived from that template will be deleted as well. If **No** is specified, then the template will not be deleted until all templates and instances derived from it are deleted first.<br>If the object is an instance, and **Yes** is specified, then all objects associated with that object will be deleted as well. If **No** is specified, then the object instance will not be deleted until the contained objects are deleted first. |
| FilterName | The name of the filter that will be used to determine what FCMs get deleted. This attribute should not be specified if **ObjectName** is given.<br><br>**Note** Keep in mind that ArchestrA objects **$Integer**, **$Float**, **$Double** and **$StringA2** are actually all derived from **$FieldReference**. |

### Examples:

Delete object **Integer01**. Do not delete the object if it hosts or contains other objects.

```
<DeleteObject Object="Integer01"/>
```

Delete object instance **MyFloat**, and delete any other ArchestrA objects that may be contained or hosted by it:

```
<DeleteObject Object="MyFloat" Cascade="Yes"/>
```

Delete all templates and instances based on **$Boolean** that have the characters **MyBool** in their name. If any of the objects host or contain other objects, delete them as well.

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Boolean"/>
<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="%MyBool%"/>
<DeleteObject Filter="Filter1" Cascade="Yes"/>
```

# DeletePlantUnit Command

The **DeletePlantUnit** command allows you to delete one or more plant unit instances.

### Syntax:

```
<DeletePlantUnit    PlantUnit="PlantUnitName"
                    Cascade="CascadeValue"/>

                    Or

<DeletePlantUnit    Filter="FilterName"
                    Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| PlantUnitName | The name of the plant unit to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **Yes** is specified, then all objects associated with that plant unit will be deleted as well. If **No** is specified, then the plant unit instance will not be deleted until the associated objects are deleted first. |
| FilterName | The name of the filter that will be used to determine what plant units get deleted. This attribute should not be specified if **PlantUnitName** is given. |

### Examples:

Delete plant unit **Plant_Unit1**. Do not delete the plant unit if it has any other objects associated with it.

```
<DeletePlantUnit PlantUnit="Plant_Unit1"/>
```

Delete plant unit **Plant_Unit2**, which is parented by **Plant_Unit1**. Do not delete the plant unit if it has any other objects associated with it.

```
<DeletePlantUnit PlantUnit="Plant_Unit1.Plant_Unit2"/>
```

Delete equipment unit instance **Plant_Unit1**, and delete any objects (for example, devices and other plant units) that may be associated with it:

```
<DeletePlantUnit PlantUnit="Plant_Unit1" Cascade=”Yes”/>
```

Delete all plant units beginning with the characters **Plant**. Also, delete any objects (for example, devices and other plant units) that may be associated with each one:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Plant_Unit"/>

<QueryFilter Filter="Filter1" Condition="NamedLike" Value="Plant%"/>

<DeletePlantUnit Filter="Filter1" Cascade="Yes"/>
```

# DeleteStrategy Command

The **DeleteStrategy** command allows you to delete a control strategy template or instance.

### Syntax:

```
<DeleteStrategy    Strategy="StrategyName"
                   Cascade="CascadeValue"/>

                   Or

<DeleteStrategy    Filter="FilterName"
                   Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy to be deleted. This attribute should not be specified if **FilterName** is specified. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If the strategy is a template, and **Yes** is specified, all templates and instances derived from that strategy template will be deleted as well. If **No** is specified, then the template will not be deleted until all templates and instances derived from it are deleted first.<br>If the strategy is an instance, and **Yes** is specified, then all child strategies associated with that strategy will be deleted as well. If **No** is specified, then the strategy instance will not be deleted until the child strategies are deleted first. |
| FilterName | The name of the filter that will be used to determine what strategies get deleted. This attribute should not be specified if **StrategyName** is given. |

### Examples:

Delete strategy instance **My_Strategy1** that is contained within compound **CMPND_001**. Do not delete the strategy if it has child strategies associated with it.

```
<DeleteStrategy Strategy="CMPND_001.My_Strategy1"/>
```

Delete strategy template **$MY_STRAT**. Also, delete any derived templates and instances that may be derived from that strategy.

```
<DeleteStrategy Strategy="$MY_STRAT" Cascade=”Yes”/>
```

Delete child strategy instance **INNER_LOOP** contained within the strategy **OUTER_LOOP** in compound **CMP_001**. Also, delete any child strategies that may also be associated with **INNER_LOOP**:

```
<DeleteStrategy Strategy="CMP_001.OUTER_LOOP.INNER_LOOP"
Cascade=”Yes”/>
```

Delete all strategies contained in compound **COMPND_002** that begin with **STR**. If any child strategies are encountered, delete those as well.

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_002"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="STR%"/>
<DeleteStrategy Filter="Filter1" Cascade="Yes"/>
```

# DeleteSwitch Command

The **DeleteSwitch** command allows you to delete one or more switch instances.

### Syntax:

```
<DeleteSwitch    Switch="SwitchName"
                 Cascade="CascadeValue"/>
                 Or
<DeleteSwitch    Filter="FilterName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| SwitchName | The name of the switch to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>**Note** For this release of Control Editors software, there are no objects attached to a switch in a containment or hosting sense. Specifying **Yes** or **No** for this command yields the same results. |
| FilterName | The name of the filter that will be used to determine what switches get deleted. This attribute should not be specified if **SwitchName** is given. |

### Examples:

Delete switch **SWCH01**. Do not delete the switch if it has any other objects associated with it (which should not happen).

```
<DeleteSwitch Switch="SWCH01"/>
```

Delete all switches that are contained within **EQUIP_UNIT_001**.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$IA_SWITCH"/>
```

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>

<DeleteSwitch Filter="Filter1"/>
```

# DeleteUserAttribute Command

The **DeleteUserAttribute** command allows you to delete a user attribute within a strategy.

### Syntax:

```
<DeleteUserAttribute   Attribute="AttributeName"
                       Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| AttributeName | The name of the user attribute to delete within the strategy. |
| StrategyName | The name of the strategy in which this user attribute is to be deleted. This attribute may be the name of an existing template (for example, **$My_Strategy**) or instance. If this attribute is missing, or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

### Examples:

Delete a user attribute called **Project** on strategy **Strategy_001**:

```
<DeleteUserAttribute Attribute="Project " Strategy="Strategy_001"/>
```

# DeleteWorkstation Command

The **DeleteWorkstation** command allows you to delete one or more workstation instances.

**Note** This command also enables you to delete ATS.

### Syntax:

```
<DeleteWorkstation   Workstation="WorkstationName"
                     Cascade="CascadeValue"/>

                     Or

<DeleteWorkstation   Filter="FilterName"
                     Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The name of the workstation to be deleted. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade delete is to be performed.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br><br>**Note** For this release of Control Editors, there are no objects attached to a workstation. Specifying **Yes** or **No** for this command provides the same results. |
| FilterName | The name of the filter that will be used to determine what workstations get deleted. This attribute should not be specified if **WorkstationName** is given. |

### Examples:

Delete workstation **STA001**. Do not delete the workstation if it has any other objects associated with it (which should not happen).

```
<DeleteWorkstation Workstation="STA001"/>
```

Delete all workstations with letterbugs that begin with **AW** that are currently assigned to equipment unit **EQUIP_UNIT_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$AW70P"/>
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EQUIP_UNIT_001"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="AW%"/>
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ATS"/>
<DeleteWorkstation Filter="Filter1"/>
```

CHAPTER 9

# Deployment/Undeployment Operations

This chapter describes deployment and undeployment operations. These operations allow you to deploy and undeploy specified Control Software objects, mark Control Software objects as deployed or undeployed (without actually deploying or undeploying them), selectively deploy attributes of the Control Software objects that have been marked as modified in the Galaxy since last deployment, as well as upload specified attributes of deployed Control Software objects.

## Contents

- MarkControllerAsUndeployed Command
- MarkECBAsUndeployed Command
- MarkEquipUnitAsUndeployed Command
- MarkNodeAsUndeployed Command
- MarkStrategyAsUndeployed Command
- UndeployBlock Command
- UndeployCompound Command
- UndeployController Command
- UndeployECB Command
- UndeployEquipUnit Command
- UndeployNode Command
- UndeployObject Command
- UndeployStrategy Command
- UploadCompound Command
- UploadController Command
- UploadStrategy Command

# CheckpointController

The **CheckpointController** command allows you to execute a checkpoint on one or more Control Processors.

### Syntax:

```
<CheckpointController   Controller="ControllerName" />
                        Or
<CheckpointController   Filter="FilterName" />
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the CP for which the checkpoint will be executed.<br>**ControllerName** must be an instance. |
| FilterName | The name of the query filter that will be used to determine what CPs in the Galaxy are to be considered for this operation.<br>This attribute should not be specified if **ControllerName** is given. |

### Examples:

Checkpoint controller **CPX001**:

```
<CheckpointController Controller="CPX001" />
```

Checkpoint all FCP280 controllers:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />
<CheckpointController Filter="Filter1" />
```

# DeployBlock Command

The **DeployBlock** command allows you to deploy a single block.

Although a block is a Control Software object, it is not an ArchestrA object. Therefore, a block needs to be deployed only to the Control Procesor. The **DeployBlock** command performs this task.

If you are deploying a block via the name by which it is known in the controller, then you must specify a value for the TypeName attribute, **TypeName="Tagname"**. If you do not do this, then Direct Access will attempt to find a block in the strategy with a contained name equal to **BlockName**, and deploy that block instead.

If the change tracking feature is enabled (see "EnableCTS Command" on page 134), you must specify a reason for the deployment of the Reason attribute.

### Syntax:

```
<DeployBlock    Strategy="StrategyName"
                Block="BlockName"
                TypeName="TypeName"
                Checkpoint="CheckpointValue"
                Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the block to be deployed.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| BlockName | The name of the block to be deployed. |
| TypeName | Specifies the type of block name signified by **BlockName**. Valid values are **ContainedName** or **Tagname**. **TypeName** defaults to **ContainedName** if not specified. **Tagname** signifies the name by which the block is known to the controller. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. |

| Attribute | Description |
|---|---|
| CheckpointValue | Dictates if checkpoint of the CP that contains block being deployed shall be executed by the end of deployment. CheckpointValue defaults to “Yes” if not specified. If **CheckpointValue** is “Yes”, the checkpoint will be executed. If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required. While “Reason1” is listed in the examples below, this field should contain a rational reason for deployment. |

**Examples:**

Deploy the block with a contained name of **AIN_1_C** in strategy **Strategy_001**, and then execute a CP checkpoint after deployment:

```
<DeployBlock Block="AIN_1_C" Strategy="Strategy_001"
Reason="Reason1" />
```

Deploy the block with a tagname of **AIN_1_T** in strategy **Strategy_001**, but do not execute a CP checkpoint after deployment:

```
<DeployBlock Block="AIN_1_T" Strategy="Strategy_001"
TypeName="Tagname" Checkpoint="No" Reason="Reason1" />
```

# DeployCompound Command

The **DeployCompound** command allows you to deploy one or more compounds.

A compound is a Control Software object, and must therefore be deployed to both the ArchestrA environment and Control Processor. Both of these deployment operations are performed via a single **DeployCompound** command. The deployment states of an object that is deployable to both ArchestrA and the control processor must remain synchronized. If a cascade deployment is performed, the compound, all its ECBs (if any), along with any strategies and blocks assigned to that compound are deployed to the controller. In addition, the compound and strategies are deployed to the ArchestrA environment, because they are ArchestrA objects, while ECBs and blocks are not.

If a cascade deploy is not performed, only the compound is deployed. In this case, the compound will be deployed to both the controller and ArchestrA systems.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for deployment for the Reason attribute.

**Syntax:**

```
<DeployCompound  Compound="CompoundName"
                 Cascade="CascadeValue"
                 Checkpoint="CheckpointValue"
                 Reason="ReasonForDeployment"/>

                 Or

<DeployCompound  Filter="FilterName"
                 Cascade="CascadeValue"
                 Checkpoint="CheckpointValue"
                 Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound being deployed.<br>This attribute must be the name of a compound instance; deployment is not supported for a compound template.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade deployment is to be performed.<br>If **CascadeValue** = **Yes**, then the compound, any ECBs it may contain, and any strategies assigned to that compound will be deployed.<br>If **CascadeValue** = **No**, then only the compound will be deployed. |
| CheckpointValue | Dictates if checkpoint of the CP that contains compound being deployed shall be executed by the end of deployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| FilterName | The name of the filter that will be used to determine what compounds get deployed. This attribute should not be specified if **CompoundName** is given. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for deployment. |

**Examples:**

Deploy compound **COMPND_001**, its ECBs (if any) and all contained strategies and blocks, and then execute a CP checkpoint after deployment:

```
<DeployCompound Compound="COMPND_001" Cascade="Yes"
Reason="Reason1" />
```

Deploy all compounds beginning with the characters **CO** that are assigned to controller **CP0100**, but do not execute a CP checkpoint after deployment:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="CO%" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0100" />

<DeployCompound Filter="Filter1" Cascade="Yes" Checkpoint="No"
Reason="Reason1" />
```

# DeployController Command

The **DeployController** command allows you to deploy one or more controllers.

A controller is a Control Software object, and must therefore be deployed to both ArchestrA and to the Control Processor. Both of these operations are done via a single **DeployController** command. The deployment states of an object that is deployable to both ArchestrA and control processor s must remain synchronized.

If a cascade deploy is performed, the controller, its station and ECB compounds (and ECBs), regular compounds, and all contained strategies and blocks are deployed to the controller. In addition, the controller and affected compounds and strategies are deployed to the ArchestrA environment, because they are ArchestrA objects, while ECBs and blocks are not.

If a cascade deploy is not performed, then the controller is deployed only to the ArchestrA environment. No deployment to the controller will occur.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for the deployment of the Reason attribute.

### Syntax:

```
<DeployController  Controller="ControllerName"
                   Cascade="CascadeValue"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForDeployment"/>

                   Or

<DeployController  Filter="FilterName"
                   Cascade="CascadeValue"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller being deployed. This attribute must be the name of a controller instance.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all compounds and strategies assigned to that controller shall be deployed.<br>If **CascadeValue** = **No**, then only the controller is deployed. |
| CheckpointValue | Dictates if checkpoint of the CP shall be executed by the end of deployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| FilterName | The name of the filter that will be used to determine what controllers get deployed. This attribute should not be specified if **ControllerName** is given. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for deployment. |

### Examples:

Deploy controller **FCP500** and all contained compounds and strategies to the Control Processor, and then execute a CP checkpoint after deployment:

```
<DeployController Controller="FCP500" Cascade="Yes" Reason="Reason1" />
```

Deploy all controllers that are assigned to equipment unit **EquipUnit_001**, but do not execute a CP checkpoint after deployment:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270 />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<DeployController Filter="Filter1" Cascade="Yes" Checkpoint="No"
Reason="Reason1" />
```

# DeployECB Command

The **DeployECB** command allows you to deploy a single ECB.

Although an ECB is a Control Software object, it is not an ArchestrA object. Therefore, an ECB needs to be deployed only to the Control Processor. The **DeployECB** command performs this task.

If the change tracking feature is enabled (see "EnableCTS Command" on page 134), you must specify a reason for deployment for the Reason attribute.

### Syntax:

```
<DeployECB      Compound="CompoundName"
                ECB="ECBName"
                Checkpoint="CheckpointValue"
                Reason="ReasonForDeployment" />
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be deployed.<br>If this attribute is missing or blank, the most recently referenced compound will be used. If no compound had been previously referenced within the XML data, an error will result. |
| ECBName | The name of the ECB to be deployed. |
| CheckpointValue | Dictates if checkpoint of CP that contains ECB being deployed shall be executed by the end of deployment. CheckpointValue defaults to "Yes" if not specified.<br>If **CheckpointValue** is "Yes", the checkpoint will be executed.<br>If **CheckpointValue** is "No", the checkpoint will not be executed. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While "Reason1" is listed in the examples below, this field should contain a rational reason for deployment. |

### Examples:

Deploy the ECB **F00001** in compound **MY_COMPND**, but do not execute a CP checkpoint after deployment:

```
<DeployECB ECB="F00001" Compound="MY_COMPND"
Checkpoint="No" Reason="Reason1" />
```

# DeployEquipUnit Command

The **DeployEquipUnit** command allows you to deploy one or more equipment units.

An equipment unit is a Control Software object, and must therefore be deployed to both the ArchestrA and t. Both of these deployment operations are performed via a single **DeployEquipUnit** command. The deployment states of an object that is deployable to both ArchestrA and the Control Processor must remain synchronized. If a cascade deployment is performed, the Equipment Unit, the Controller, compound, all its ECBs (if any), along with any strategies and blocks assigned to that compound are deployed to the controller. In addition, the equipment unit, compound and strategies are deployed to the ArchestrA environment, because they are ArchestrA objects, while controllers, ECBs and blocks are not.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134) and **Cascade** is “**Yes**”, you must specify a reason for deployment for the Reason attribute.

### Syntax:

```
<DeployEquipUnit   EquipUnit="EquipUnitName"
                   Cascade="CascadeValue"
                   Reason="ReasonForDeployment"/>

                   Or

<DeployEquipUnit   Filter="FilterName"
                   Cascade="CascadeValue"
                   Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit being marked as deployed. This attribute must be the name of an equipment unit instance.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all controllers, compounds, and strategies assigned to that equipment unit shall be marked as deployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the equipment unit is marked as deployed. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled and **CascadeValue** = **Yes**. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for deployment. |
| FilterName | The name of the filter that will be used to determine what Equipment Units get deployed. This attribute should not be specified if **EquipUnitName** is given. |

**Examples:**

Deploy Equipment Unit **Equip_Unit_001** and all contained objects to the controller:

```
<DeployEquipUnit EquipUnit="Equip_Unit_001" Cascade="Yes"
Reason="Reason1" />
```

Deploy all Equipment Units to the controller:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="="$Equip_Unit" />

<DeployEquipUnit Filter="Filter1" Cascade="Yes" Reason="Reason1" />
```

# DeployNode Command

The **DeployNode** command allows you to deploy one or more nodes.

A node is a Control Software object and must be deployed to ArchestrA and the Control Processor. Both these deployment operations are performed via a single **DeployNode** command. The deployment state of an object that is deployable to ArchestrA and Control Processor must remain synchronized. If a cascade deployment is performed, the node, controller, compound, and all of its ECBs (if any) along with any strategies and blocks assigned to that compound are deployed to the controller. In addition, the node, compound, and strategies are deployed to the ArchestrA environment because they are ArchestrA objects, while controllers, ECBs and blocks are not.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134) and cascade is **Yes**, you must specify a reason for deployment for the **Reason** attribute.

**Syntax:**

```
<DeployNode     Node="NodeName"
                Cascade="CascadeValue"
                Reason="ReasonForDeployment"/>

                Or

<DeployNode     Filter="FilterName"
                Cascade="CascadeValue"
                Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node being marked as deployed. This attribute must be the name of a node instance. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all controllers, compounds, and strategies assigned to that node will be marked as deployed.<br>If **CascadeValue** = **No**, then only the node is marked as deployed. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled and **CascadeValue** = **Yes**. Otherwise, it is not required. While **Reason1** is listed in the examples below, this field should contain a rational reason for deployment. |
| FilterName | The name of the filter that will be used to determine what nodes get deployed. This attribute should not be specified if **NodeName** is given. |

**Examples:**

Deploy node **NODE01** and all contained objects to the controller:

```
<DeployNode Node="NODE01" Cascade="Yes" Reason="Reason1"/>
```

Deploy all nodes to the controller:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Node"/>
<DeployNode Filter="Filter1" Cascade="Yes" Reason="Reason1" />
```

# DeployObject Command

The **DeployObject** command allows you to deploy one or more ArchestrA objects.

**Syntax:**

```
<DeployObject    Object="ObjectName"
                 Cascade="CascadeValue"/>
                 Or
<DeployObject    Filter="FilterName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object to deploy. This attribute must be an instance, not a template. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Specifies whether or not a cascade deployment should be performed.<br>If *CascadeValue* = **Yes**, then all objects hosted or contained by this object will be deployed as well. This attribute defaults to **No**. |
| FilterName | The name of the filter that will be used to determine what objects get deployed. This attribute should not be specified if **ObjectName** is given. |

This command should **not** be used to deploy a Control Software object. Do not use the **DeployObject** command to deploy an equipment unit, controller, compound, or strategy. Using this command on one of the aforementioned objects may result in an out-of-synch condition that could potentially prevent the object from being able to be deployed to the controller.

This command has no effect on the control deployment.

### Examples:

Deploy ArchestrA object **Integer01**:

```
<DeployObject Object="Integer01"/>
```

Deploy an ArchestrA platform **MyPlatform** and all app engines, areas, and DI objects below it:

```
<DeployObject Object="MyPlatform" Cascade="Yes"/>
```

Deploy all ArchestrA objects hosted by the object **My_Host**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="My_Host" />

<DeployObject Filter="Filter1" Cascade="Yes"/>
```

# DeployStrategy Command

The **DeployStrategy** command allows you to deploy one or more strategies.

A strategy is a Control Software object, and must therefore be deployed to both ArchestrA and the Control Processor. Both of these operations are done via a single **DeployStrategy** command. The deployment states of an object that is deployable to both ArchestrA and the Control Processor must remain synchronized.

If a cascade deploy is performed, the strategy and its blocks, along with any child strategies (and their blocks) are deployed to the controller. In addition, the strategy (and any affected child strategies) are deployed to the ArchestrA environment, because they are ArchestrA objects, while blocks are not.

If a cascade deploy is not performed, only the strategy is deployed. In this case, the strategy will be deployed to the ArchestrA environment. No deployment to the controller will occur.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for deployment for the Reason attribute.

**Syntax:**

```
<DeployStrategy    Strategy="StrategyName"
                   Cascade="CascadeValue"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForDeployment" />

                   Or

<DeployStrategy    Filter="FilterName"
                   Cascade="CascadeValue"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForDeployment"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being deployed. This attribute must be the name of a strategy instance; deployment is not supported for a strategy template.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all blocks and child strategies (and their blocks) contained by that strategy will be deployed.<br>If **CascadeValue** = **No**, then only the strategy will be deployed. |
| CheckpointValue | Dictates if checkpoint of the CP that contains the strategy being deployed shall be executed by the end of deployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| FilterName | The name of the filter that will be used to determine what strategies get deployed. This attribute should not be specified if **ObjectName** is given. |
| ReasonFor Deployment | Specifies the reason for the deployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for deployment. |

### Examples:

Deploy strategy **Strategy_001** and all contained strategies to the controller, and then execute a CP checkpoint after deployment:

```
<DeployStrategy Strategy="Strategy_001" Cascade="Yes"
Reason="Reason1" />
```

Deploy all strategies that are contained in compound **COMPND_001** to the controller, but do not execute a CP checkpoint after deployment:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_001" />

<DeployStrategy Filter="Filter1" Cascade="Yes" Checkpoint="No"
Reason="Reason1" />
```

# EnableCTS Command

The **EnableCTS** command allows you to enable or disable change tracking on the Galaxy. This command works only when the Galaxy security is on and the user logged in has administrator privileges.

### Syntax:

```
<EnableCTS          Enable="EnableValue"/>
```

| Attribute | Description |
|---|---|
| EnableValue | The value for the enable attribute. Valid value for this attribute is either "**TRUE**" or "**FALSE**" (not case-sensitive). If this attribute is missing or blank or other than the above valid strings, it throws an error. |

### Examples:

Enable change tracking on the Galaxy:

```
<EnableCTS Enable="TRUE" />
```

Disable change tracking on the Galaxy:

```
<EnableCTS Enable="FALSE" />
```

# MarkBlockAsDeployed Command

The **MarkBlockAsDeployed** command allows you to mark a single block as deployed, for synchronization purposes only. **Using this command does not actually deploy the block.**

### Syntax:

```
<MarkBlockAsDeployed  Strategy="StrategyName"
                      Block="BlockName"
                      TypeName="TypeName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the block to be marked as deployed.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| BlockName | The name of the block to be marked as deployed. |
| TypeName | Specifies the type of block name signified by **BlockName**. Valid values are **ContainedName** or **Tagname**. **TypeName** defaults to **ContainedName** if not specified. **Tagname** signifies the name by which the block is known to the controller. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. |

Although a block is a Control Software object, it is not an ArchestrA object. Therefore, a block needs to be marked as deployed only to the controller. The **MarkBlockAsDeployed** command performs this task.

If you are marking a block as deployed via the name by which it is known in the controller, then you must specify a value for the TypeName attribute, **TypeName="Tagname"**. If you do not do this, then Direct Access will attempt to find a block in the strategy with a contained name equal to **BlockName**, and mark that block as deployed instead.

### Examples:

Mark as deployed the block with a contained name of **AIN_1_C** in strategy **Strategy_001**:

```
<MarkBlockAsDeployed Block="AIN_1_C" Strategy="Strategy_001" />
```

Mark as deployed the block with a tagname of **AIN_1_T** in strategy **Strategy_001**:

```
<MarkBlockAsDeployed Block="AIN_1_T" Strategy="Strategy_001"
TypeName="Tagname"/>
```

# MarkCompoundAsDeployed Command

The **MarkCompoundAsDeployed** command allows you to mark one or more compounds as deployed, for synchronization purposes only. **Using this command does not actually deploy the compound.**

### Syntax:

```
<MarkCompoundAsDeployed  Compound="CompoundName"
Cascade="CascadeValue"/>

Or

<MarkCompoundAsDeployed  Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound being marked as deployed. This attribute must be the name of a compound instance; this operation is not supported for a compound template. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed.<br>If **CascadeValue** = **Yes**, then the compound, any ECBs it may contain, and any strategies assigned to that compound will be marked as deployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the compound will be marked as deployed. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. This attribute should not be specified if **CompoundName** is given. |

A compound is a Control Software object, and must therefore be marked as deployed to both ArchestrA and to the controller. Both of these operations are done via a single **MarkCompoundAsDeployed** command. The deployment states of an object that is deployable to both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark compound **COMPND_001** as deployed:

```
<MarkCompoundAsDeployed Compound="COMPND_001"/>
```

Mark as deployed all compounds beginning with the characters **CO** that are assigned to controller **CP0100**:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="CO%" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0100" />

<MarkCompoundAsDeployed Filter="Filter1"/>
```

# MarkControllerAsDeployed Command

The **MarkControllerAsDeployed** command allows you to mark one or more controllers as deployed, for synchronization purposes only. **Using this command does not actually deploy the controller.**

### Syntax:

```
<MarkControllerAsDeployed  Controller="ControllerName"
Cascade="CascadeValue"/>

Or

<MarkControllerAsDeployed  Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller being marked as deployed. This attribute must be the name of a controller instance. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all compounds and strategies assigned to that controller shall be marked as deployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the controller is marked as deployed. |
| FilterName | The name of the filter that will be used to determine what controllers get updated. This attribute should not be specified if **ControllerName** is given. |

A controller is a Control Software object, and must therefore be marked as deployed to both ArchestrA and to the Control Processor. Both of these operations are done via a single **MarkControllerAsDeployed** command. The deployment states of an object that is deployable to both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark controller **FCP500** as deployed:

```
<MarkControllerAsDeployed Controller="FCP500"/>
```

Mark all controllers as deployed that are assigned to equipment unit **EquipUnit_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270 />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkControllerAsDeployed Filter="Filter1”/>
```

# MarkECBAsDeployed Command

The **MarkECBAsDeployed** command allows you to mark a single ECB as deployed, for synchronization purposes only. **Using this command does not actually deploy the ECB.**

### Syntax:

```
<MarkECBAsDeployed    Compound="CompoundName"
                      ECB="ECBName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be marked as deployed.<br>If this attribute is missing or blank, the most recently referenced compound will be used. If no compound had been previously referenced within the XML data, an error will result. |
| ECBName | The name of the ECB to be marked as deployed. |

Although an ECB is a Control Software object, it is not an ArchestrA object. Therefore, an ECB needs to be marked as deployed only to the controller. The **MarkECBAsDeployed** command performs this task.

### Examples:

Mark the ECB **F00001** in compound **MY_COMPND** as deployed:

```
<MarkECBAsDeployed ECB="F00001"
Compound="MY_COMPND" />
```

# MarkEquipUnitAsDeployed Command

The **MarkEquipUnitAsDeployed** command allows you to mark one or more equipment units as deployed, for synchronization purposes only. **Using this command does not actually deploy the equipment unit.**

### Syntax:

```
<MarkEquipUnitAsDeployed    EquipUnit="EquipUnitName"
                            Cascade="CascadeValue"/>
                            Or
<MarkEquipUnitAsDeployed    Filter="FilterName"
                            Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit being marked as deployed. This attribute must be the name of an equipment unit instance.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed.<br>If **CascadeValue** = **Yes**, then all controllers, compounds, and strategies assigned to that equipment unit shall be marked as deployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the equipment unit is marked as deployed. |
| FilterName | The name of the filter that will be used to determine what equipment units get updated. This attribute should not be specified if **EquipUnitName** is given. |

An equipment unit is a Control Software object, and must therefore be marked as deployed to both ArchestrA and the Control Processor. Both of these operations are done via a single *MarkEquipUnitAsDeployed* command. The deployment states of an object that is deployable to both ArchestrA and controller must remain synchronized.

### Examples:

Mark equipment unit **EQUIP1** as deployed:

```
<MarkEquipUnitAsDeployed EquipUnit="EQUIP1" />
```

Mark all equipment units as deployed that are assigned to equipment unit **EquipUnit_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$EquipUnit" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkEquipUnitAsDeployed Filter="Filter1"/>
```

# MarkNodeAsDeployed Command

The **MarkNodeAsDeployed** command allows you to mark one or more nodes as deployed, for synchronization purposes only. **Using this command does not actually deploy the node.**

### Syntax:

```
<MarkNodeAsDeployed        Node="NodeName"
                           Cascade="CascadeValue"/>
```

Or

```
<MarkNodeAsDeployed Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node being marked as deployed. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed.<br>If **CascadeValue** = **Yes**, all controllers, compounds, and strategies assigned to that node will be marked as deployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, only the node is marked as deployed. |
| FilterName | The name of the filter that will be used to determine which node gets updated. This attribute should not be specified if **NodeName** is given. |

A node is a Control Software object and must be marked as deployed to both ArchestrA and Control Processor. Both these operations are done via a single MarkNodeAsDeployed command. The deployment states of an object that is deployable to both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark node **NODE00** as deployed:

```
<MarkNodeAsDeployed Node="NODE00"Cascade="Yes"/>
```

Mark all nodes that are assigned to equipment unit **EquipUnit_001** as deployed:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$NODE" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkNodeAsDeployed Filter="Filter1”/>
```

# MarkStrategyAsDeployed Command

The **MarkStrategyAsDeployed** command allows you to mark one or more strategies as deployed, for synchronization purposes only. **Using this command does not actually deploy the strategy.**

### Syntax:

```
<MarkStrategyAsDeployed Strategy="StrategyName"
Cascade="CascadeValue"/>
```

Or

```
<MarkStrategyAsDeployed  Filter="FilterName"
                         Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being marked as deployed. This attribute must be the name of a strategy instance; this operation is not supported for a strategy template. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of deployment is to be performed. If **CascadeValue** = **Yes**, then all blocks and child strategies (and their blocks) contained by that strategy will be marked as deployed. If **CascadeValue** = **No**, then only the strategy will be marked as deployed. |
| FilterName | The name of the filter that will be used to determine what strategies get updated. This attribute should not be specified if **StrategyName** is given. |

A strategy is a Control Software object, and must therefore be marked as deployed in both ArchestrA and the Control Processor. Both of these operations are done via a single **MarkStrategyAsDeployed** command. The deployment states of an object that is deployable to both ArchestrA and Control Processor must remain synchronized.

### Examples:

Mark strategy **Strategy_001** as deployed:

```
<MarkStrategyAsDeployed Strategy="Strategy_001" />
```

Mark all strategies that are contained in compound **COMPND_001** as deployed:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_001" />

<MarkStrategyAsDeployed Filter="Filter1"/>
```

# MarkBlockAsUndeployed Command

The **MarkBlockAsUndeployed** command allows you to mark a single block as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the block.**

### Syntax:

```
<MarkBlockAsUndeployed    Strategy="StrategyName"
                          Block="BlockName"
                          TypeName="TypeName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the block to be marked as undeployed.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| BlockName | The name of the block to be marked as undeployed. |
| TypeName | Specifies the type of block name signified by **BlockName**. Valid values are **ContainedName** or **Tagname**. **TypeName** defaults to **ContainedName** if not specified. **Tagname** signifies the name by which the block is known to the controller. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. |

Although a block is a Control Software object, it is not an ArchestrA object. Therefore, a block needs to be marked as undeployed only from the controller. The **MarkBlockAsUndeployed** command performs this task.

If you are marking a block as undeployed via the name by which it is known to the controller, then you must specify a value for the TypeName attribute, **TypeName="Tagname"**. If you do not do this, then Direct Access will attempt to find a block in the strategy with a contained name equal to **BlockName**, and mark that block as undeployed instead.

### Examples:

Mark as undeployed the block with a contained name of **AIN_1_C** in strategy **Strategy_001**:

```
<MarkBlockAsUndeployed Block="AIN_1_C"
Strategy="Strategy_001" />
```

Mark as undeployed the block with a tagname of **AIN_1_T** in strategy **Strategy_001**:

```
<MarkBlockAsUndeployed Block="AIN_1_T" Strategy="Strategy_001"
TypeName="Tagname"/>
```

# MarkCompoundAsUndeployed Command

The **MarkCompoundAsUndeployed** command allows you to mark one or more compounds as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the compound.**

### Syntax:

```
<MarkCompoundAsUndeployed Compound="CompoundName"
Cascade="CascadeValue"/>

Or

<MarkCompoundAsUndeployed Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound being marked as undeployed. This attribute must be the name of a compound instance; this operation is not supported for a compound template. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of undeployment is to be performed.<br>If **CascadeValue** = **Yes**, then the compound, any ECBs it may contain, and any strategies assigned to that compound will be marked as undeployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the compound will be marked as undeployed. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. This attribute should not be specified if **CompoundName** is given. |

A compound is a Control Software object, and must therefore be marked as undeployed from both ArchestrA and the Control Processor. Both of these operations are done via a single **MarkCompoundAsUndeployed** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark compound **COMPND_001** as undeployed:

```
<MarkCompoundAsUndeployed Compound="COMPND_001"/>
```

Mark as undeployed all compounds beginning with the characters **CO** that are assigned to controller **CP0100**:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="CO%" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0100" />

<MarkCompoundAsUndeployed Filter="Filter1"/>
```

# MarkControllerAsUndeployed Command

The **MarkControllerAsUndeployed** command allows you to mark one or more controllers as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the controller.**

### Syntax:

```
<MarkControllerAsUndeployed  Controller="ControllerName"
Cascade="CascadeValue"/>

Or

<MarkControllerAsUndeployed  Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller being marked as undeployed. This attribute must be the name of a controller instance. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of undeployment is to be performed.<br>If **CascadeValue** = **Yes**, then all compounds and strategies assigned to that controller shall be marked as undeployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the controller is marked as undeployed. |
| FilterName | The name of the filter that will be used to determine what controllers get updated. This attribute should not be specified if **ControllerName** is given. |

A controller is a Control Software object, and must therefore be marked as undeployed from both ArchestrA and the Control Processor. Both of these operations are done via a single **MarkControllerAsUndeployed** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark controller **FCP500** as undeployed:

```
<MarkControllerAsUndeployed Controller="FCP500"/>
```

Mark all controllers as undeployed that are assigned to equipment unit **EquipUnit_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270" />

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270 />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkControllerAsUndeployed Filter="Filter1”/>
```

# MarkECBAsUndeployed Command

The **MarkECBAsUndeployed** command allows you to mark a single ECB as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the ECB.**

### Syntax:

```
<MarkECBAsUndeployed  Compound="CompoundName"
                      ECB="ECBName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be marked as undeployed.<br>If this attribute is missing or blank, the most recently referenced compound will be used. If no compound had been previously referenced within the XML data, an error will result. |
| ECBName | The name of the ECB to be marked as undeployed. |

Although an ECB is a Control Software object, it is not an ArchestrA object. Therefore, an ECB needs to be marked as undeployed only from the Control Processor. The **MarkECBAsUndeployed** command performs this task.

### Example:

Mark the ECB **F00001** in compound **MY_COMPND** as undeployed:

```
<MarkECBAsUndeployed ECB="F00001"
Compound="MY_COMPND" />
```

# MarkEquipUnitAsUndeployed Command

The **MarkEquipUnitAsUndeployed** command allows you to mark one or more equipment units as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the equipment unit.**

### Syntax:

```
<MarkEquipUnitAsUndeployed    EquipUnit="EquipUnitName"
                              Cascade="CascadeValue"/>
                              Or
<MarkEquipUnitAsUndeployed    Filter="FilterName"
                              Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of the equipment unit being marked as undeployed. This attribute must be the name of an equipment unit instance.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of undeployment is to be performed.<br>If **CascadeValue** = **Yes**, then all controllers, compounds, and strategies assigned to that equipment unit shall be marked as undeployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the equipment unit is marked as undeployed. |
| FilterName | The name of the filter that will be used to determine what equipment units get updated. This attribute should not be specified if **EquipUnitName** is given. |

An equipment unit is a Control Software object, and must therefore be marked as undeployed from both ArchestrA and the Control Processor. Both of these operations are done via a single **MarkEquipUnitAsUndeployed** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark equipment unit **EQUIP1** as undeployed:

```
<MarkEquipUnitAsUndeployed EquipUnit="EQUIP1" />
```

Mark all equipment units as undeployed that are assigned to equipment unit **EquipUnit_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$EquipUnit" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkEquipUnitAsUndeployed Filter="Filter1"/>
```

# MarkNodeAsUndeployed Command

The **MarkNodeAsUndeployed** command allows you to mark one or more nodes as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the node.**

### Syntax:

```
<MarkNodeAsUndeployed      Node="NodeName"
                           Cascade="CascadeValue"/>
```

Or

```
<MarkNodeAsUndeployed Filter="FilterName"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node being marked as undeployed. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of undeployment is to be performed.<br>If **CascadeValue** = **Yes**, then all controllers, compounds, and strategies assigned to that node will be marked as undeployed.<br>If **CascadeValue** is missing or **CascadeValue** = **No**, then only the node will be marked as undeployed. |
| FilterName | The name of the filter that will be used to determine what nodes get updated. This attribute should not be specified if **NodeName** is given. |

A node is a Control Software object and must be marked as undeployed from both ArchestrA and the Control Processor. Both these operations are done via a single MarkNodeAsUnDeployed command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark node **NODE00** as undeployed:

```
<MarkNodeAsDeployed Node="NODE00"Cascade="Yes"/>
```

Mark all nodes that are assigned to equipment unit **EquipUnit_001** as undeployed:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$NODE" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />

<MarkNodeAsDeployed Filter="Filter1" />
```

# MarkStrategyAsUndeployed Command

The **MarkStrategyAsUndeployed** command allows you to mark one or more strategies as undeployed, for synchronization purposes only. **Using this command does not actually undeploy the strategy.**

### Syntax:

```
<MarkStrategyAsUndeployed  Strategy="StrategyName"
                           Cascade="CascadeValue"/>
                           Or
<MarkStrategyAsUndeployed  Filter="FilterName"
                           Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being marked as undeployed. This attribute must be the name of a strategy instance; this operation is not supported for a strategy template. This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade marking of undeployment is to be performed. If **CascadeValue** = **Yes**, then all blocks and child strategies (and their blocks) contained by that strategy will be marked as undeployed. If **CascadeValue** = **No**, then only the strategy will be marked as undeployed. |
| FilterName | The name of the filter that will be used to determine what strategies get updated. This attribute should not be specified if **EquipUnitName** is given. |

A strategy is a Control Software object, and must therefore be marked as undeployed from both ArchestrA and the Control Processor Both of these operations are done via a single **MarkStrategyAsUndeployed** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

### Examples:

Mark strategy **Strategy_001** as undeployed:

```
<MarkStrategyAsUndeployed Strategy="Strategy_001" />
```

Mark all strategies that are contained in compound **COMPND_001** as undeployed:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />
<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_001" />
<MarkStrategyAsUndeployed Filter="Filter1”/>
```

# UndeployBlock Command

The **UndeployBlock** command allows you to undeploy a single block.

Although a block is a Control Software object, it is not an ArchestrA object. Therefore, a block needs to be undeployed only from the controller. The **UndeployBlock** command performs this task.

If you are undeploying a block via the name by which it is known by the controller, then you must specify a value for the **TypeName** attribute, **TypeName="Tagname"**. If you do not do this, then Direct Access will attempt to find a block in the strategy with a contained name equal to **BlockName**, and undeploy that block instead.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for undeployment for the Reason attribute.

### Syntax:

```
<UndeployBlock    Strategy="StrategyName"
                  Block="BlockName"
                  TypeName="TypeName"
                  Checkpoint="CheckpointValue"
                  Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the block to be undeployed.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced within the XML data, an error will result. |
| BlockName | The name of the block to be undeployed. |
| TypeName | Specifies the type of block name signified by **BlockName**. Valid values are **ContainedName** or **Tagname**.<br>**TypeName** defaults to **ContainedName** if not specified.<br>**Tagname** signifies the name by which the block is known to the controller. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. |

| Attribute | Description |
|---|---|
| CheckpointValue | Dictates if checkpoint of the CP that contains block being undeployed shall be executed by the end of the undeployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for undeployment. |

**Examples:**

Undeploy the block with a contained name of **AIN_1_C** in strategy **Strategy_001**, and then execute a CP checkpoint after the undeployment:

```
<UndeployBlock Block="AIN_1_C" Strategy="Strategy_001"
Reason="Reason1" />
```

Undeploy the block with a tagname of **AIN_1_T** in strategy **Strategy_001**, but do not execute a CP checkpoint after the undeployment:

```
<UndeployBlock Block="AIN_1_T" Strategy="Strategy_001"
TypeName="Tagname" Checkpoint="No" Reason="Reason1" />
```

# UndeployCompound Command

The **UndeployCompound** command allows you to undeploy one or more compounds. An undeploy is always cascade.

A compound is a Control Software object, and must therefore be undeployed from both ArchestrA and the Control Processor. Both of these operations are done via a single **UndeployCompound** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

When a compound is undeployed, the compound, all its ECBs (if any), along with any strategies and blocks assigned to that compound are undeployed from the Control Processor. In addition, the compound and strategies are undeployed from ArchestrA, because they are ArchestrA objects, while ECBs and blocks are not.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for undeployment for the Reason attribute.

**Syntax:**

```
<UndeployCompound  Compound="CompoundName"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForUndeployment" />
                   Or
<UndeployCompound  Filter="FilterName"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| CompoundName | The name of a deployed compound instance. This attribute should not be specified if **FilterName** is given. |
| CheckpointValue | Dictates if checkpoint of the CP that contains compound being undeployed shall be executed by the end of the undeployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| FilterName | The name of the filter that will be used to determine what compounds get undeployed. This attribute should not be specified if **CompoundName** is given. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for undeployment. |

**Examples:**

Undeploy compound **COMPND_001** and all contained strategies from the controller, and then execute a CP checkpoint after the undeployment:

```
<UndeployCompound Compound="COMPND_001"/>
```

Undeploy all compounds beginning with the characters **CO** that are assigned to controller **CP0100**, but do not execute a CP checkpoint after the undeployment:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="CO%" />

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0100" />

<UndeployCompound Filter="Filter1" Checkpoint="No"
Reason="Reason1" />
```

# UndeployController Command

The **UndeployController** command allows you to undeploy one or more controllers. An undeploy is always cascade.

A controller is a Control Software object, and must therefore be undeployed from both ArchestrA and the Control Processor. Both of these operations are done via a single **UndeployController** command. The deployment states of an object that is undeployable from both ArchestrA and the Control Processor must remain synchronized.

When a controller is undeployed, the controller, its station and ECB compounds (and ECBs), regular compounds, and all contained strategies and blocks are undeployed from the controller. In addition, the controller and affected compounds and strategies are undeployed from ArchestrA, because they are ArchestrA objects, while ECBs and blocks are not.

The controller is checkpointed at the end of the command.

**Note** If a controller is undeployed (whether cascade or not), it will result in the initialization of the controller. The controller must be manually rebooted following an **UndeployController** command.

If the change tracking feature is enabled (see "EnableCTS Command" on page 134), you must specify a reason for undeployment for the Reason attribute.

### Syntax:

```
<UndeployController    Controller="ControllerName"
                       Reason="ReasonForUndeployment" />
                       Or
<UndeployController    Filter="FilterName"
                       Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| ControllerName | The name of a deployed controller. This attribute should not be specified if **FilterName** is given. |
| FilterName | The name of the filter that will be used to determine what controllers get undeployed. This attribute should not be specified if **ControllerName** is given. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While "Reason1" is listed in the examples below, this field should contain a rational reason for undeployment. |

### Examples:

Undeploy controller **FCP500** and all contained compounds and strategies from the controller, and then execute a CP checkpoint after the undeployment:

```
<UndeployController Controller="FCP500" Reason="Reason1" />
```

Undeploy all controllers that are assigned to equipment unit **EquipUnit_001**, and then execute a checkpoint on all the controllers after the undeployment:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270" />
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270 />
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />
<UndeployController Filter="Filter1" Reason="Reason1" />
```

# UndeployECB Command

The **UndeployECB** command allows you to undeploy a single ECB.

Although an ECB is a Control Software object, it is not an ArchestrA object. Therefore, an ECB needs to be undeployed only from the controller. The **UndeployECB** command performs this task.

**Note** When an ECB is undeployed, its associated FBM will also be marked as undeployed.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for undeployment for the Reason attribute.

### Syntax:

```
<UndeployECB    Compound="CompoundName"
                ECB="ECBName"
                Checkpoint="CheckpointValue"
                Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be undeployed.<br>If this attribute is missing or blank, the most recently referenced compound will be used. If no compound had been previously referenced within the XML data, an error will result. |
| ECBName | The name of the ECB to be undeployed. |

| Attribute | Description |
|---|---|
| CheckpointValue | Dictates if checkpoint of CP that contains the ECB being undeployed shall be executed by the end of the undeployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for undeployment. |

**Example:**

Undeploy the ECB **F00001** in compound **MY_COMPND**, but do not execute a CP checkpoint after the undeployment:

```
<UndeployECB ECB="F00001" Compound="MY_COMPND"
Checkpoint="No" Reason="Reason1" />
```

# UndeployEquipUnit Command

The **UndeployEquipUnit** command allows you to undeploy one or more Equipment Units. An undeploy is always cascade.

An equipment unit is a Control Software object, and must therefore be undeployed from both the ArchestrA and Control Processor. Both of these undeployment operations are performed via a single **UndeployEquipUnit** command. The undeployment states of an object that is deployable to both ArchestrA and Control Processor must remain synchronized. When an equipment unit is undeployed, the equipment unit, controller, compound, all its ECBs (if any), along with any strategies and blocks assigned to that compound are undeployed from the Control Processor. In addition, the equipment unit, compound and strategies are undeployed from ArchestrA, because they are ArchestrA objects, while controllers, ECBs and blocks are not.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for undeployment for the Reason attribute.

**Syntax:**

```
<UndeployEquipUnit    EquipUnit="EquipUnitName"
                      Reason="ReasonForUndeployment" />

                      Or

<UndeployEquipUnit    Filter="FilterName"
                      Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| EquipUnitName | The name of a deployed equipment unit instance.<br>This attribute should not be specified if **FilterName** is given. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While "Reason1" is listed in the examples below, this field should contain a rational reason for undeployment. |
| FilterName | The name of the filter that will be used to determine what equipment units get undeployed. This attribute should not be specified if **EquipUnitName** is given. |

**Examples:**

Undeploy Equipment Unit **Equip_Unit_001** and all contained controllers, compounds and strategies from the controller:

```
<UndeployEquipUnit EquipUnit="Equip_Unit_001" Reason="Reason1"
/>
```

Undeploy all Equipment Units from the controller:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="="$Equip_Unit" />

<UndeployEquipUnit Filter="Filter1" Reason="Reason1" />
```

# UndeployNode Command

The **UndeployNode** command allows you to undeploy one or more nodes. An undeploy is always a cascade.

A node is a Control Software object, and must be undeployed from both ArchestrA and the Control Processor. Both of these undeployment operations are performed via a single **UndeployNode** command. The undeployment state of an object that is deployable to both ArchestrA and Control Processor must remain synchronized. When a node is undeployed, the node, controller, compound, all of its ECBs (if any), along with any strategies and blocks assigned to that compound are undeployed from the Control Processor. In addition, the node, compound and strategies are undeployed from ArchestrA, because they are ArchestrA objects, while controllers, ECBs, and blocks are not.

If the change tracking feature is enabled (see "EnableCTS Command" on page 134), you must specify a reason for undeployment for the **Reason** attribute.

**Syntax:**

```
<UndeployNode    Node="NodeName"
                 Reason="ReasonForUndeployment"/>
                 Or
<UndeployNode    Filter="FilterName"
                 Reason="ReasonForUndeployment"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of a deployed Node instance. This attribute should not be specified if **FilterName** is given. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required. While **Reason1** is listed in the examples below, this field should contain a rational reason for undeployment. |
| FilterName | The name of the filter that will be used to determine what nodes get undeployed. This attribute should not be specified if **NodeName** is given. |

**Examples:**

Undeploy node **NODE01** and all contained controllers, compounds and strategies from the controller:

```
<UndeployNode Node="NODE01" Reason="Reason1"/>
```

Undeploy all nodes from the controller:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Node"/>
<UndeployNode Filter="Filter1" Reason="ReasonForUndeployment"/>
```

# UndeployObject Command

The **UndeployObject** command allows you to undeploy one or more ArchestrA objects. All objects hosted or contained by this object will be undeployed as well.

**Syntax:**

```
<UndeployObject    Object="ObjectName"/>
                   Or
<UndeployObject    Filter="FilterName"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object to undeploy. This attribute must be an instance, not a template.<br>This attribute should not be specified if **FilterName** is given. |
| FilterName | The name of the filter that will be used to determine what objects get undeployed. This attribute should not be specified if **ObjectName** is given. |

This command should **not** be used to undeploy a Control Software object. Do not use the **UndeployObject** command to undeploy an equipment unit, controller, compound, or strategy. Using this command on one of the aforementioned objects may result in an out-of-synch condition that could potentially prevent the object from being able to be undeployed from the controller.

This command has no effect on the controller deployment status.

### Examples:

Undeploy ArchestrA object **Integer01**:

```
<UndeployObject Object="Integer01"/>
```

Undeploy an ArchestrA platform **MyPlatform** and all app engines, areas, and DI objects below it:

```
<UndeployObject Object="MyPlatform"/>
```

Undeploy all ArchestrA objects hosted by the object **My_Host**:

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="My_Host" />

<UndeployObject Filter="Filter1"/>
```

# UndeployStrategy Command

The **UndeployStrategy** command allows you to undeploy one or more strategies from the Control Processor and from ArchestrA.

A strategy is a Control Software object, and must therefore be undeployed from both ArchestrA and Control Processor. Both of these operations are done via a single **UndeployStrategy** command. The deployment states of an object that is undeployable from both ArchestrA and Control Processor must remain synchronized.

When a strategy is undeployed, the strategy and its blocks, along with any child strategies (and their blocks) are undeployed from the controller. In addition, the strategy (and any affected child strategies) are undeployed from ArchestrA, because they are ArchestrA objects, while blocks are not.

If the change tracking feature is enabled (see “EnableCTS Command” on page 134), you must specify a reason for undeployment for the Reason attribute.

### Syntax:

```
<UndeployStrategy  Strategy="StrategyName"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForUndeployment" />
```

Or

```
<UndeployStrategy  Filter="FilterName"
                   Checkpoint="CheckpointValue"
                   Reason="ReasonForUndeployment" />
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being undeployed. This attribute must be the name of a strategy instance; undeployment is not supported for a strategy template.<br>This attribute should not be specified if **FilterName** is given. |
| CheckpointValue | Dictates if checkpoint of the CP that contains the strategy being undeployed shall be executed by the end of the undeployment. CheckpointValue defaults to “Yes” if not specified.<br>If **CheckpointValue** is “Yes”, the checkpoint will be executed.<br>If **CheckpointValue** is “No”, the checkpoint will not be executed. |
| ReasonFor Undeployment | Specifies the reason for the undeployment. It is a required attribute if the change tracking feature is enabled. Otherwise, it is not required.<br>While “Reason1” is listed in the examples below, this field should contain a rational reason for undeployment. |
| FilterName | The name of the filter that will be used to determine what strategies get undeployed. This attribute should not be specified if **StrategyName** is given. |

### Examples:

Undeploy strategy **Strategy_001** and all contained child strategies and blocks from the controller, and then execute a CP checkpoint after the undeployment:

```
<UndeployStrategy Strategy="Strategy_001" Reason="Reason1" />
```

Undeploy all strategies that are contained in compound **COMPND_001** to the controller, but do not execute a CP checkpoint after the undeployment:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_001" />

<UndeployStrategy Filter="Filter1" Checkpoint="No"
Reason="Reason1" />
```

# UploadCompound Command

The **UploadCompound** command allows you to upload one or more compounds from the controller.

### Syntax:

```
<UploadCompound  Compound="CompoundName"
                 Cascade="CascadeValue"/>
                 Or
<UploadCompound  Filter="FilterName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound being uploaded. This attribute must be the name of a compound instance; upload is not supported for a compound template.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade upload is to be performed.<br>If **CascadeValue** = **Yes**, then the compound, any ECBs it may contain, and the blocks contained in any strategies assigned to that compound will be uploaded.<br>If **CascadeValue** = **No**, then only the compound will be uploaded. |
| FilterName | The name of the filter that will be used to determine what compounds get uploaded. This attribute should not be specified if **CompoundName** is given. |

If a cascade upload is performed, the compound, all its ECBs (if any), along with any blocks contained in strategies assigned to that compound are uploaded from the controller.

If a cascade upload is not performed, only the compound is uploaded.

### Examples:

Upload compound **COMPND_001**, its ECBs (if any), and all contained strategies and blocks:

```
<UploadCompound Compound="COMPND_001" Cascade="Yes" />
```

Upload all compounds beginning with the characters **CO** that are assigned to controller **CP0100**:

```
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="CO%" />
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0100" />
<UploadCompound Filter="Filter1" Cascade="Yes"/>
```

# UploadController Command

The **UploadController** command allows you to upload one or more controllers from the controller.

### Syntax:

```
<UploadController  Controller="ControllerName"
                   Cascade="CascadeValue"/>
                   Or
<UploadController  Filter="FilterName"
                   Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller being uploaded. This attribute must be the name of a controller instance; uploading is not supported for a controller template.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade upload is to be performed.<br>If **CascadeValue** = **Yes**, then all compounds and blocks contained in strategies assigned to that controller shall be uploaded.<br>If **CascadeValue** = **No**, then only the controller's ECB and Station compounds will be uploaded. |
| FilterName | The name of the filter that will be used to determine what controllers get uploaded. This attribute should not be specified if **ControllerName** is given. |

If a cascade upload is performed, the controller, its station and ECB compounds (and ECBs), regular compounds, and all blocks in contained strategies are uploaded from the Control Processor.

### Examples:

Upload controller **FCP500**, and all of the compounds, ECBs and blocks contained within strategies associated with that controller from the Control Processor:

```
<UploadController Controller="FCP500" Cascade="Yes" />
```

Upload all controllers (and all compounds, ECBs, and blocks) that are assigned to equipment unit **EquipUnit_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280" />
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP270" />
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$ZCP270 />
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="EquipUnit_001" />
```

```
<UploadCompound Filter="Filter1" Cascade="Yes"/>
```

# UploadStrategy Command

The **UploadStrategy** command allows you to upload one or more strategies from the Control Processor.

**Note** Since strategies do not actually exist within the Control Processor, to upload a strategy actually means that the blocks contained within the strategy are uploaded from the Control Processor.

### Syntax:

```
<UploadStrategy    Strategy="StrategyName"
                   Cascade="CascadeValue"/>
                   Or
<UploadStrategy    Filter="FilterName"
                   Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy being uploaded. This attribute must be the name of a strategy instance; uploading is not supported for a strategy template.<br>This attribute should not be specified if **FilterName** is given. |
| CascadeValue | Dictates whether or not a cascade upload is to be performed.<br>If **CascadeValue** = **Yes**, then all blocks and child strategies (and their blocks) contained by that strategy will be uploaded.<br>If **CascadeValue** = **No**, then only the blocks contained in the strategy will be uploaded. No child strategies (and their blocks) will be processed. |
| FilterName | The name of the filter that will be used to determine what strategies get uploaded. This attribute should not be specified if **StrategyName** is given. |

If a cascade upload is performed, the blocks contained in the strategy as well as the blocks contained in any child strategies are uploaded from the Control Processor.

If a cascade upload is not performed, only the blocks contained in the strategy are uploaded. No child strategies (or their blocks) will be uploaded from Control Processor.

### Examples:

Upload the blocks in strategy **Strategy_001,** and the blocks in all contained strategies to the Control Processor:

```
<UploadStrategy Strategy="Strategy_001" Cascade="Yes" />
```

Upload the blocks contained within any strategies that are contained in compound **COMPND_001** to the Control Processor:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="COMPND_001" />

<UploadStrategy Fitler="Filter1" Cascade="Yes"/>
```

CHAPTER 10

# Import/Export Operations

This chapter describes import operations, which allow you to import specified Control Software files from a user-specified directory into a Galaxy, and export operations, which allow you to export specified Control Software objects from the Galaxy to a Control Software file in a user specified directory.

Direct Access can import the following file formats:

- CSV: Import one or more files created from a Galaxy Dump command. This is supported only for native ArchestrA application objects.
- aaPDF: Import one or more objects generated by Control Editors development.
- aaPKG: Import one or more objects exported from a non-Control Software galaxy in native ArchestrA automation object format.
- aqPKG: Import one or more objects exported in proprietary Control Editors format.

Direct Access can export to the following file formats:

- aqPKG: Export specified objects in proprietary Control Editors format. This export format is supported for all ArchestrA objects as well as for Control Software objects, excluding InTouchView App objects.
- aaPKG: Export specified objects in proprietary ArchestrA object format. This format export is supported for all ArchestrA objects, InTouch View App objects as well as for ArchestrA graphic symbols.
- CSV: Export specified objects as a series of files in CSV format.
- Excel: Export specified objects as a series of worksheets in an Excel spreadsheet.
- SQL Table: Export specified objects as a series of SQL tables.
- XML: Export specified objects as a Direct Access command file.
- IncludeLibrary: Export control logic files of include library objects into a user specified directory. This option is applicable only for SFC and HLBL include library objects.

## Contents

- ImportFile Command
- ImportLibrary Command

# Export Command

The **Export** command allows you to export objects in the Galaxy to an .aqPKG file, .aaPKG file, CSV files, an Excel spreadsheet, SQL tables, or a Direct Access command file. The objects exported will be determined by the query filter specified at the time the **Export** command is issued. When exporting compounds, blocks and ECBs, the objects and attributes that are exported are determined by the current settings (if any) for block and attribute query filters.

**Note** To perform an Excel export, you MUST have Microsoft® Excel 2010 installed, and the Microsoft.Office.Interop.Excel primary interop installed as well.
For legacy Foxboro Control Software (FCS) v3.1 or earlier, the command uses Excel 2007, while for FCS v4.x and Foxboro Evo Control Software v5.0 or later, the command uses Excel 2010.

**Note** Non-Foxboro objects can also be exported. When that occurs, all textual attributes that are found to be contained within the object will be exported.

**Syntax:**

```
<Export          Filter="FilterName"
                 Type="ExportType"
                 FileName ="ExportFileName"
                 Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what objects in the Galaxy are to be exported. This query filter must be created by the **QueryFilter** command.<br>If the query filter results in the same object being found multiple times, it will be exported only once.<br><br>**Note** If the filter is a combination of a **QueryFilter**, **BlockQueryFilter**, and/or **AttributeQueryFilter**, then the filter name used for all three types of filters must be the same, and must equal the name specified by **FilterName**.<br><br>If an object being exported is created from a derived template, then all templates back to (but not including) the base template will be exported as well. This includes blocks within strategies, and ECBs within compounds. |

| Attribute | Description |
|---|---|
| ExportType | The type of Export to perform. If the export type is set to something else, it will be ignored and InFusion will be used as the export type instead. See the following section for valid types of exports on page 166.<br><br>**Note** The valid type of HART and DTM objects export is InFusion (resulting in .aqPKG files). |
| ExportFileName | The name of the export file to be created – valid only for exports of type **InFusion**, **Protected**, **Excel**, or **CommandFile**. This option is ignored for exports of type **CSV** or **SQLTable**, since separate files/tables are created for those types of exports. For Excel exports, both the .xls and .xlsx file formats are supported. Refer to "Opening Direct Access Generated .xls Files with Microsoft Excel 2010" on page 329.<br>If the filename is missing for any of those types of exports, it will default to the following:<br>• **InFusionExport.aqPKG** for a Control Editors export<br>• **InFusionExport.xls** for an Excel export<br>• **InFusionExport.xml** for a CommandFile export<br><br>The file will be created in the directory pointed to by the **ExportOptions** command. If the **ExportOptions** command has not been issued, then the command file will be located in the current directory. |

| Attribute | Description |
|---|---|
| CascadeValue | Specifies whether or not a cascade export is to be performed. This setting is ignored for **ExportType =InFusion** and **ExportType = CommandFile** (**when ExportOption has CommandFileType set to MoveCPConfiguration**), since that export performs a cascade export already.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, the objects that get exported vary depending upon the objects that satisfy the query filter. For example, if an object being exported is a compound template, and cascade = **Yes**, no further objects are exported. However, if the object being exported is a compound instance, then all ECBs and strategies contained within that compound will be exported as well.<br>When **CascadeValue** = **Yes**, the objects that are exported with the indicated "parent" object are:<br>• Compounds – ECBs, strategies, blocks<br>• Controller – software, FCMs, FBMs, and compounds<br>• Devices – none<br>• EquipUnits – nested equipment units, stations, controllers, switches, modules, nodes and devices<br>• Modules – software, child FBMs, and devices<br>• Objects – any hosted or contained objects<br>• PlantUnits – nested plant units and devices<br>• Strategies – strategies, blocks, and ECBs<br>• Workstation – software and peripherals<br>• Nodes – Controllers and ATS |

### Valid ExportType Values

Valid ExportType values are:

- **InFusion** – export objects to an .aqPKG file, which is the normal export that is produced from within the Control Editors. A .aqPKG file can be transferred to another Galaxy, and imported either directly from the Control Editors or via the **ImportFile** command in Direct Access.

  Exporting to the .aaPKG file format is supported for InTouch application objects as well as for ArchestrA objects (ex: boolean, integer, device objects etc.). If you want to export objects that are a combination of Control Software objects and ArchestrA objects, .aqPKG is the only supported file format.

**Note** Export of objects that are a combination of Control Software objects and InTouch application objects is not supported.

The .aqPKG file will be created by the name FileName.aqPKG. If **FileName** is empty or missing, the filename will default to InFusionExport.aqPKG.

The file will be created in the export directory specified by the most recent **ExportOptions** command. If the **ExportOptions** command has not been issued, the .aqPKG or the .aaPKG file will be located in the current directory.

- **Protected** – export protected objects to an .aqPKG or .aaPKG file based on the user selected file format.

**Note** Only Templates can be exported as protected objects. Import of protected Control Software objects is not supported.

- **Excel** – export objects to an Excel spreadsheet. The spreadsheet will be created by the name FileName.xls. If **FileName** is empty or missing, the filename will default to InFusionExport.xls.

  The spreadsheet will be created in the export directory specified by the most recent **ExportOptions** command. If the **ExportOptions** command has not been issued, then the spreadsheet will be located in the current directory.

  The spreadsheet is divided into several worksheets – each worksheet dedicated to one specific type of data. The worksheets and their contents are:

  - Compounds – compounds
  - Declarations – strategy declarations
  - Devices – devices
  - Networks – equipment units and nodes
  - Historization – historization & security
  - Modules – FBMs and FCMs
  - Objects – non-Control Software objects
  - Peripherals – peripherals
  - PlantUnits – plant units
  - Software – software attributes
  - Stations – controllers and workstations
  - Strategies – strategies, blocks, and ECBs
  - Switches – switches

**Note** When performing an Excel export, an **AttributeQueryFilter** should be used to specify that only configurable and non-default values should be exported. If all attributes are exported, there is the danger of the export exceeding the number of columns allowed in an Excel worksheet (256), depending upon the mix and variety of blocks types encountered.

- **CSV –** export objects to CSV files. One CSV file will be created for each type of object encountered. All export files will created in the directory

pointed to by the most recent **ExportOptions** command. If the **ExportOptions** command has not been issued, the CSV files will be located in the current directory. The CSV files are:

- Compounds.csv – compounds
- Declarations.csv – strategy declarations
- Devices.csv – devices
- EquipUnits.csv – equipment units
- Historization.csv – historization & security
- Modules.csv – FBMs and FCMs
- Networks.csv – networks
- Objects.csv – non-Control Software objects
- Peripherals.csv – peripherals
- PlantUnits.csv – plant units
- Software.csv – software attributes
- Stations.csv – controllers and workstations
- Strategies.csv – strategies, blocks, and ECBs
- Switches.csv – switches

**Note** All values that are output to CSV with embedded commas or quotes will be properly "escaped" so that the CSV file can be viewed via Excel.

- **CommandFile** - exports a Direct Access command file. The command exports one of two types of command files: a file that recreates the selected objects, or a file that moves selected controller(s) configuration to another controller. The type of command file is specified by the ExportOption command. The file will be created by the name **FileName.xml**. If **FileName** is empty or missing, the filename will default to **InFusionExport.xml**.

  The file will be created in the directory pointed to by the **ExportOptions** command. If the **ExportOptions** command has not been issued, the command file will be located in the current directory.

**Note** The command supports configurations moved from controllers of type CP40B, CP60, ZCP270, FCP270, and FCP280 to FCP270 and FCP280

- **SQLTable** – export objects to SQL tables. The tables will be located within the SQL Server database called **InFusion_Data**, with table names based on the current Windows login username (usually '**Fox**'):
  - [*Username*]_InFusion_ExportedCompounds – compounds
  - [*Username*]_InFusion_ExportedDeclarations – strategy declarations
  - [*Username*]_InFusion_ExportedDevices – devices
  - [*Username*]_InFusion_ExportedEquipUnits – equipment units
  - [*Username*]_InFusion_ExportedHistorization – historization and security

- [*Username*]_InFusion_ExportedModules – FBMs and FCMs
- [*Username*]_InFusion_ExportedObjects – non-Control Software objects
- [*Username*]_InFusion_ExportedNetworks
- [*Username*]_InFusion_ExportedPeripherals – peripherals
- [*Username*]_InFusion_ExportedPlantUnits – plant units
- [*Username*]_InFusion_ExportedSoftware – software attributes
- [*Username*]_InFusion_ExportedStations – controllers and workstations
- [*Username*]_InFusion_ExportedStrategies – strategies, blocks, and ECBs
- [*Username*]_InFusion_ExportedSwitches – switches

**Note** If errors occur when attempting to export to SQL tables, refer to Chapter 20, “Troubleshooting”.

- **IncludeLibrary** – Export control logic files of include library objects to user specified directory structure, which is the normal **Control Editors -> Export -> Include library** export that is produced from within the Control Editors.

  This option is applicable only for SFC and HLBL include library objects.

  Control logic files can be imported either from the Control Editors or via the **ImportLibrary** command in Direct Access.

  The files will be created with the directory name of the library.

  The library directory will be created in the export directory specified by the most recent **ExportOptions** command. If the **ExportOptions** command has not been issued, the library directory will be located in the current directory.

### Examples:

Export all objects based on **$Strategy** to the Control Editors export file C:\temp\ExportFiles\MyStrategies.aqPKG.

```
<ExportOptions Directory="C:\temp\ExportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>

<Export Filter="Filter1" Type="InFusion" FileName="MyStrategies"/>
```

Export all the template objects as protected based on **$Strategy** to the Control Editors export file C:\temp\ExportFiles\MyProtectedStrategies.aqPKG.

```
<?xml version="1.0" encoding="utf-8"?>

<DirectAccess>

<ExportOptions Directory="C:\Temp\ExportFiles" />

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="$Strategy"/>
```

```
<Export Filter="Filter1" Type="Protected"
FileName="MyProtectedStrategies"/>

</DirectAccess>
```

Export all the template objects as protected based on **$InTouchViewApp** to the Control Editors export file, C:\temp\ExportFiles\InTouchViewAppDerivedTemplate.aapkg.

```
<?xml version="1.0" encoding="utf-8"?>

<DirectAccess>

<ExportOptions Directory=" C:\Temp\ExportFiles" />

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="$InTouchViewApp"/>

<Export Filter="Filter1" Type="Protected"
FileName="InTouchViewAppDerivedTemplate.aaPKG"/>

</DirectAccess>
```

Export all objects hosted or contained within all equipment units to the Excel spreadsheet C:\temp\ExportFiles\InFusionExport.xls. For any compounds, blocks, and ECBs encountered during the export process, export only configurable attributes having non-default values.

```
<ExportOptions Directory="C:\temp\ExportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Equip_Unit"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<Export Filter="Filter1" Type="Excel" Cascade="Yes"/>
```

Export all objects hosted or contained within all equipment units to the appropriate tables within the InFusion_Data database. For any compounds, blocks and ECBs encountered during the export process, export only configurable attributes having non-default values.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Equip_Unit"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<Export Filter="Filter1" Type="SQLTable" Cascade="Yes"/>
```

Export all objects hosted or contained within all equipment units to the appropriate CSV files within the directory C:\temp\ExportFiles. Dump the source code for any sequence or PLB blocks encountered during the export into the **InFusionSupport** directory. Export ALL attributes for all compounds, blocks, and ECBs, regardless of their setting and/or value.

**Note** You probably would NOT want to do this to an Excel spreadsheet, as the number of columns in this type of export could exceed 256 columns, the maximum number of columns allowed in an Excel spreadsheet.

```
<ExportOptions Directory="C:\temp\ExportFiles"
DumpOptions="Source"/>

<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Equip_Unit"/>

<Export Filter="Filter1" Type="CSV" Cascade="Yes"/>
```

Export all blocks derived from **$AIN** contained in any strategy in the Galaxy to the spreadsheet C:\temp\ExportFiles\InFusionExport.xls. Export only the **HSCO1**, **LSCO1**, and **EO1** attributes for each block found

```
<ExportOptions Directory="C:\temp\ExportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="HSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="LSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="EO1"/>

<Export Filter="Filter1" Type="Excel" Cascade="Yes"/>
```

Export all blocks derived from **$AIN** contained in any strategy in the Galaxy to the spreadsheet C:\temp\ExportFiles\MyStrategies.xls. Export only configurable, settable, or data store attributes with non-default values.

```
<ExportOptions Directory="C:\temp\ExportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Settable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="DataStore"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<Export Filter="Filter1" Type="Excel" FileName="MyStrategies”
Cascade="Yes"/>
```

Export control logic files of include library objects to user specified directory structure, which is the normal **Control Editors -> Export -> Include library** export.

```
<ExportOptions Directory="C:\temp\Export"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="$IA_Library_001"/>

<Export Filter="Filter1" Type="IncludeLibrary"/>
```

Export a command file that moves a configuration from FCPB70 to FCPB80. The command file is exported into C:\temp\ExportFiles\MyExportFile.xml file.

```
<ExportOptions Directory="c:\temp\ExportFiles"

CommandFileType="MoveCPConfiguration"

TargetController="FCPB80"/>

<QueryFilter Filter="MyFilter"

Condition="NameEquals"

Value="FCPB70"/>

<Export Filter="MyFilter"

Type="CommandFile"

FileName="MyExportFile"/>
```

# ExportOptions Command

The **ExportOptions** command allows you to specify the directory and other options to use while exporting objects.

### Syntax:

```
<ExportOptions      Directory="ExportDirectory"
                    PrintTemplate="PrintTemplateName"
                    DumpOptions="DumpSourceAndGraphics"
                    LogicBlockColoring="LogicColoring"
                    CommandFileType="ExportCommandFileType"
                    TargetController="TargetControllerName"/>
```

| Attribute | Description |
|---|---|
| ExportDirectory | The name of the directory to which objects are exported.<br>During export, if this directory does not exist, it will be created. If **ExportDirectory** is not specified, the current working directory will be used for storage of export files. |
| PrintTemplateName | The name of the print template to use when exporting a strategy. The ".vsd" extension will automatically be added if not provided within the command.<br><br>**Note** Supplying the appropriate value for PrintTemplateName is important because the Print Template is used during strategy diagram layout to prevent shapes from overlapping page boundaries. This is especially important when exporting a strategy that was generated without a diagram from either Direct Access or Bulk Data.<br><br>PrintTemplateName value may be either the name of a standard print template, or a print template that has been customized by the user. Standard print templates include:<br>• IATemplate_Landscape_Ledger<br>• IATemplate_Landscape_Legal<br>• IATemplate_Landscape_Letter<br>• IATemplate_Portrait_Legal<br>• IATemplate_Portrait_Letter<br>• IATemplate_Portrait_Tabloid<br>If blank or missing, the print template will default to **IATemplate_Portrait_Letter.vsd** |

| Attribute | Description |
|---|---|
| DumpSourceAndGraphics | Indicates whether or not to dump source code for sequence, PLB, and logic blocks, and separate graphics for a strategy, logic blocks, PLB logic and/or SFC logic into the InFusionSupport directory.<br>Valid values are **None**, **Source**, **Graphics**, and **SourceAndGraphics**. If not specified, defaults to **None**.<br>If **DumpSourceAndGraphics** is set to **Source** or **SourceAndGraphics**, the following will be exported:<br>• .s files for HLBL and SFC<br>• .g files for SFC<br>• .k files for SFC<br>• .p files for PLB<br>• .rpn files for Logic blocks<br>If **DumpSourceAndGraphics** is set to **Graphics** or **SourceAndGraphics**, the following will be exported as they are encountered:<br>• .vsd Visio diagram for strategy<br>• .vsd Visio diagram for SFC blocks<br>• .vsd Visio diagram for PLB blocks<br>• .vsd Visio diagram for Logic blocks |

| Attribute | Description |
|---|---|
| ExportCommandFileType | Specifies the type of command file that will be created by Export command. The attribute is used when Export command has ExportType attribute set to CommandFile option.<br>The attribute has two values:<br>**CreateObjects** - Use this to export a command file that recreates the selected objects.<br>**MoveCPConfiguration** --Use this to export a command file that moves controller's configuration to another controller. The target controller is specified by TargetControllerName. The target controller type can be FCP270, or FCP280. The source controller type can be CP40B, CP60, ZCP270, FCP270, or FCP280. The created command files moves compounds, FBMs, and their contained objects (strategies, blocks, devices, ECBs, etc) from one controller to another. The execution orders, IO assignment, ECB containment, etc are preserved during the move. When a configuration is moved from aa ZCP270 or CP60, the command file doesn't move any FCM100, FCM10, DCI10, or FBI10, but assigns their child FBMs directly to the target controller. ExportCommandFileType defaults to CreateObjects if not specified. |

| Attribute | Description |
|---|---|
| TargetControllerName | Specifies the target CP name. The attribute is used when Export command has ExportType attribute set to CommandFile option, and CommandFileType is set to MoveCPConfiguration. The attribute can only specify the tagname of controllers of type FCP270 or FCP280. |
| LogicColoring | Indicates whether or not to fill logic shapes with color when using Logic diagram Appearance Objects and dumping Logic Blocks (CALC, CALCA, LOGIC, and MATH blocks) to a .vsd diagram. Valid values are 'Yes', 'No', and 'None'. If not specified, the default value is 'None'. If 'Yes' is chosen, all logic shapes on a given Logic block are filled with a particular color to help the eye distinguish one block from another on the canvas. If the choice is 'No', all exported logic shapes will be filled with 'white'. If the default 'None' is chosen, all exported logic shapes are colored or white based on whether or not color was chosen when the diagram was last saved. In this case, diagrams will not be updated only to force a coloring choice. LogicColoring is only a valid command argument when the 'DumpOptions' argument is set to either 'Graphics' or 'SourceAndGraphics', and is ignored for all default 'green' Block Appearance Objects |

**Note** An **InFusionSupport** directory will automatically be created in the directory specified by **ExportDirectory**, which will contain a copy of all sequence logic, ladder logic, and strategy diagrams that are encountered while producing the export. What gets copied is determined by the options specified within the **DumpOptions** attribute.

**Note** If a .vsd Visio diagram for a strategy does not exist, (for example, if the strategy was created using DirectAccess or BulkData, and if it has not yet been opened in the Control Editors) the diagram is automatically created during the export.

**Examples:**

Store all the files exported during execution of an **Export** command in the directory **C:\temp\ExportDir**.

```
<ExportOptions Directory="C:\temp\ExportDir" />
```

Generate command file that moves configuration from selected CP(s) to controller FCPB80. Store all the files exported during execution of an Export command in the directory C:\temp\ExportDir.

```
<ExportOptions Directory="C:\temp\ExportDir"
CommandFileType="MoveCPConfiguration"
TargetController="FCPB80" />
```

# ImportOptions Command

The **ImportOptions** command specifies a directory relative to which one or more files will be imported.

**Syntax:**

```
<ImportOptions      Directory="DirectoryName"/>
```

| Attribute | Description |
|---|---|
| DirectoryName | The name of the directory in or below which files to be imported are located. This attribute may be relative from the user's current directory location.<br>If not specified, this attribute defaults to the user's current working directory. |

**Example:**

Specify the directory that contains data to be imported as the **C:\InFusion\bin** directory.

```
<ImportOptions Directory="C:\InFusion\bin"/>
```

# ImportFile Command

The **ImportFile** command specifies the .csv, .aaPDF, .aaPKG, and/or .aqPKG files to be imported.

**Syntax:**

```
<ImportFile      Files="FileSearchPattern"
                 NameConflict="NameConflictSetting"
                 VersionConflict="VersionConflictSetting"
                 NameToAppend="NameToAppendValue"
                 BreakLinkToTemplate="BreakLinkToTemplateFlag"
                 Overwrite="OverwriteSetting"
                 RenameOverwriteConflict=
                     "RenameOverwriteConflictSetting"
                 Override="OverrideSetting"/>
```

| Attribute | Description |
|---|---|
| FileSearchPattern | The name of a file, or files, in .csv, .aaPDF, .aaPKG, and/or .aqPKG format, that are to be imported into the Galaxy. The location of the files is relative to the value specified by the most recent **ImportOptions** command.<br>One wildcard character per **ImportFile** command is allowed. Wildcard characters supported are:<br>• ***** – one or more alphanumeric characters<br>• **?** – exactly one alphanumeric character |
| VersionConflictSetting | Specifies action to take during import when an object version conflict occurs (that is, the object being imported has a version different from that of the same object already found in the Galaxy).<br>Valid values are:<br>• **Migrate**. Migrate the object in the Galaxy to the new version. This would be used primarily for version upgrades as new versions of objects are released for Control Editors patches or quick fixes.<br>• **Skip**. Skip the object being imported, leaving existing objects unchanged.<br><br>If not specified, defaults to **Skip**. |

| Attribute | Description |
|---|---|
| NameConflictSetting | Specifies action to take during import when an object name conflict occurs (that is, the object being imported has the same name as an object already found in the Galaxy).<br><br>**Note** This option applies only to templates and instances derived from different base templates. For example, if object name = XYZ in Galaxy and is derived from **$Strategy**, then a name conflict will occur only if there is an object named XYZ on the import derived from a different base template.<br><br>Valid values are:<br>• **RenameObjectInGalaxy**. Rename the object in the Galaxy to a new name by appending to its current name the string (up to 4 characters) specified by **NameToAppendValue**.<br>• **Skip**. Skip the object being imported when a name conflict is encountered<br><br>If not specified, defaults to **Skip**. |
| NameToAppendValue | A string to use to append to an existing tagname, which will be used to rename objects in the Galaxy when a name conflict occurs during import.<br>May be up to four characters in length.<br>If not specified, defaults to **_old**. |
| BreakLinkToTemplateFlag | This attribute is not applicable to Field devices. |
| OverwriteSetting | Specifies whether or not to overwrite objects in a Galaxy also found in an imported file during an import operation.<br>Valid values are:<br>• **Overwrite**. If an object being imported has a version currently in the Galaxy, this overwrites the object in the Galaxy with the imported version.<br>• **Skip**. If an object being imported has a version currently in the Galaxy, this skips the object from the import operation so the version in the Galaxy is left untouched. |

| Attribute | Description |
|---|---|
| RenameOverwriteConflict Setting | Specifies the action to take when a conflict occurs during an overwrite operation, when an object being imported tries to overwrite another version of this object currently in the Galaxy.<br>Valid values are **True** or **False**. **True** causes the imported object to be renamed if an overwrite conflict occurs. **False** causes the imported operation to skip the import of this object if an overwrite conflict occurs. If not specified, defaults to **False**. |
| OverrideSetting | Specifies the action to take when the Control Software version compatibility check states that the current version is incompatible with the Export version.<br>When an aqPKG file is being imported it tries to override the result of the version compatibility check.<br>Example:<br>Export FCS version 4.0.2, Current FCS version 4.0.1.<br>Valid values are:<br>• **True** causes the import to proceed even if the Export FCS version is incompatible with the Current FCS version.<br>• **False** causes the Import to stop if the Export FCS version is higher than the Current FCS version.<br>If not specified, defaults to False.<br>**Note**<br>1. This option is applicable only for aqPKG format.<br>2. This feature is not supported for major incompatible imports, for example, Import of an aqPKG file created in Foxboro Evo Control Software 5.0 to FCS 4.0.<br>Import will stop even if the OverrideSetting is set to **True**.<br>**Warning:**<br>Overriding incompatible import may corrupt the Galaxy database. You must back up the Galaxy before continuing with this Import Override operation. |
| NeverOverwrite UnprotectedWithProtected | Specifies not to overwrite unprotected objects with protected objects during import operation. Valid values are True or False. |

The mix of files supported by this import process is generated by:

- **csv** – one or more files created from a Galaxy Dump command – NOT SUPPORTED FOR Control Software objects – only ArchestrA objects.
- **aaPDF** – one or more objects generated by the development organization, ready to be imported into the Galaxy
- **aaPKG** – one or more objects exported from a non-I/A Series Galaxy in native ArchestrA automation object format
- **aqPKG** – one or more objects exported from an I/A Series Galaxy.

During import, objects being imported are first evaluated for version conflict, then name conflict.

### Examples:

Import the file **Strategy1.aaPDF** into the Galaxy, skip objects being imported if a version or name conflict is encountered, and do not break the link between any objects being imported and their defining templates:

```
<ImportFile Files="Strategy1.aaPDF”/>
```

Import all .aaPDF files beginning with the letter **B** in the **IA_Blocks** directory, relative to the directory specified by the most recent **ImportOptions** command, skip objects being imported if a version or name conflict is encountered, and do not break the link between any objects being imported and their defining templates:

```
<ImportFile Files="IA_Blocks\B*.aaPDF”/>
```

Import all .aqPKG files found in the directory C:\temp\ImportFiles. If a version conflict is encountered, skip the object being imported. If a name conflict is encountered, rename the object in the Galaxy to its original name with the characters **_ORG** appended to its tagname.

```
<ImportOptions Directory="C:\temp\ImportFiles"/>

<ImportFile Files="*.aqPKG" VersionConflict="Skip"
NameConflict="RenameObjectInGalaxy" NameToAppend="_ORG" />
```

Import operation with “NeverOverwriteUnProtectedWithProtected” set to True ensures that unprotected objects are not overwritten by protected objects.

```
<?xml version="1.0" encoding="utf-8"?>

<DirectAccess>

<ImportOptions Directory="C:\temp\ImportDir" />

<ImportFile File="ImportFile.aqPKG" NameConflict="Skip"
VersionConflict="Migrate" Overwrite="Overwrite"
NeverOverwriteUnProtectedWithProtected="True"/>

</DirectAccess>
```

# ImportLibrary Command

The ImportFile command specifies the include library control logic files to be imported into the specified library. This command is applicable only for SFC and HLBL include library objects.

**Syntax:**

```
<ImportLibrary      Files="FileSearchPattern"
                    Library="LibraryName"
                    LibraryType="LibraryType"
                    Overwrite="OverwriteSetting"
                    Cascade="CascadeSetting"/>
```

| Attribute | Description |
|---|---|
| FileSearchPattern | The name of a file, or files in .h, .s, .ks, .km, and .kx format, that are to be imported into the library. The location of the files is relative to the value specified by the most recent **ImportOptions** command.<br>One wildcard character per **ImportLibrary** command is allowed. Wildcard character supported is:<br>• * – one or more alphanumeric characters |
| LibraryName | Specifies the name of the include library into which the files need to be imported. |
| LibraryType | Specifies the type of the Include library.<br>Valid values are **HLBL Includes** or **SFC Includes**. |

| Attribute | Description |
|---|---|
| OverwriteSetting | Specifies whether or not to overwrite files in a library also found in an imported file during an import operation.<br>Valid values are:<br>• **All**. If the files being imported into the library already have a version currently in the library, this overwrites the library files with the imported files.<br>• **None**. If the files being imported into the library already have a version currently in the library, this skips the files from the import operation so the version in the library is left untouched. |
| CascadeSetting | Specifies whether or not to import the files in the subdirectories of the specified root directory.<br>Valid values are:<br>• **Yes**. Imports all the files including all the subdirectories.<br>• **No**. Imports only the files present in the specified root directory.<br>The default value is **No**. |

### Examples:

Import all files of the specified file search pattern found in the directory **C:\temp\ImportFiles** into the library. If the files being imported into the library already have a version currently in the library, this overwrites the library files with the imported files.

```
<ImportOptions Directory="C:\temp\ImportFiles”/>

<ImportLibrary Files="*.h, *.s, *.ks, *.km Overwrite="All"
Library="$IA_Library_001" LibraryType="SFC Includes" />
```

CHAPTER 11

# Rename Operations

This chapter describes rename operations, which allow you to rename derived Control Software objects.

## Contents



# RenameBlock Command

The **RenameBlock** command allows you to rename a block contained within a strategy. Block templates cannot be renamed in the current release of Direct Access software.

### Syntax:

```
<RenameBlock    Strategy="StrategyName"
                OldName="OldBlockName"
                NewName="NewBlockName"
                TypeName="TypeName"
                TypeOldName="TypeOldName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the block to be renamed.<br>If blank or missing, the strategy will default to the most recently referenced strategy (template or instance). |
| OldBlockName | The block's old name. The name specified must be the contained name or the tagname of the block, as indicated by **TypeOldName** (or **TypeName** if **TypeOldName** is not specified). |
| NewBlockName | The block's new name.<br>If changing the contained name, the new name for the block must be unique within the strategy, or an error will result.<br>If changing the tagname, the new name for the block must be unique within the associated compound, or an error will result during validation. |
| TypeName | The type of name specified by **NewBlockName**. Values may be **Tagname** or **ContainedName**.<br>**TypeName** defaults to **ContainedName** if not specified.<br>**Tagname** signifies the name by which the block is known to Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. |
| TypeOldName | The type of name **OldBlockName** represents. Using **TypeOldName** is necessary only if the type of **NewBlockName** is different from the type of **OldBlockName**. Valid values are **Tagname** or **ContainedName**. If not supplied, this attribute defaults to the same value as **TypeName**. |

### Examples:

Rename the contained name of block **AIN1** to **FLOW** in strategy template **$MY_STRAT1**. Note that the attribute for **TypeName** is not specified in the command, so the program will assume the user is renaming the contained name of the block.

```
<RenameBlock Strategy="$MY_STRAT1" OldName="AIN1"
NewName="FLOW"/>
```

Rename the tagname of block **PIDA1** from **SECONDARY** to **PRIMARY** in the strategy instance **CASCADE_LOOP** in compound **CMPD_001**. The tagname of the block is the name by which the block is known to the Control Processor.

```
<RenameBlock Strategy="CMPD_001.CASCADE_LOOP"
OldName="SECONDARY" NewName="PRIMARY"
TypeName="Tagname"/>
```

# RenameCompound Command

The **RenameCompound** command allows you to rename a compound template or instance.

### Syntax:

```
<RenameCompound   OldName="OldCompoundName"
                  NewName="NewCompoundName"/>
```

| Attribute | Description |
|---|---|
| OldCompoundName | The name of the compound to be renamed. |
| NewCompoundName | The name to which the compound is to be renamed. |

### Examples:

Rename compound instance **COMPND_001** to **COMPND_002**:

```
<RenameCompound OldName="COMPND_001"
NewName="COMPND_002"/>
```

Rename compound template **$MY_COMPND_1** to **$MY_COMPND_2**:

```
<RenameCompound OldName="$MY_COMPND_1"
NewName="$MY_COMPND_2"/>
```

# RenameController Command

The **RenameController** command allows you to rename a controller instance.

### Syntax:

```
<RenameController   OldName="OldControllerName"
                    NewName="NewControllerName"/>
```

| Attribute | Description |
|---|---|
| OldControllerName | The name of the controller to be renamed. |
| NewControllerName | The name to which the controller is to be renamed. |

### Example:

Rename controller instance **CP2801** to **CP2803**:

```
<RenameController OldName="CP2801" NewName="CP2803"/>
```

# RenameDeclaration Command

The **RenameDeclaration** command allows you to rename a declaration contained within a strategy.

### Syntax:

```
<RenameDeclaration   Strategy="StrategyName"
                     OldName="OldDeclName"
                     NewName="NewDeclName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the declaration to be renamed. |
| OldDeclName | The name of the declaration to be renamed. |
| NewDeclName | The name to which the declaration is to be renamed. The new name for the declaration must be unique within the strategy, or an error will result. |

### Examples

Rename declaration **Input_1** to **Primary** in strategy **INNER_LOOP**:

```
<RenameDeclaration Strategy="INNER_LOOP" OldName="Input_1"
NewName="Primary"/>
```

# RenameDevice Command

The **RenameDevice** command allows you to rename a device template or instance (either tagname or contained name).

### Syntax:

```
<RenameDevice     OldName="OldDeviceName"
                  NewName="NewDeviceName"/>
```

| Attribute | Description |
|---|---|
| OldDeviceName | The name of the device to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldDeviceName** and **NewDeviceName** must reflect the same hierarchy – that is, you cannot use the **RenameDevice** command to move a device to a different FBM. |
| NewDeviceName | The new name of the device.<br>If a hierarchical name is provided for **OldDeviceName** and **NewDeviceName**, the device's contained name will be modified – otherwise, the device's tagname will be changed. |

**Examples:**

Rename device instance **DEV001** to **DEV002**:

```
<RenameDevice OldName="DEV001" NewName="DEV002"/>
```

Rename the contained name of device **CP0011.F00023.DEV001** to **DEV002**. The device's tagname will remain unchanged:

```
<RenameDevice OldName="CP0011.F00023.DEV001" NewName="
CP0011.F00023.DEV002"/>
```

Rename device template **$MY_DEV01** to **$MY_DEV02**:

```
<RenameDevice OldName="$MY_DEV01"
NewName="$MY_DEV02"/>
```

# RenameECB Command

The **RenameECB** command allows you to rename an ECB template, or an ECB instance contained within a compound.

Users are not allowed to rename a HART device's ECB. A HART device's ECB name will always be the same as the HART device's tag name.

**Syntax:**

```
<RenameECB      Compound="CompoundName"
                OldName="OldECBName"
                NewName="NewECBName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be renamed.<br>If this attribute is missing or blank, the most recently referenced compound will be used. If no compound had been previously referenced within the XML data, then it is assumed that **OldECBName** refers to a block template. |
| OldECBName | The name of the ECB to be renamed. If it begins with a dollar sign (**$**), will attempt to rename the indicated ECB template. |
| NewECBName | The new name of the ECB. If **OldECBName** refers to a template, then **NewECBName** must refer to a template as well (that is, both must begin with a dollar sign (**$**) if renaming an ECB template). |

### Examples:

Rename ECB **F00001** to **F00002** in compound **COMPND_001**:

```
<RenameECB Compound="COMPND_001" OldName="F00001"
NewName="F00002"/>
```

Rename ECB template **$MY_ECB** to **$MY_ECB2**:

```
<RenameECB OldName="$MY_ECB" NewName="$MY_ECB2"/>
```

# RenameEquipUnit Command

The **RenameEquipUnit** command allows you to rename an equipment unit instance.

### Syntax:

```
<RenameEquipUnit    OldName="OldEquipUnitName"
                    NewName="NewEquipUnitName"/>
```

| Attribute | Description |
|---|---|
| OldEquipUnitName | The name of the equipment unit to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldEquipUnitName** and **NewEquipUnitName** must reflect the same hierarchy – that is, you cannot use the **RenameEquipUnit** command to move an equipment unit to a different parent equipment unit. |
| NewEquipUnitName | The new name of the equipment unit.<br>If a hierarchical name is provided for **OldEquipUnitName** and **NewEquipUnitName**, the equipment unit's contained name will be modified – otherwise, the equipment unit's tagname will be changed. |

**Examples:**

Rename the tagname for equipment unit instance **Equip_Unit_1** to **Equip_Unit_B**:

```
<RenameEquipUnit OldName="Equip_Unit_1"
NewName="Equip_Unit_B"/>
```

Rename the contained name for **EQUIP_UNIT_A.EQUIP_UNIT_B** to **EQUIP_UNIT_C:**

```
<RenameEquipUnit OldName="EQUIP_UNIT_A.EQUIP_UNIT_B"
NewName="EQUIP_UNIT_A.EQUP_UNIT_C"/>
```

# RenameFBM Command

The **RenameFBM** command allows you to rename an FBM instance.

**Syntax:**

```
<RenameFBM      OldName="OldFBMName"
                NewName="NewFBMName"/>
```

| Attribute | Description |
|---|---|
| OldFBMName | The name of the FBM to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldFBMName** and **NewFBMName** must reflect the same hierarchy – that is, you cannot use the **RenameFBM** command to move an FBM to a different controller or FCM. |
| NewFBMName | The new name of the FBM.<br>If a hierarchical name is provided for **OldFBMName** and **NewFBMName**, the FBM's contained name will be modified – otherwise, the FBM's tagname will be changed. |

**Examples:**

Rename the tagname for FBM **F00001** to **F00003**:

```
<RenameFBM OldName="F00001" NewName="F00003"/>
```

Rename the contained name of FBM **ZCP001.FC0023.F00001** to **F00003**:

```
<RenameFBM OldName="ZCP001.FC0023.F00001"
NewName="ZCP001.FC0023.F00003"/>
```

# RenameFCM Command

The **RenameFCM** command allows you to rename an FCM instance.

**Note** This command enables you to rename FCM100, FCM10, DCM10, and FBI10.

**Syntax:**

```
<RenameFCM      OldName="OldFCMName"
                NewName="NewFCMName"/>
```

| Attribute | Description |
|---|---|
| OldFCMName | The name of the FCM to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldFCMName** and **NewFCMName** must reflect the same hierarchy – that is, you cannot use the **RenameFCM** command to move an FCM to a different controller. |
| NewFCMName | The new name of the FCM.<br>If a hierarchical name is provided for **OldFCMName** and **NewFCMName**, the FCM's contained name will be modified – otherwise, the FCM's tagname will be changed. |

### Examples:

Rename the tagname for FCM **FC0001** to **FC0003**:

```
<RenameFCM OldName="FC0001" NewName="FC0003"/>
```

Rename the contained name for FCM **ZCP001.FC0001** to **FC0003**:

```
<RenameFCM OldName="ZCP001.FC0001"
NewName="ZCP001.FC0003"/>
```

# RenameISCM Command

The **RenameISCM** command allows you to rename an ISCM instance.

### Syntax:

```
<RenameISCM      OldName="OldISCMName"
                 NewName="NewISCMName"/>
```

| Attribute | Description |
|---|---|
| OldISCMName | The name of the ISCM to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldISCMName** and **NewISCMName** must reflect the same hierarchy - that is, you cannot use the **RenameISCM** command to move an ISCM to a different controller. |
| NewISCMName | The new name of the ISCM.<br>If a hierarchical name is provided for **OldISCMName** and **NewISCMName**, the ISCM's contained name will be modified - otherwise, the ISCM's tagname will be changed. |

### Examples:

Rename the tagname for ISCM **ISC0M1** to **ISC1M1**:

```
<RenameISCM OldName="ISC0M1" NewName="ISC1M1" />
```

Rename the contained name for ISCM **ZCP001.FC0001.ISC1MA** to **ZCP001.FC0001.ISC2MA**:

```
<RenameISCM OldName="ZCP001.FC0001.ISC1MA"
NewName="ZCP001.FC0001.ISC2MA" />
```

# RenameNetworkPrinter Command

The **RenameNetworkPrinter** command allows you to rename an existing network printer PRTNET instance.

**Syntax:**

```
<RenameNetwork   OldName="OldPrinterName"
Printer          NewName="NewPrinterName"/>
```

| Attribute | Description |
|---|---|
| OldPrinterName | The name of the network printer to be renamed. |
| NewPrinterName | The new name of the network printer. |

**Examples:**

Rename Network Printer PRTNET instance **PR0001** to **PR0002**:

```
<RenameNetworkPrinter OldName="PR0001" NewName="PR0002"/>
```

# RenameNode Command

The **RenameNode** command allows you to rename a node instance.

**Syntax:**

```
<RenameNode      OldName="OldNodeName"
                 NewName="NewNodeName"/>
```

| Attribute | Description |
|---|---|
| OldNodeName | The name of the node to be renamed. |
| NewNodeName | The new name of the node. |

**Examples:**

Rename the tagname for node instance **NODE_1** to **NODE_B**:

```
<RenameNode OldName="NODE_1" NewName="NODE_B"/>
```

# RenameObject Command

The **RenameObject** command allows you to rename an object template or instance.

**Syntax:**

```
<RenameObject    OldName="OldObjectName"
                 NewName="NewObjectName"/>
```

| Attribute | Description |
|---|---|
| OldObjectName | The name of the object to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldObjectName** and **NewObjectName** must reflect the same hierarchy – that is, you cannot use the **RenameObject** command to move an object to a different parent object |
| NewObjectName | The new name of the object.<br>If a hierarchical name is provided for **OldObjectName** and **NewObjectName**, the object's contained name will be modified – otherwise, the object's tagname will be changed. |

**Examples:**

Rename object template **$Test1** to **$Test2**:

```
<RenameObject OldName="$Test1" NewName="$Test2"/>
```

Rename the object **Integer_001** contained within template **$DiffTest** to **Integer_003**:

```
<RenameObject OldName="$DiffTest.Integer_001"
NewName="$DiffTest.Integer_003"/>
```

Rename object instance **DiffTest_002** to **DiffTest_003**:

```
<RenameObject OldName="DiffTest_002" NewName="DiffTest_003"/>
```

Rename the contained name of the object **DiffTest_003.Integer_001** to **Integer_005**:

```
<RenameObject OldName="DiffTest_003.Integer_001"
NewName="DiffTest_003.Integer_005"/>
```

# RenamePlantUnit Command

The **RenamePlantUnit** command allows you to rename a plant unit instance.

**Syntax:**

```
<RenamePlantUnit    OldName="OldPlantUnitName"
                    NewName="NewPlantUnitName"/>
```

| Attribute | Description |
|---|---|
| OldPlantUnitName | The name of the plant unit to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldPlantUnitName** and **NewPlantUnitName** must reflect the same hierarchy – that is, you cannot use the **RenamePlantUnit** command to move a plant unit to a different parent plant unit. |
| NewPlantUnitName | The new name of the plant unit.<br>If a hierarchical name is provided for **OldPlantName** and **NewPlantName**, the plant unit's contained name will be modified – otherwise, the plant unit's tagname will be changed. |

**Examples:**

Rename plant unit instance **Plant_Unit_1** to **Plant_Unit_B**:

```
<RenamePlantUnit OldName="Plant_Unit_1"
NewName="Plant_Unit_B"/>
```

Rename plant unit instance **Plant_Unit_B**, contained by **Plant_Unit_A** to **Plant_Unit_C**:

```
<RenamePlantUnit OldName="Plant_Unit_A.Plant_Unit_B"
NewName="Plant_Unit_A.Plant_Unit_C"/>
```

# RenameStrategy Command

The **RenameStrategy** command allows you to rename a strategy template or instance.

**Syntax:**

```
<RenameStrategy    OldName="OldStrategyName"
                   NewName="NewStrategyName"/>
```

| Attribute | Description |
|---|---|
| OldStrategyName | The name of the strategy to be renamed.<br>If a hierarchical name is supplied, then the contained name will be modified. Both **OldStrategyName** and **NewStrategyName** must reflect the same hierarchy – that is, you cannot use the **RenameStrategy** command to move a strategy to a different compound. |
| NewStrategyName | The new name of the strategy.<br>If a hierarchical name is provided for **OldStrategyName** and **NewStrategyName**, the strategy's contained name will be modified – otherwise, the strategy's tagname will be changed. |

### Examples:

Rename strategy **MY_STRAT1** to **MY_STRAT2**:

```
<RenameStrategy OldName="MY_STRAT1"
NewName="MY_STRAT2"/>
```

Rename the contained name of strategy **MY_STRAT1** in compound **COMPND1** to **MY_STRAT2**:

```
<RenameStrategy OldName="COMPND1.MY_STRAT1"
NewName="COMPND1.MY_STRAT2"/>
```

Rename child strategy **INNER_LOOP** contained within parent strategy template **$REACTOR** to **$CASCADE**:

```
<RenameStrategy OldName="$REACTOR.INNER_LOOP"
NewName="$REACTOR.CASCADE"/>
```

# RenameSwitch Command

The **RenameSwitch** command allows you to rename a switch instance.

### Syntax:

```
<RenameSwitch        OldName="OldSwitchName"
                     NewName="NewSwitchName"/>
```

| Attribute | Description |
|---|---|
| OldSwitchName | The name of the switch to be renamed. |
| NewSwitchName | The name to which the switch is to be renamed. |

### Example

Rename switch instance **SW2801** to **SW2803**:

```
<RenameSwitch OldName="SW2801" NewName="SW2803"/>
```

# RenameUserAttribute Command

The **RenameUserAttribute** command allows you to rename a user attribute contained within a strategy.

### Syntax:

```
<RenameUserAttribute  Strategy="StrategyName"
                      OldName="OldAttrName"
                      NewName="NewAttrName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy containing the user attribute to be renamed. |
| OldAttrName | The name of the user attribute to be renamed. |
| NewAttrName | The name to which the user attribute is to be renamed. The new name for the user attribute must be unique within the strategy, or an error will result. |

### Example

Rename user attribute **Attr_1** to **Primary** in strategy **INNER_LOOP**:

```
<RenameUserAttribute Strategy="INNER_LOOP" OldName="Attr_1"
NewName="Primary"/>
```

# RenameWorkstation Command

The **RenameWorkstation** command allows you to rename a workstation instance.

**Note** This command also enables you to rename ATS.

### Syntax:

```
<RenameWorkstation    OldName="OldWorkstationName"
                      NewName="NewWorkstationName"/>
```

| Attribute | Description |
|---|---|
| OldWorkstationName | The name of the workstation to be renamed. |
| NewWorkstationName | The new name of the workstation. |

### Examples:

Rename workstation instance **STA001** to **STA002**:

```
<RenameWorkstation OldName="STA001" NewName="STA002"/>
```

Rename workstation instance **ATS001** to **ATS002**:

```
<RenameWorkstation OldName="ATS001" NewName="ATS002"/>
```

CHAPTER 12

# Reporting Operations

This chapter describes reporting operations, which allow you to produce a customizable report for query results in a textual or graphical report format.

Direct Access supports the following report formats:

- CSV file
- Excel spreadsheet
- SQL Table
- Visio® document
- PDF document

## Contents



# ProduceReport Command

The **ProduceReport** command allows you to produce a report.

**Syntax:**

```
<Produce Report    ReportName="ReportName"
                   Type="ReportType"
                   FileName="FileName"
                   TableName="TableName"
                   Filter="FilterName"
                   Cascade="CascadeValue"
                   ICCPRTTYPE="PARAMETERS | BLOCKORDER"
                   DeployedFilter="DeployedOnly | All"
                   NamePrefixOption = "CPNAME" | "HOSTANDCPNAME"/>
```

**Note** To produce an Excel report, you MUST have Microsoft Excel 2010 software and Microsoft.Office.Interop.Excel primary interop installed. For legacy Foxboro Control Software (FCS) v3.1 or earlier, the command uses Excel 2007, while for FCS v4.x and Foxboro Evo Control Software v5.0 or later, the command uses Excel 2010.

**Note** Rather than being printed directly, all reports currently are produced as one or more pages in one or more Visio .vsd or .PDF files. You can then open these files and print any or all pages as desired.

| Attribute | Description |
|---|---|
| ReportName | The name of the report to be produced. Currently, the reports that are supported are (report contents are described later in this section):<br>• **BlockDetail** – Reports on all blocks, ECBs, and compounds that are processed as a result of the query filter.<br>• **IOChannels** – Provides an I/O channel report of all FBMs and devices that are processed as a result of the query filter.<br>• **BlockIO** – Provides a list of all I/O blocks that are processed as a result of the query filter.<br>• **StrategyDetail** – Provides a graphical printout of any strategy that is processed as a result of the query filter.<br>• **WhereUsed** – Provides a list of all blocks (and attributes) that reference blocks specified by the **BlockQueryFilter** command.<br>• **UploadReport** – Provides a list of attribute differences between the Galaxy and the running controller for any blocks that are processed as a result of the query filter (on a per controller basis only)<br>• **ObjectList** – Provides a list of all objects (names only) processed as a result of the query filter.<br>• **ICCPRT** – Generates a report in the legacy ICCPRT format. The report lists the parameters with their values for a specified CP, or compound, or lists all compounds for a given CP, or all blocks for a given compound.<br>• **Safety** – Generates a report for one or more Safety Nodes as returned by the query filter. The report lists all alias tags associated with the given Safety Node(s) and related control blocks and parameters as detailed below. |

| Attribute | Description |
|---|---|
| ReportType | The type of report to be produced. Currently, the valid types that are supported are:<br>• **CSV** – Outputs the results to a CSV file named **FileName.csv** in the directory that was specified by the **ReportOptions** command. **CSV** may not be specified in conjunction with the **StrategyDetail** report.<br>**Note** All values that are output to CSV with embedded commas or quotes will be properly "escaped" so that the CSV file can be viewed in Excel software. This is not supported for the ICCPRTTYPE = "BLOCKORDER" report.<br>• **Visio** – Outputs the report results in the directory that was specified by the **ReportOptions** command. The default is a single .vsd file. This report type is only valid when the value of the ReportName attribute is set to **StrategyDetail**. See the ReportOptions attribute **FileFormat** to specify whether a single or multiple .vsd files are generated, and the naming conventions for these report files.<br>• **PDF -** Outputs the report results in the directory that was specified by the **ReportOptions** command. The default is a single .pdf file. This report type is only valid when the value of the ReportName attribute is set to **StrategyDetail**. See the ReportOptions attribute **FileFormat** to specify whether a single or multiple .pdf files are generated, and the naming conventions for these report files.<br>• **Excel** – Outputs the result of the report to a single Excel spreadsheet named **FileName.xls** or **Filename.xlsx** in the directory that was specified by the **ReportOptions** command. **Excel** may not be specified in conjunction with the **StrategyDetail** report.<br>• **SQLTable** – Outputs the result of the report to a SQL table named **TableName** in the database **InFusion_Data**. **SQLTable** may not be specified in conjunction with the **StrategyDetail** report.<br>**Note** SQL table exports are only supported when run from the Galaxy Server.<br>• **TEXT** – Outputs the result of the report to a text file. This is an optional value for the **ICCPRT** report. If TYPE is missing, the default value will be TEXT if FileName is specified without an extension. This is a valid option for the **ICCPRT** report only. |

| Attribute | Description |
|---|---|
| FileName | Name of the file to be created to hold the contents of the report. The type of file created is dependent upon the report type specified by **ReportType**. The file extension may be specified if desired, but if it is, the correct extension type must be used for the report type specified (that is, ".vsd" for Visio format, ".xls" or ".xlsx" for Excel format (refer to "Opening Direct Access Generated .xls Files with Microsoft Excel 2010" on page 329), ".csv" for CSV files, and ".pdf" for PDF files. If left off, the proper file extension will automatically be added to the **FileName**.<br>This attribute should not be specified for **ReportType = SQLTable**.<br><br>**Note** **FileName** can be specified with an extension for the ICCPRT report. Valid extensions are "txt" and "csv" only. For example, if **FileName** is Report.csv, a CSV report is generated; if **FileName** is Report.txt, a TEXT report is generated. **ReportType** is ignored if **FileName** is specified with an extension. The CSV report is supported only for ICCPRTTYPE = PARAMETERS. The CSV report is not supported for the BLOCKORDER report. |
| TableName | The name of the SQL table to be created to hold the contents of the report. **TableName** is valid only for a **ReportType = SQLTable**; it will be ignored for all other report types. |
| NamePrefixOption | There are two valid values for this option, **CPNAME** and **HOSTANDCPNAME**. If NamePrefixOption = **CPNAME**, the NAME field in the report will contain the object name with the controller name. If NamePrefixOption = **HOSTANDCPNAME**, the NAME field in the report will contain the object name along with the controller and host of the controller name. This is a valid option for the **ICCPRT** report only. This option is not supported for the BLOCKORDER report. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what objects in the Galaxy are to be considered for the report. This query filter must be created by the **QueryFilter** command.<br>If the query filter results in the same object being found multiple times, it will be reported only once.<br><br>**Note** If the filter is a combination of a **QueryFilter**, **BlockQueryFilter**, and/or **AttributeQueryFilter**, then the filter name used for all three types of filters must be the same, and must equal the name specified by **FilterName**.<br><br>If an object being considered for the report does not contain any information pertinent to the **ReportType**, then no action will occur on that object. |
| CascadeValue | Specifies whether or not a cascade report is to be produced.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the report will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |
| ICCPRTTYPE | Valid values are **PARAMETERS** and **BLOCKORDER** for this option.<br>If **ICCPRTTYPE = PARAMETERS**, the report will list parameters for compounds and blocks for the specified CP or compound in QueryFilter.<br>If **ICCPRTTYPE = BLOCKORDER**, the report will list all names of blocks or compounds for the specified compound or CP respectively.<br>This is a valid option for the **ICCPRT** report only. |
| DeployedFilter | Valid values are **DeployedOnly** and **All** for this option.<br>If **DeployedFilter** = **DeployedOnly**, only deployed objects will be reported; that is, either fully deployed or modified. The **PARAMETERS** report will list the lastdeployedvalue in this case.<br>If **DeployedFilter = ALL**, all objects (Deployed/Modified/Undeployed) will be reported.<br>In the case of the **PARAMETRS** report, all parameters will report their latest value in FCS Configuration Tools regardless of their deployment status.<br>This is a valid option for the **ICCPRT** report only. |

**Report Contents:**

- **BlockDetail** – for each compound, ECB, or block (template or instance) found within the results of the query filter, the report will contain:
  - Hierarchical name of the block, ECB, or compound
  - The control name for the block, ECB, or compound

- The type of block
- For each attribute that is found to have a value that is different from the default value for that attribute:
  - Attribute Name
  - Attribute Value
  - If connection reference, the appropriate C:B.P reference.

**Note** If **CascadeValue="Yes"**, then all blocks contained within any object that is returned by the query filter will be reported on, including equipment units, controllers, compounds, and strategies.

- **BlockIO** – for each block (instance only) found within the results of the query filter, the report will contain (the converse of the **IOChannels** report):
  - Hierarchical name of the block
  - The block's control name
  - The name of the FBM associated with the block
  - The channel with which the block is associated

**Note** If **CascadeValue="Yes"**, then all blocks contained within any object that is returned by the query filter will be reported on, including equipment units, controllers, compounds, and strategies.

- **IOChannels** – for each FBM or device (instance only) found within the results of the query filter, the report will contain (converse of the **BlockIO** report):
  - Hierarchical name of the FBM or device
  - The type of FBM or device
  - The channel of the FBM or device
  - If the channel is being utilized, then the name of the block utilizing the channel

**Note** If **CascadeValue="Yes"**, then all FBMs or devices contained within any object that is returned by the query filter will be reported on, including equipment units, controllers, and FCMs.

- **StrategyDetail** – for each strategy (template or instance) found within the results of the query filter, the report will contain (Visio or PDF format only):
  - Graphical representation of the strategy diagram, drawn according to the specifications provided by the **FitToPageFlag** and **DrawingScale** attributes of the **ReportOptions** command.
  - For each declaration found within the strategy:
    - Declaration name
    - Declaration type (that is, Input or Output)

- Declaration value (that is, what it is connected to)
- Control reference in C:B.P format
- The block detail report for each block found within the strategy.

**Note** If **CascadeValue="Yes"**, then the report will contain all the child strategies contained within this strategy as well.

- **WhereUsed** – for each block (instance only) found within the results of the block query filter, the report will contain:
  - Source block hierarchical name
  - Source attribute (in C:B.P format)
  - Sink block hierarchical name
  - Sink attribute (in C:B.P format).

**Note** Please note that the **WhereUsed** report can take several minutes to run, depending on how many strategies and blocks there are in the Galaxy.

- **UploadReport** – for each block found within the results of the query filter, the report will contain:
  - Block name
  - Attribute name
  - Attribute value from controller
  - Attribute value from Galaxy.

**Note** Please note that only those attributes that are different will be included in this report. Currently, this report is available only for all blocks in an entire controller.
Also note that **BlockQueryFilter** and **AttributeQueryFilter** are ignored for this report.

- **ObjectList** – for each object found within the results of the query filter, the report will contain:
  - Tagname
  - Hierarchical name
  - Object type.

  If the setting for **CascadeValue="Yes"**, then the query filter will return the following for each object encountered:
  - All objects contained by that object
  - All objects hosted by that object
  - If the object is a compound, all ECBs within that compound
  - If the object is a strategy, all blocks within that strategy.
- **ICCPRT** – There are two types of reports supported by ICCPRT:
  - PARAMETER report – list of all parameters for a given CP or compound

- BLOCKORDER report – list of all compounds for a given CP or all blocks for a given compound

**Note** The **Cascade** option is not supported for this report.

- **Safety** - for each Safety Node (instances only of Tricon, Trident, or Tri_GP) found within the results of the query filter, the report will contain:
  - Safety Node name
  - Tag Alias name
  - Tag Alias number
  - Data Type (BOOL, DINT, or REAL)
  - Accessibility (READ or READWRITE)
  - Tag Description
  - Block Name (of the control block associated with this alias tag)
  - PNT_NO (Point Number of the control block)
  - Deploy Status
  - Block Type (BIN, BOUT, IIN, IOUT, RIN, ROUT, PAKIN, PAKOUT)
  - Controller
  - Compound
  - Strategy
  - FDSI
  - ECB201

**Note** Valid report types for Safety include CSV, Excel, and SqlTable. The **Cascade** option is not supported for this report.

### Examples:

Produce a **BlockDetail** CSV report containing attributes with non-default values for all compounds, blocks, and ECBs found within equipment unit **Equip_Unit_001**. Place a copy of all sequence and/or PLB logic into the **InFusionSupport** directory underneath the specified report directory:

```
<ReportOptions Directory="C:\temp\ReportFiles"
PrintTemplate="IATemplate_Portrait_Letter" DumpOptions="Source"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Equip_Unit_001" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** CSV report for the compound template **$MYCOMPND**:

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="$MYCOMPND" />

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **StrategyDetail** Visio report for all strategies found within equipment unit **Equip_Unit_001,** but not including strategies found in any "child" Equipment Units. Since this involves a graphical output (that is, the strategy), a scale of 50% is to be applied. For each block found within the strategies, report on all attributes having non-default values.

```
<ReportOptions Directory="C:\temp\ReportFiles"
PrintTemplate="IATemplate_Landscape_Letter" Scale="50"
DumpOptions="None"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="Equip_Unit_001" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="Visio" ReportName="StrategyDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a separate StrategyDetail PDF report for each strategy found within any 'child' equipment units contained by **Equip_Unit_001**, but not including strategies found within **Equip_Unit_001** itself. Since this involves a graphical output (that is, the strategy), a scale of 50% is to be applied. For each block found within the strategies, report on all attributes having non-default values.

```
<ReportOptions Directory="C:\temp\ReportFiles"
PrintTemplate="IATemplate_Landscape_Letter" Scale="50"
DumpOptions="None" FileFormat="Multiple"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="Equip_Unit_001" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="PDF" ReportName="StrategyDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **WhereUsed** report on the AIN block **AIN_1** contained within the strategy **S1_001_001**. Store the report contents in an Excel spreadsheet called **MyReport.xls**.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<BlockQueryFilter Filter="Filter1" Condition="NameEquals"
Value="AIN_1" />
```

```
<BlockQueryFilter Filter="Filter1" Condition="ContainedBy"
Value="S1_001_001" />

<ProduceReport Type="Excel" ReportName="WhereUsed"
FileName="MyReport" Filter="Filter1"/>
```

Produce a **BlockDetail** report on attributes having non-default values for all blocks found within strategy **S1_001_001**. Store the results of the report in a SQL table called **MyBlockTable** within the database **InFusion_Data**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="S1_001_001" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="SQLTable" ReportName="BlockDetail"
TableName="MyBlockTable" Filter="Filter1" Cascade="Yes"/>
```

Produce an **ObjectList** report for all strategies found within entire Galaxy. In addition, report block names found within the strategies, and store the results of the report in a CSV file called **MyReport.csv** within the directory **C:\temp\ReportFiles**:

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<ProduceReport Type="CSV" ReportName="ObjectList"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce an **ObjectList** report for all compounds found within entire Galaxy. For each compound located, report all ECBs in the compound, all strategies within the compound, and all blocks within each strategy. Store the results of the report in a CSV file called **MyReport.csv** within the directory **C:\temp\ReportFiles**:

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND" />

<ProduceReport Type="CSV" ReportName="ObjectList"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** that contains all the attributes for all blocks derived from **$PLB** existing in every strategy in the Galaxy. For each PLB block encountered, dump the source (.p file) and the Visio ladder diagram to the **InFusionSupport** directory.

```
<ReportOptions Directory="C:\temp\ReportFiles"
DumpOptions="SourceAndGraphics"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$PLB"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>
```

```
<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce an **IOChannels** report called **C:\temp\ReportFiles\MyReport.csv** for all blocks derived from **$AIN** that exist in strategies contained by the controller **CP0100**.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN" />

<ProduceReport Type="CSV" ReportName="IOChannels"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for all blocks derived from **$AIN** that exist in any strategy in the Galaxy. The report should contain ALL configurable attributes regardless of their current value.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for all blocks derived from **$AIN** that exist in any strategy in the Galaxy. The report should contain only configurable or settable attributes with values different from their default values.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Configurable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="Settable"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NonDefaultsOnly"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for all blocks derived from **$AIN** that exist in any strategy in the Galaxy. The report should contain values only for the attributes **HSCO1**, **LSCO1**, and **EO1**.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="HSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="LSCO1"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="EO1"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for compounds in the Galaxy. The report should contain the values only for **PERIOD** and **PHASE** for each compound reported on.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="PERIOD"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="PHASE"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for ECB compound **CP0100_ECB**. The report should contain the values only for **PERIOD** and **PHASE** for all ECBs found within the compound.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100_ECB" />

<AttributeQueryFilter Filter="Filter1" Cåndition="AttributeName"
Value="PERIOD"/>

<AttributeQueryFilter Filter="Filter1" Cåndition="AttributeName"
Value="PHASE"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce a **BlockDetail** report called **C:\temp\ReportFiles\MyReport.csv** for all compounds in the Galaxy. The report should contain the values only for **PERIOD** and **PHASE** for all compounds, ECBs, and blocks found within the compound.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="PERIOD"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="PHASE"/>

<ProduceReport Type="CSV" ReportName="BlockDetail"
FileName="MyReport" Filter="Filter1" Cascade="Yes"/>
```

Produce an **UploadReport** report called **C:\temp\ReportFiles\MyReport.csv** for all blocks that are currently running within the controller CP0100. Do not dump the logic of any strategies, PLB blocks, or sequence blocks that are found to be within the scope of the report.

```
<ReportOptions Directory="C:\temp\ReportFiles"
PrintTemplate="IATemplate_Portrait_Letter" DumpOptions="None"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100" />

<ProduceReport Type="CSV" ReportName="UploadReport"
FileName="MyReport" Filter="Filter1"/>
```

Produce an **ICCPRT CSV** report of type **PARAMETERS** for blocks in a compound COMPND_001. The report will list all the parameters with their values for the blocks and compound based on the deploy filter.

```
<ReportOptions Directory="D:\DA Report"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMPND_001" />

<ProduceReport Type="CSV" ReportName="ICCPRT"
ICCPRTTYPE="PARAMETERS" FileName="Report" Filter="Filter1"
DeployedFilter="DeployedOnly" NamePrefixOption="CPNAME"/>
```

Produce an **ICCPRT TEXT** report of type **BLOCKORDER** for blocks in a compound COMPND_001. The report will list the blocks in compound COMPND_001 based on the deploy filter.

```
<ReportOptions Directory="D:\DA Report"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMPND_001" />

<ProduceReport Type="TEXT" ReportName="ICCPRT"
ICCPRTTYPE="BLOCKORDER" FileName="Report" Filter="Filter1"
DeployedFilter="DeployedOnly"/>
```

Produce an **ICCPRT CSV** report of type **PARAMETERS** for blocks in compound COMPND_001. The report will list all the parameters with their values for the blocks and compound based on the deploy filter.

```
<ReportOptions Directory="D:\DA Report"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMPND_001" />
```

```
<ProduceReport ReportName="ICCPRT"
ICCPRTTYPE="PARAMETERS" FileName="Report.csv"
Filter="Filter1" DeployedFilter="DeployedOnly"
NamePrefixOption="CPNAME"/>
```

Produce an **ICCPRT TEXT** report of type **PARAMETERS** for blocks in a compound COMPND_001. The report will list all the parameters with their values for the blocks and compound based on the deploy filter. Type is ignored as FileName is specified with an extension.

```
<ReportOptions Directory="D:\DA Report"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMPND_001" />

<ProduceReport Type="CSV" ReportName="ICCPRT"
ICCPRTTYPE="PARAMETERS" FileName="Report.txt"
Filter="Filter1" DeployedFilter="DeployedOnly"
NamePrefixOption="CPNAME"/>
```

Produce a **Safety** report called **C:\temp\ReportFiles\MyReport.csv** for all alias tags and related control blocks associated with Safety Node TRICON_A1.

```
<ReportOptions Directory="C:\temp\ReportFiles"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="TRICON_A1" />

<ProduceReport Type="CSV" ReportName="Safety"
FileName="MyReport" Filter="Filter1" />
```

Produce a **Safety** report for all alias tags and related control blocks associated with each Safety Node in the Galaxy that is derived from **$Tricon**. Store the results of the report in a SQL table called **SafetyTable1** within the database **InFusion_Data**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Tricon" />

<ProduceReport Type="SqlTable" ReportName="Safety"

TableName="SafetyTable1" Filter="Filter1" />
```

# ReportOptions Command

The **ReportOptions** command allows you to specify the reporting options to use while producing reports.

**Note** An **InFusionSupport** directory will automatically be created in the directory specified by **ReportDirectory** which will contain a copy of all sequence logic, ladder logic, and strategy diagrams that are encountered while producing the report. What gets copied is determined by the options specified within the **DumpOptions** attribute.

**Syntax:**

```
<ReportOptions      Directory="ReportDirectory"
                    PrintTemplate="PrintTemplateName"
                    FitToPage="FitToPageFlag"
                    Scale="DrawingScale"
                    DumpOptions="DumpSourceAndGraphics"
                    LogicBlockColoring="LogicColoring"
                    FileFormat="FileFormatType"/>
```

| Attribute | Description |
|---|---|
| ReportDirectory | The name of the directory in which to store the reports.<br>During the reporting process, if this directory does not exist, it will be created. If **ReportDirectory** is not specified, the current working directory (where DirectAccess.exe is located) will be used for storage of report files. |
| PrintTemplateName | The name of the print template to use while generating this particular report. The ".vsd" extension will automatically be added if not provided within the command.<br><br>**Note** **PrintTemplateName** is valid only when producing a Visio or PDF report; it is ignored for all other report types.<br><br>This may be either the name of a standard print template, or a print template that has been customized by the user. Standard print templates include:<br>• IATemplate_Landscape_Ledger<br>• IATemplate_Landscape_Legal<br>• IATemplate_Landscape_Letter<br>• IATemplate_Portrait_Legal<br>• IATemplate_Portrait_Letter<br>• IATemplate_Portrait_Tabloid<br><br>If blank or missing, the print template will default to **IATemplate_Portrait_Letter.vsd**. |

| Attribute | Description |
|---|---|
| FitToPageFlag | If specified, indicates whether or not a graphical printout will be shrunk or expanded to fit on one page, or whether it will be printed at 100% drawing scale.<br>Valid values are **Yes** or **No**. If not specified, default value is **Yes**.<br>If **DrawingScale** is specified, then **FitToPageFlag** must be set to **No**.<br><br>**Note** **FitToPageFlag** is valid only when producing a Visio or PDF report; it is ignored for all other report types.<br><br>**Note** The Visio portion of the report is generated in high resolution. A large complex strategy diagram is readable with the FitToPageFlag set to True. However, if the amount of zoom available while viewing a diagram is still not sufficient, Visio has a feature to provide even greater resolution. Press and hold the **Crtl +Shift + left mouse button** while dragging to capture an area of the diagram, then release the keys/buttons, that area of the diagram becomes magnified. This feature is not available or needed when viewing diagrams in PDF format, which can be zoomed to a much greater resolution than Visio diagrams. |
| DrawingScale | If specified, indicates the scale of the output with which to render the graphical output. For the value specified by **DrawingScale** to be effective, the **FitToPageFlag** must be set to **No**. This attribute should be specified as a whole number representing a percentage of 100. For example, to print something graphically at ½ scale, a value of **Scale="50"** should be specified (50/100 = ½).<br>If both **FitToPageFlag** and **DrawingScale** are not specified, any graphical output will be rendered as though **FitToPageFlag = Yes**.<br><br>**Note** **DrawingScale** is valid only when producing a Visio or PDF report; it is ignored for all other report types. |

| Attribute | Description |
|---|---|
| DumpSourceAndGraphics | Indicates whether or not to dump source code for sequence, PLB, and logic blocks, and separate graphics for a strategy, logic blocks, PLB logic, and/or SFC logic into the **InFusionSupport** directory.<br>Valid values are **None**, **Source**, **Graphics**, and **SourceAndGraphics**. If not specified, **DumpSourceAndGraphics** defaults to **None**.<br><br>If **DumpSourceAndGraphics** is set to **Source** or **SourceAndGraphics**, the following will be reported:<br>• .s files for HLBL and SFC<br>• .g files for SFC<br>• .k files for SFC<br>• .p files for PLB<br>• .rpn files for Logic blocks<br><br>If **DumpSourceAndGraphics** is set to **Graphics** or **SourceAndGraphics**, the following will be reported as they are encountered:<br>• .vsd Visio diagram for strategy<br>• .vsd Visio diagram for SFC blocks<br>• .vsd Visio diagram for PLB blocks<br>• .vsd Visio diagram for Logic blocks |

| Attribute | Description |
|---|---|
| LogicColoring: | Indicates whether or not to fill logic shapes with color when using Logic diagram Appearance Objects and dumping Logic Blocks (CALC, CALCA, LOGIC, and MATH blocks) to a .vsd diagram. Valid values are 'Yes', 'No', and 'None'. If not specified, the default value is 'None'. If 'Yes' is chosen, all logic shapes on a given Logic block are filled with a particular color to help the eye distinguish one block from another on the canvas. If the choice is 'No', all exported logic shapes will be filled with 'white'. If the default 'None' is chosen, all exported logic shapes are colored or white based on whether or not color was chosen when the diagram was last saved. In this case, diagrams will not be updated only to force a coloring choice. LogicColoring is only a valid command argument when the 'DumpOptions' argument is set to either 'Graphics' or 'SourceAndGraphics', and is ignored for all default 'green' Block Appearance Objects. |

| Attribute | Description |
|---|---|
| FileFormatType | If specified, indicates whether or not the graphical output from this command will be written to a single file, or multple files, generating one file for each strategy.<br><br>Valid values are **Single** or **Multiple**. If not specified, the default value is **Single**. All generated files are placed in the directory specified by the ReportOptions **Directory** attribute.<br><br>• If ProduceReport (report) Type=**Visio**, ReportOptions FileFormat=**Single** and ProduceReport FileName=**myReport**, then one Visio report will be generated named **myReport.vsd**.<br><br>• If ProduceReport (report) Type=**PDF**, ReportOptions FileFormat=**Single** and ProduceReport FileName=**myReport**, then one PDF report will be generated named **myReport.pdf**.<br><br>• If ProduceReport (report) Type=**Visio**, ReportOptions FileFormat=**Multiple** and ProduceReport FileName=**myReport**, then a Visio report will be generated for each strategy that will be named **myReport_<StrategyName>.vsd**.<br><br>• If ProduceReport (report) Type=**PDF**, ReportOptions FileFormat=**Multiple** and ProduceReport FileName=**myReport**, then a PDF report will be generated for each strategy that will be named **myReport_<StrategyName>.pdf**. |

**Note** If a .vsd Visio diagram for a strategy does not exist, (for example, if the strategy was created using DirectAccess or BulkData, and if it has not yet been opened in the Control Editors) the diagram is automatically created during the execution of this ProduceReport command.
The diagram is created only if the DumpSourceAndGraphics option is set to **Graphics** or **SourceAndGraphics**.

### Example:

Print reports using the print template **IATemplate_Portrait_Tabloid,** scale graphical output to fit on a single page, and do not dump out any strategy diagrams, PLB logic, or sequence logic.

```
<ReportOptions PrintTemplate="IATemplate_Portrait_Tabloid"
FitToPage="Yes" DumpOptions="None"/>
```

CHAPTER 13

# Repeat Operations

This chapter describes repeat operations, which allow you to perform single or nested loops of commands.

## Contents



# PerformOperation Command

The **PerformOperation** command signals a repeat cycle. The end of the repeat cycle is signified by the presence of an end tag (</PerformOperation>). This command allows you to quickly build up a large configuration, with uniquely named objects, with the minimal number of commands.

Repeat cycles can be nested, that is, you can specify one repeat cycle to occur within another. If done in that manner, then the nested repeat cycle will occur fully for each cycle of the outer repeat cycle. There is no limit to the depth to which repeat cycles can be nested, except for physical memory limitations.

**Syntax:**

```
<PerformOperation    Repeat="RepeatTimes"
                     Start="StartNumber"
                     RememberStart="RememberStart"
                     Pattern="SubstitutionPattern">
                     <Command1… >
                     <Command2… >
                     <Command3… >
                     </PerformOperation>
```

| Attribute | Description |
|---|---|
| RepeatTimes | The number of times commands contained within the repeat cycle are to be performed.<br>All commands found between the start and end tag of the repeat cycle will be repeated **RepeatTimes**.<br>If **RepeatTimes** is not specified, it defaults to 1, executing all commands contained within the repeat cycle once. |
| StartNumber | The number to use to represent the start of the repeat cycle. If **StartNumber** is not specified, it defaults to 1.<br><br>**Note** The cycle number can be any positive integral number – for example, 5, 10268, 322, etc. It does not need to be a single-digit number. The string represented by **SubstitutionPattern** will be replaced by whatever number is represented by the current cycle number. |

| Attribute | Description |
|---|---|
| RememberStart | Indicates whether or not Direct Access should retain the starting number specified by **StartNumber** (applicable only if this is a nested repeat cycle).<br>**Yes** indicates that the start number specified by **StartNumber** will be remembered on subsequent loops, if this **PerformOperation** command is performed within a nested operation.<br>**No** indicates that the start number will not be remembered, and that if the repeat operations contained within this nested operation are executed more than once, that the starting number will be one more than where the previous operation left off. |
| SubstitutionPattern | Indicates a string of one or more characters that will be replaced by the current operation number during a repeat cycle.<br>If not specified, the default substitution pattern will be a single caret (‘^’).<br>During the repeat cycle, all strings containing the substitution pattern will be modified to reflect the cycle number of the current operation. For example, if **StartNumber** = 5, the first time through the repeat cycle, all occurrences of the substitution pattern will be replaced with the number 5. The second time through the cycle, all occurrences of the substitution pattern will be replaced with the number 6, and so on.<br>There is no direct correlation between the number of characters in **SubstitutionPattern**, and the number that is used to replace them. For example, if the substitution pattern is **XYZ**, and the current operation number is 5, then the characters **XYZ** will be replaced with **5** in the current command.<br>If the substitution pattern is used for object names that are required to be a specific length (that is, a letterbug), then care should be taken to ensure that the string resulting from the substitution results in the correct number of characters.<br>If the current command is within a repeat operation that is nested, then string substitution is first performed on the outermost loop, then the next, and so forth, until the innermost loop is reached. The substitution pattern for nested repeat cycles can be the same, but only the outermost matching substitution will actually be performed. |

### Examples:

**Example 1**: Create 100 controllers, each having 16 FBMs. The FBMs under each controller will all be called the same name, so the **RememberStart** option will be used on the **PerformOperation** command for the FBMs.

Notice the use of the pattern substitution for the controller letterbug within the inner repeat cycles.

```
<PerformOperation Repeat="100" Start="100" Pattern="ZZZ">
    <CreateController Template="$FCP270" Controller="AZZZ00"/>
    <PerformOperation Repeat="8" Start="1" RememberStart="Yes"
    Pattern="#">
        <CreateFBM Template="$FBM201" FBM="AZZZ0#"
        Controller="AZZZ00"/>
    </PerformOperation>
    <PerformOperation Repeat="8" Start="11" RememberStart="Yes"
    Pattern="??">
        <CreateFBM Template="$FBM201" FBM="AZZZ??"
        Controller="AZZZ00"/>
    </PerformOperation>
</PerformOperation>
```

**Example 2**: This example depicts the substitution that occurs within inner repeat cycles.

```
<StartTimer Name="TimerTest"/>
<PerformOperation Repeat="2" Start="3" Pattern="kk">
<StartTimer Name="MyTimerkk"/>
<PerformOperation Repeat="2" Start="9" Pattern="zz">
<StartTimer Name="MyTimerkkzz"/>
<PerformOperation Repeat="2" Start="20" Pattern="yy">
<StartTimer Name="MyTimeryykkzzyy"/>
</PerformOperation>
</PerformOperation>
</PerformOperation>
```

Results in the output:

```
Starting timer TimerTest...
Beginning primary loop...
Starting timer MyTimer3...
Repeat nested operations for loop 1...
Starting timer MyTimer39...
Repeat nested operations for loop 2...
Starting timer MyTimer203920...
Starting timer MyTimer213921...
Finished with nested operations
Starting timer MyTimer310...
Repeat nested operations for loop 2...
Starting timer MyTimer2231022...
Starting timer MyTimer2331023...
Finished with nested operations
Finished with nested operations
Starting timer MyTimer4...
Repeat nested operations for loop 1...
```

```
Starting timer MyTimer411...
Repeat nested operations for loop 2...
Starting timer MyTimer2441124...
Starting timer MyTimer2541125...
Finished with nested operations
Starting timer MyTimer412...
Repeat nested operations for loop 2...
Starting timer MyTimer2641226...
Starting timer MyTimer2741227...
```

**Example 3**: Perform a series of commands twice, starting with the cycle number 7. No substitution pattern is specified, so the caret ('^') character will be used.

```
<PerformOperation Repeat="2" Start="7">
    <CreateCompound Template="$COMPND"
    Compound="MyCompound^"/>
    <CreateStrategy Template="$Strategy" Strategy="MyStrategy1^"
    Compound="MyCompound^"/>
    <CreateBlock Template="$AIN" Block="MY_AIN^"
    Strategy="MyStrategy1^"/>
    <CreateBlock Template="$PIDA" Block="MY_PIDA^"
    Strategy="MyStrategy1^"/>
    <CreateBlockAddressCxn Strategy="MyStrategy1^"
    Sink="MY_PIDA^" SinkParm="MEAS"
    SinkValue="MY_AIN^.PNT"/>
    <UpdateBlockAttribute Strategy="MyStrategy1^"
    Block="MY_AIN^" ParmName="HSCO1" ParmValue="105.3"/>
    <CreateStrategy Template="$Strategy" Strategy="MyStrategy2^"/>
    <AssignStrategy Strategy="MyStrategy2^"
    Compound="MyCompound^"/>
</PerformOperation>
```

The commands that are executed the first time through the cycle are:

```
<CreateCompound Template="$COMPND"
Compound="MyCompound^"/>
```

Creates a compound derived from **$COMPND** called **MyCompound7.**

```
<CreateStrategy Template="$Strategy" Strategy="MyStrategy1^"
Compound="MyCompound^"/>
```

Creates a strategy instance derived from **$Strategy** called **MyStrategy17** and assigns it to the compound instance **MyCompound7**.

```
<CreateBlock Template="$AIN" Block="MY_AIN^"
Strategy="MyStrategy1^"/>
```

Creates a block derived from **$AIN** called **MY_AIN7** in strategy instance **MyStrategy17**.

<CreateBlock Template="$PIDA" Block="MY_PIDA^" Strategy="MyStrategy1^"/>

Creates a block derived from **$PIDA** called **MY_PIDA7** in strategy instance **MyStrategy17**.

<CreateBlockAddressCxn Strategy="MyStrategy1^" Sink="MY_PIDA^" SinkParm="MEAS" SinkValue="MY_AIN^.PNT"/>

Creates a connection on the **MY_PIDA7** block within strategy instance **MyStrategy17** by setting the MEAS parameter to **MY_AIN7.PNT**.

<UpdateAttribute Strategy="MyStrategy1^" Block="MY_AIN^" ParmName="HSCO1" ParmValue="105.3"/>

Updates the block **MY_AIN7** on strategy instance **MyStrategy17** by setting its **HSCO1** attribute to **105.3**.

<CreateStrategy Template="$Strategy" Strategy="MyStrategy2^"/>

Creates a strategy instance derived from **$Strategy** called **MyStrategy27**.

<AssignStrategy Strategy="MyStrategy2^" Compound="MyCompound^"/>

Assigns strategy instance **MyStrategy27** to the compound instance **MyCompound7**.

The next time through the repeat cycle, the cycle number will be 8. All of the commands shown above will be reproduced, this time using 8 instead of 7.

CHAPTER 14

# Reset Operations

This chapter describes reset operations, which allow you to reset, or change, the most recently referenced object.

## Contents



# Reset Command

The **Reset** command allows you to reset, or change, the most recently referenced object. Many operations allow you to rely on the most recently referenced object, so that you do not need to re-enter it each time it's referenced. However, the Reset operation allows those most recently referenced objects to be changed, or nulled out.

If named, then the new object must exist in the Galaxy, or an error will result.

**Syntax:**

```
<Reset          Block="BlockName"
                Compound="CompoundName"
                Controller="ControllerName"
                Device="DeviceName"
                EquipUnit="EquipUnitName"
                FBM="FBMName"
                FCM="FCMName"
                ISCM="ISCMName"
                Filter="FilterName"
                Node="NodeName"
                Printer="PrinterName"
                PlantUnit="PlantUnitName"
                Strategy="StrategyName"
                Switch="SwitchName"
                Workstation="WorkstationName"/>
```

| Attribute | Description |
|---|---|
| BlockName | The name of the block template to become the most recently referenced block template. If empty, then no block template will be set as the most recently referenced block template. |
| CompoundName | The name of the compound to become the most recently referenced compound. If empty, then no compound will be set as the most recently referenced compound. |
| ControllerName | The name of the controller or control processor to become the most recently referenced controller. If empty, then no controller will be set as the most recently referenced controller. |
| DeviceName | The name of the device to become the most recently referenced device. If empty, then no device will be set as the most recently referenced device. |
| EquipUnitName | The name of the equipment unit to become the most recently referenced equipment unit. If empty, then no equipment unit will be set as the most recently referenced equipment unit. |
| FBMName | The name of the FBM to become the most recently referenced FBM. If empty, then no FBM will be set as the most recently referenced FBM. |
| FCMName | The name of the FCM to become the most recently referenced FCM. If empty, then no FCM will be set as the most recently referenced FCM. |
| ISCMName | The name of the ISCM to become the most recently referenced ISCM. If empty, then no ISCM will be set as the most recently referenced ISCM. |
| FilterName | The name of the query filter to remove from the list of available filters. If empty, then no query filter will be removed. |
| NodeName | The name of the node to become the most recently referenced node. If empty, then no node will be set as the most recently referenced node. |
| PrinterName | The name of the network printer (PRTNET) to become the most recently referenced network printer. If empty, no network printer will be set as the most recently referenced network printer (PRTNET). |
| PlantUnitName | The name of the plant unit to become the most recently referenced plant unit. If empty, then no plant unit will be set as the most recently referenced plant unit. |
| StrategyName | The name of the strategy to become the most recently referenced strategy. If empty, then no strategy will be set as the most recently referenced strategy. |

| Attribute | Description |
|---|---|
| SwitchName | The name of the switch to become the most recently referenced switch. If empty, then no switch will be set as the most recently referenced switch. |
| WorkstationName | The name of the workstation to become the most recently referenced workstation. If empty, then no workstation will be set as the most recently referenced workstation. |

**Examples:**

Reset the most-recently referenced strategy and compound, so that no references exist:

```
<Reset Strategy="" Compound=""/>
```

Reset the most-recently referenced strategy to reference strategy **MY_STRAT** in compound **COMPND_001**:

```
<Reset Strategy="COMPND_001.MY_STRAT"/>
```

Change the most recently-referenced control processor to **CP2801**:

```
<Reset Controller="CP2801"/>
```

Reset the query filter named **Filter1**. After resetting, the filter **Filter1** will no longer be available:

```
<Reset Filter="Filter1"/>
```

**Note** The same query filter name can be used more than once in a single script, but be sure to include **Reset** commands to avoid inadvertent query combining.

CHAPTER 15

# Attribute Update Operations

This chapter describes attribute update operations, which allow you to update the attributes of various configurable Control Software objects like blocks, compounds, controllers, ECBs, software, and so forth.

## Contents

- UpdateUserAttribute Command
- UpdateWorkstationAttributes Command

# BreakLinkToTemplate Command

The **BreakLinkToTemplate** command allows you to break the link between a derived strategy (template or instance) and its defining template.

**Note** When the link between a derived strategy (template or instance) is broken to its defining template, changes made to the defining template will no longer propagate to the derived strategy. The link cannot be re-established.

You cannot break the link between a child strategy and its defining template if the containing strategy's link is intact. You must break the link at the topmost strategy before the link can be broken in the child strategies.

### Syntax:

```
<BreakLinkToTemplate  Strategy="StrategyName"/>
                      Or
<BreakLinkToTemplate  Filter="FilterName"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy to have its link broken. **StrategyName** should not be specified if **FilterName** is given. |
| FilterName | The name of the filter that will be used to determine what strategies get their links broken. **FilterName** should not be specified if **StrategyName** is given. |

### Examples:

Break the link between strategy **Strategy_001** and its defining template:

```
<BreakLinkToTemplate Strategy="Strategy_001"/>
```

Break the link on all strategies that are derived from **$Strategy_001**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$Strategy_001"/>

<BreakLinkToTemplate Filter="Filter1"/>
```

# MarkAttributeDirty

The **MarkAttributeDirty** command allows you to mark the block's configurable attributes as "dirty" (that is, the block needs redeployment to the Control Processor).

**Syntax:**

```
<MarkAttributeDirty  Strategy="StrategyName"
                     Block="BlockName"
                     ParmName="ParmName"
                     TypeName="TypeName"/>
                     Or
<MarkAttributeDirty  Filter="FilterName"
                     Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. **StrategyName must be an** instance. If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file. |
| BlockName | The name of the block in which the attribute exists. **BlockName** may specify an instance. If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute to be marked dirty in the block. **ParmName** must be the name of a valid control attribute. If not found, an error will result. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName** – not valid if a **FilterName** is specified. **TypeName** defaults to **ContainedName** if not specified and no filter has been specified. **Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be instances of strategies, compounds, controllers and/or equipment units. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

**Examples:**

Mark configurable attributes of deployed block instances as “dirty” if their deployed value does not match the result of their connection resolution value. Note that the “AttributeQueryFilter” **AttributeType=Configurable** is not needed when **AttributeType=NotYetPropagated** is used:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy" />

<AttributeQueryFilter Filter="Filter1" Condition="AttributeType"
Value="NotYetPropagated" />

<MarkAttributeDirty Filter="Filter1"Cascade="Yes" />
```

Mark the **PARM1** attribute of **PID_1** Block in the **MyStrategy1** strategy as “dirty” if the **PID_1** block is deployed and **PARM** is a configurable parameter:

```
<MarkAttributeDirty Strategy="MyStrategy1" Block="PID_1"
ParmName="PARM1" TypeName="Tagname" />
```

# UpdateBlockAttribute Command

The **UpdateBlockAttribute** command allows you to update an attribute value on one or more blocks.

**Syntax:**

```
<UpdateBlockAttribute    Strategy="StrategyName"
                         Block="BlockName"
                         ParmName="ParmName"
                         ParmValue="ParmValue"
                         TypeName="TypeName"/>
```

Or

```
<UpdateBlockAttribute   Filter="FilterName"
                        ParmName="ParmName"
                        ParmValue="ParmValue"
                        Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. **StrategyName** may be a template or instance.<br>If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file. |
| BlockName | The name of the block in which the attribute exists. **BlockName** may specify either a template or an instance.<br>If **BlockName** refers to a block template, then **StrategyName** is ignored.<br>If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute to be updated in the block. **ParmName** must be the name of a valid control block attribute. If not found, an error will result. |
| ParmValue | The value that the attribute is to be updated to.<br>**Note** Although similar in nature to the ability to update a connection reference value via **CreateBlockAddressCxn**, the value placed into an attribute value using the **UpdateAttribute** command will not resolve properly to a C:B.P address at deployment time.<br>Also, **CreateBlockAddressCxn** cannot be used on block templates. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName** – not valid if a **FilterName** is specified.<br>**TypeName** defaults to **ContainedName** if not specified and no filter has been specified.<br>**Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block.<br>This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br><br>**Note** If a filter is composed of only a QueryFilter specification, then the objects returned by the filter must be either block templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates, or instances of strategies, compounds, controllers and/or equipment units. The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

**Examples:**

Update the **HSCO1** attribute on the **MY_AIN** block within strategy **MyStrategy1** to **105.3**:

```
<UpdateBlockAttribute Strategy="MyStrategy1" Block="MY_AIN"
ParmName="HSCO1" ParmValue="105.3"/>
```

Update the **HSCO1** attribute on the $**MY_AIN** block template to **105.3**:

```
<UpdateBlockAttribute Block="$MY_AIN" ParmName="HSCO1"
ParmValue="105.3"/>
```

Update the **HSCO1** attribute on all blocks derived from **$AIN** that are found in **Strategy_002**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy_002"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UpdateBlockAttribute Filter="Filter1" ParmName="HSCO1"
ParmValue="105.3"/>
```

Update the **HSCO1** attribute on all blocks derived from **$AIN** that are found on any strategies and/or child strategies that may be found in compounds assigned to controller **CP0100**.

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UpdateBlockAttribute Filter="Filter1" ParmName="HSCO1"
ParmValue="105.3" Cascade="Yes"/>
```

# UpdateBlockHistory Command

The **UpdateBlockHistory** command allows you to update historization settings for one or more block attributes.

**Syntax:**

```
<UpdateBlockHistory     Strategy="StrategyName"
                        Block="BlockName"
                        ParmName="ParmName"
                        Enable="EnableFlag"
                        EnableLocked="EnableLockedFlag"
                        Description="Description"
                        DescriptionLocked="DescriptionLockedFlag"
                        EngUnits="EngUnits"
                        EngUnitsLocked="EngUnitsLockedFlag"
                        Deadband="Deadband"
                        DeadbandLocked="DeadbandLockedFlag"
                        Period="Period"
                        PeriodLocked="PeriodLockedFlag"
                        TrendHigh="TrendHigh"
                        TrendHighLocked="TrendHighLockedFlag"
                        TrendLow="TrendLow"
                        TrendLowLocked="TrendLowLockedFlag"
                        ScanRate="ScanRate"
                        ScanRateLocked="ScanRateLockedFlag"
                        OnMessage="OnMessage"
                        OnMessageLocked="OnMessageLockedFlag"
                        OffMessage="OffMessage"
                        OffMessageLocked="OffMessageLockedFlag"
                        TypeName="TypeName"/>
```

Or

```
<UpdateBlockHistory Filter="FilterName"
ParmName="ParmName"
Enable="EnableFlag"
EnableLocked="EnableLockedFlag"
Description="Description"
DescriptionLocked="DescriptionLockedFlag"
EngUnits="EngUnits"
EngUnitsLocked="EngUnitsLockedFlag"
Deadband="Deadband"
DeadbandLocked="DeadbandLockedFlag"
Period="Period"
PeriodLocked="PeriodLockedFlag"
TrendHigh="TrendHigh"
TrendHighLocked="TrendHighLockedFlag"
TrendLow="TrendLow"
TrendLowLocked="TrendLowLockedFlag"
ScanRate="ScanRate"
ScanRateLocked="ScanRateLockedFlag"
OnMessage="OnMessage"
OnMessageLocked="OnMessageLockedFlag"
OffMessage="OffMessage"
OffMessageLocked="OffMessageLockedFlag"
Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. **StrategyName** may be a template or instance.<br>If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file. |
| BlockName | The name of the block in which the attribute exists. **BlockName** may specify either a template or an instance.<br>If **BlockName** refers to a block template, then **StrategyName** is ignored.<br>If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute on the block for which historization settings are to be updated. **ParmName** must be the name of a valid control block attribute. If not found, an error will result. |

| Attribute | Description |
|---|---|
| EnableFlag | Indicates whether or not history should be enabled for this attribute. This attribute must be **True** or **False**. If missing and **ParmName** is specified, default value is **False**.<br>If set to **False**, then no other attribute properties should be set. History will be disabled, and all historization settings will be lost. |
| EnableLockedFlag | Indicates whether or not the **EnableFlag** setting is locked – valid only for block templates and blocks within a strategy template. **EnableLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| Description | A description of the attribute for historical trending. Maximum length for this field is 512 characters. |
| DescriptionLockedFlag | Indicates whether or not the **Description** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**.<br>**DescriptionLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| EngUnits | Engineering units for deadband, high and low trends. Maximum length for this field is 32 characters. |
| EngUnitsLockedFlag | Indicates whether or not the **EngUnits** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| Deadband | For numerical attributes, the threshold value in engineering units between the new and last-stored values before storing the new value to history. For example, a deadband value of 5 means that the new value will not be historized until it changes more than 5 units from the previous value. Zero (0) is the default. Not applicable for string, Boolean or enumeration attributes. |
| DeadbandLockedFlag | Indicates whether or not the **Deadband** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**.<br>**DeadbandLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

| Attribute | Description |
|---|---|
| Period | The Force Storage Period, or the time interval (in milliseconds) after which the value must be historized even if the value has not changed. A value of zero (0) disables this feature. |
| PeriodLockedFlag | Indicates whether or not the **Period** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendHigh | For numerical attributes, the default top of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendHighLockedFlag | Indicates whether or not the **TrendHigh** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendLow | For numerical attributes, the default bottom of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendLowLockedFlag | Indicates whether or not the **TrendLow** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| ScanRate | The scan rate, in milliseconds. Allowed values are 100, 500, 1000, 1500, …9500, 10000 |
| ScanRateLockedFlag | Indicates whether or not the **ScanRate** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**. **ScanRateLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| OnMessage | Custom message string for Boolean data type only. Empty if data type other than Boolean, or if not desired. **True** is the default string. Historian clients (for example, trends) display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |

| Attribute | Description |
|---|---|
| OnMessageLockedFlag | Indicates whether or not the **OnMessage** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**.<br>**OnMessageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| OffMessage | Custom message string for Boolean data type only. Empty if data type other than Boolean, or if not desired. **False** is the default string. Historian clients (for example, trends) display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |
| OffMesageLockedFlag | Indicates whether or not the **OffMessage** setting is locked – valid only for block templates and blocks within a strategy template, and **EnableLockedFlag** is **True**.<br>**OffMesageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName**.<br>**TypeName** defaults to **ContainedName** if not specified.<br>**Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br><br>**Note** If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either block templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates, or instances of strategies, compounds, controllers and/or equipment units.<br>The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

### Examples:

Update the historization settings for the **ACHNGE** attribute on the block template **$MY_AIN**:

```
<UpdateBlockHistory Block="$MY_AIN" ParmName="ACHNGE"
Description="Test description" Deadband="2" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" Period="10" ScanRate="100"
OnMessage="Turned On" OffMessage="Turned Off" Enable="True"/>
```

Update the historization settings for the **ACHNGE** attribute on the block instance **AIN_1** contained within the strategy template **$Strategy_001**:

```
<UpdateBlockHistory Strategy="$Strategy_001" Block="AIN_1"
ParmName="ACHNGE" Description="Test description" Deadband="2"
TrendHigh="100.0" TrendLow="10.2" EngUnits="%" Period="10"
ScanRate="100" OnMessage="Turned On" OffMessage="Turned Off"
Enable="True"/>
```

Disable historization for the **ACHNGE** attribute on the block template **$MY_AIN**:

```
<UpdateBlockHistory Block="$MY_AIN" ParmName="ACHNGE"
Enable="False"/>
```

Update the historization settings for the **ACHNGE** attribute on all blocks derived from **$AIN** on all strategies derived from **$Strategy**, and contained within compound **MY_COMPND**:

```
<QueryFilter Filter="Filter1" Condition="DerivedOrInstantiatedFrom"
Value="$Strategy"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="MY_COMPND"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UpdateBlockHistory Filter="Filter1" ParmName="ACHNGE"
Description="Test description" Deadband="4" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" Period="10" ScanRate="100"
OnMessage="Turned On" OffMessage="Turned Off" Enable="True"
EnableLocked="True" DeadbandLocked="True"/>
```

Update the historization settings for the **ACHNGE** and **ALMSTA** attributes on all blocks derived from **$AIN** on all strategies derived from **$Strategy**, and contained within compound **MY_COMPND**:

```
<QueryFilter Filter="Filter1" Condition="DerivedOrInstantiatedFrom"
Value="$Strategy"/>

<QueryFilter Filter="Filter1" Condition="ContainedBy"
Value="MY_COMPND"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="ACHNGE"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="ALMSTA"/>

<UpdateBlockHistory Filter="Filter1" Enable="True"
Description="Test description" Deadband="4" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" Period="10" ScanRate="100" />
```

# UpdateBlockSecurity Command

The **UpdateBlockSecurity** command allows you to update the security setting for one or more block attributes.

### Syntax:

```
<UpdateBlockSecurity    Strategy="StrategyName"
                        Block="BlockName"
                        ParmName="ParmName"
                        Security="Security"
                        TypeName="TypeName"/>
```

Or

```
<UpdateBlockSecurity  Filter="FilterName"
                      ParmName="ParmName"
                      Security="Security"
                      Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. **StrategyName** may be a template or instance. If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file. |
| BlockName | The name of the block in which the attribute exists. **BlockName** may specify either a template or an instance. If **BlockName** refers to a block template, then **StrategyName** is ignored. If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute on the block for which the security setting is to be updated. **ParmName** must be the name of a valid control block attribute. If not found, an error will result. |
| Security | The security classification to use for this attribute. This attribute must be set to one of the following: **Configure**, **FreeAccess**, **Operate**, **ReadOnly**, **Tune**, **SecuredWrite**, **VerifiedWrite**. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName**. **TypeName** defaults to **ContainedName** if not specified. **Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block. This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br><br>**Note** If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either block templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates, or instances of strategies, compounds, controllers and/or equipment units. The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

**Examples:**

Update the security for the **KSCALE** attribute on the **MY_AIN** block within strategy **$MyStrategy1** to **VerifiedWrite**:

```
<UpdateBlockAttribute Strategy="$MyStrategy1" Block="MY_AIN"
ParmName="KSCALE" Security="VerifiedWrite"/>
```

Update the security for the **KSCALE** attribute on the **$MY_AIN** block template to **ReadOnly**:

```
<UpdateBlockSecurity Block="$MY_AIN" ParmName="KSCALE"
Security="ReadOnly"/>
```

Update the security for the **KSCALE** attribute on all blocks derived from **$AIN** that are found in **Strategy_002**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy_002"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UpdateBlockSecurity Filter="Filter1" ParmName="KSCALE"
Security="SecuredWrite" Cascade="Yes"/>
```

Update the security for the **KSCALE** attribute on all blocks derived from **$AIN** that are found on any strategies and/or child strategies that may be found in compounds assigned to controller **CP0100**.

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UpdateBlockSecurity Filter="Filter1" ParmName="KSCALE"
Security="Tune" Cascade="Yes"/>
```

# UpdateBlockSource Command

The **UpdateBlockSource** command allows you to update a block's source code. It applies to the following block types:

- PLB
- IND (HLBL,SFC)
- DEP (HLBL,SFC)
- EXC (HLBL,SFC)
- MON (HLBL)
- MATH
- LOGIC
- CALC
- CALCA

**Syntax:**

```
<UpdateBlockSource      Strategy="StrategyName"
                        Block="BlockName"
                        Source="BlockSource" />
```

| Attribute | Description |
|---|---|
| Strategy | The name of the strategy in which the block resides. The strategy may be a template or instance. |

| Attribute | Description |
|---|---|
| Block | The name of the block to update the source. The block name may specify either a template or an instance. If the block name refers to a block template, then the strategy name is ignored. |
| Source | This attribute provides the full path to the file(s) containing the source for the block to be created. The value of this attribute must be the full path to the source file(s), including base filename without any file extension. Direct Access software will append the appropriate extension, depending on the block type being updated, according to the following list:<br>Sequence Blocks:<br>• .s (HLBL source)<br>• .g (SFC source)<br>• .k (SFC source)<br>• .vsd (SFC Graphics)<br>PLB Blocks:<br>• .p (Ladder source)<br>• .vsd (Ladder graphics)<br>Logic Blocks:<br>• .rpn (Step text)<br>• .vsd (Graphical logic) |

**Examples:**

Update the source for template **$MY_DEP**:

```
<UpdateBlockSource Block="$MY_DEP"
Source="C:\Temp\InFusionSupport\$MY_DEP"/>
```

The above command will look for the following files:

- C:\Temp\InFusionSupport\$MY_DEP.s
- C:\Temp\InFusionSupport\$MY_DEP.g
- C:\Temp\InFusionSupport\$MY_DEP.k
- C:\Temp\InFusionSupport\$MY_DEP.vsd

If any of these files are present, they will be used to update the block.

Update the source for **MY_DEP** in strategy instance **Strategy_001**:

```
:<UpdateBlockSource Strategy="Strategy_001" Block="MY_DEP"
Source="C:\Temp\InFusionSupport\MY_DEP"/>
```

# UpdateCompoundAttribute Command

The **UpdateCompoundAttribute** command allows you to update an attribute value on one or more compounds.

### Syntax:

```
<UpdateCompoundAttribute  Compound="CompoundName"
                          ParmName="ParmName"
                          ParmValue="ParmValue"/>

                          Or

<UpdateCompoundAttribute  Filter="FilterName"
                          ParmName="ParmName"
                          ParmValue="ParmValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to update. **CompoundName** may be a template or instance, and should not be specified if **FilterName** is given. |
| ParmName | The name of the attribute to be updated in the compound. **ParmName** must be the name of a valid control block attribute. If not found, an error will result. |
| ParmValue | The value that the attribute is to be updated to. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. This attribute should not be specified if **CompoundName** is given. |

### Examples:

Update the **PHASE** attribute on the **MY_COMPND** compound to 2:

```
<UpdateCompoundAttribute Compound="MY_COMPND"
ParmName="PHASE" ParmValue="2"/>
```

Update the **PHASE** attribute on the **$CMPND002** compound to **2**. The big difference between this and the previous example is that if the attribute **PHASE** is locked in the compound **$CMPND002**, the value will ripple to any templates or instances derived from that compound:

```
<UpdateCompoundAttribute Compound="$CMPND002"
ParmName="PHASE" ParmValue="2"/>
```

Update the **PHASE** attribute on all the compounds assigned to controller **CP0002** to **2**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0002"/>

<UpdateCompoundAttribute Filter="Filter1" ParmName="PHASE"
ParmValue="2"/>
```

# UpdateCompoundHistory Command

The **UpdateCompoundHistory** command allows you to update historization settings for a compound attribute, including collecting workstation on one or more compounds.

**Syntax:**

```
<UpdateCompoundHistory  Compound="CompoundName"
                        Target="TargetName"
                        ParmName="ParmName"
                        Enable="EnableFlag"
                        EnableLocked="EnableLockedFlag"
                        Description="Description"
                        DescriptionLocked="DescriptionLockedFlag"
                        EngUnits="EngUnits"
                        EngUnitsLocked="EngUnitsLockedFlag"
                        Deadband="Deadband"
                        DeadbandLocked="DeadbandLockedFlag"
                        Period="Period"
                        PeriodLocked="PeriodLockedFlag"
                        TrendHigh="TrendHigh"
                        TrendHighLocked="TrendHighLockedFlag"
                        TrendLow="TrendLow"
                        TrendLowLocked="TrendLowLockedFlag"
                        ScanRate="ScanRate"
                        ScanRateLocked="ScanRateLockedFlag"
                        OnMessage="OnMessage"
                        OnMessageLocked="OnMessageLockedFlag"
                        OffMessage="OffMessage"
                        OffMessageLocked="OffMessageLockedFlag"
                        />
```

Or

```
<UpdateCompoundHistory Filter="FilterName"
Cascade="CascadeValue"
Target="TargetName"
ParmName="ParmName"
Enable="EnableFlag"
EnableLocked="EnableLockedFlag"
Description="Description"
DescriptionLocked="DescriptionLockedFlag"
EngUnits="EngUnits"
EngUnitsLocked="EngUnitsLockedFlag"
Deadband="Deadband"
DeadbandLocked="DeadbandLockedFlag"
Period="Period"
PeriodLocked="PeriodLockedFlag"
TrendHigh="TrendHigh"
TrendHighLocked="TrendHighLockedFlag"
TrendLow="TrendLow"
TrendLowLocked="TrendLowLockedFlag"
ScanRate="ScanRate"
ScanRateLocked="ScanRateLockedFlag"
OnMessage="OnMessage"
OnMessageLocked="OnMessageLockedFlag"
OffMessage="OffMessage"
OffMessageLocked="OffMessageLockedFlag"
/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to be updated. **CompoundName** should not be specified if **FilterName** is given. If neither FilterName nor CompoundName is supplied, the most recently referenced compound is used. |
| Cascade Value | Specifies if this command should be applied to all assigned objects. If the filter used for this command returns Controllers or Equipment Units, setting CascadeValue to “Yes” will apply this command to all assigned compounds under the objects returned by the QueryFilter. The default value for this option is “No”. |
| TargetName | The name of the collecting workstation (for example, **AW0001**). This is the only property that needs to be specified if the only thing that’s being set is the collecting workstation (that is, **ParmName** and every other property may be left unspecified).<br>If empty, or not specified, the existing **TargetName** will be left unchanged. |

| Attribute | Description |
|---|---|
| ParmName | The name of the attribute on the compound for which historization settings are to be updated. **ParmName** must be the name of a valid control attribute. If not found, an error will result. **ParmName** does not need to be specified if specifying only the name of the collecting workstation |
| EnableFlag | Indicates whether or not history should be enabled for this attribute. **EnableFlag** must be **True** or **False**. If missing and **ParmName** is specified, default value is **False**.<br>If set to **False**, then no other attribute properties should be set. History will be disabled, and all historization settings will be lost. |
| EnableLockedFlag | Indicates whether or not the **EnableFlag** setting is locked – valid only for compound templates. **EnableLockedFlag** must be **True** or **False**. If it is missing and **ParmName** is specified, the default value is **False**. |
| Description | A description of the attribute for historical trending. The maximum length for this field is 512 characters. |
| DescriptionLockedFlag | Indicates whether or not the **Description** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**.<br>**DescriptionLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| EngUnits | Engineering units for deadband, high and low trends. The maximum length for this field is 32 characters. |
| EngUnitsLockedFlag | Indicates whether or not the **EngUnits** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **EngUnitsLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| Deadband | For numerical attributes, the threshold value in engineering units between the new and last-stored values before storing the new value to history. For example, a deadband value of 5 means that the new value will not be historized until it changes more than 5 units from the previous value. Zero (0) is the default. Not applicable for string, Boolean or enumeration attributes. |
| DeadbandLockedFlag | Indicates whether or not the **Deadband** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

| Attribute | Description |
|---|---|
| Period | The Force Storage Period, or the time interval (in milliseconds) after which the value must be historized even if the value has not changed. A value of zero (0) disables this feature. |
| PeriodLockedFlag | Indicates whether or not the **Period** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendHigh | For numerical attributes, the default top of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendHighLockedFlag | Indicates whether or not the **TrendHigh** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendLow | For numerical attributes, the default bottom of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendLowLockedFlag | Indicates whether or not the **TrendLow** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| ScanRate | The scan rate, in milliseconds. Allowed values are 100, 500, 1000, 1500, …, 9500, 10000. |
| ScanRateLockedFlag | Indicates whether or not the **ScanRate** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **ScanRateLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| OnMessage | Custom message string for Boolean data type only. Empty if data type other than Boolean, or if not desired. **True** is the default string. Historian clients (for example, trends) display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |
| OnMessageLockedFlag | Indicates whether or not the **OnMessage** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **OnMessageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

| Attribute | Description |
|---|---|
| OffMessage | Custom message string for Boolean data type only. Empty if data type other than Boolean, or if not desired. **False** is the default string. Historian clients (for example, trends) display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |
| OffMessageLockedFlag | Indicates whether or not the **OffMessage** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **OffMesageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. This attribute should not be specified if **CompoundName** is given. |

**Examples:**

Update the historization settings for the **ALMLEV** attribute on the compound template **$COMPND_001**:

```
<UpdateCompoundHistory Compound="$COMPND_001"
ParmName="ALMLEV" Description="Test description" Deadband="2"
TrendHigh="100.0" TrendLow="10.2" EngUnits="%" Period="10"
ScanRate="100" OnMessage="Turned On" OffMessage="Turned Off"
Enable="True"/>
```

Update the collecting workstation for the compound template **$COMPND_001**. Leave all other historization settings unchanged:

```
<UpdateCompoundHistory Compound="$COMPND_001"
Target="AW0001"/>
```

Update the historization settings for the **ALMLEV** attribute on the compound instance **MY_COMPND**:

```
<UpdateCompoundHistory Compound="MY_COMPND"
ParmName="ALMLEV" Description="Test description" Deadband="2"
TrendHigh="100.0" TrendLow="10.2" EngUnits="%" Period="10"
ScanRate="100" OnMessage="Turned On" OffMessage="Turned Off"
Enable="True />
```

Disable historization settings for the **ALMLEV** attribute on the compound template **$COMPND_001**:

```
<UpdateCompoundHistory Compound="$COMPND_001"
ParmName="ALMLEV" Enable="False"/>
```

Update historization settings for the **ALMLEV** attribute on all compounds assigned to the controller **CP0002**. Lock the historization settings for deadband and engineering units:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>
```

```
<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0002"/>

<UpdateCompoundHistory Filter="Filter1" ParmName="ALMLEV"
Enable="True" EnableLocked="True" Description="Test description"
Deadband="2" DeadbandLocked="True" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" EngUnitsLocked="True" Period="10"
ScanRate="100" OnMessage="Turned On" OffMessage="Turned Off"/>
```

Update the historization settings for the **ALMLEV** and **CINHIB** attributes on all compounds contained by all controllers assigned to the Equipment Unit "EquipUnit_001":

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="EquipUnit_001"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="ALMLEV"/>

<AttributeQueryFilter Filter="Filter1" Condition="AttributeName"
Value="CINHIB"/>

<UpdateCompoundHistory Filter="Filter1" Enable="True"
Cascade="Yes" Description="Test description" Deadband="4"
TrendHigh="100.0" TrendLow="10.2" EngUnits="%" Period="10"
ScanRate="100" />
```

# UpdateCompoundSecurity Command

The **UpdateCompoundSecurity** command allows you to update the security setting for an attribute on one or more compounds.

### Syntax:

```
<UpdateCompoundSecurity  Compound="CompoundName"
                         ParmName="ParmName"
                         Security="Security"/>

                         Or

<UpdateCompoundSecurity  Filter="FilterName"
                         ParmName="ParmName"
                         Security="Security"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to be updated. **CompoundName** should not be specified if **FilterName** is given. |
| ParmName | The name of the attribute on the compound for which the security setting is to be updated. **ParmName** must be the name of a valid control attribute. If not found, an error will result. |

| Attribute | Description |
|---|---|
| Security | The security classification to use for this attribute. **Security** must be set to one of the following: **Configure**, **FreeAccess**, **Operate**, **ReadOnly**, **Tune**, **SecuredWrite**, **VerifiedWrite**. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. **FilterName** should not be specified if **CompoundName** is given. |

**Examples:**

Update the security setting for the **CINHIB** attribute on the compound template **$COMPND_001**:

```
<UpdateCompoundSecurity Compound="$COMPND_001"
ParmName="CINHIB" Security="FreeAccess"/>
```

Update the security setting for the **CINHIB** attribute on the compound instance **MY_COMPND**:

```
<UpdateCompoundSecurity Compound="MY_COMPND"
ParmName="CINHIB" Security="FreeAccess"/>
```

Update the security setting for the **CINHIB** attribute on all compounds assigned to the controller **CP0002**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="AssignedTo"
Value="CP0002"/>

<UpdateCompoundSecurity Filter="Filter1" ParmName="CINHIB"
Security="FreeAccess"/>
```

# UpdateControllerAttributes Command

The **UpdateControllerAttributes** command allows you to update non-software attributes for a controller. Any or all attributes may be specified in a single command.

**Syntax:**

```
<UpdateControllerAttributes    Controller="ControllerName"
                               DESCRP="DESCRPValue"
                               TCPIP="TCPIP_Address"
                               FTMAC="FTMAC_Address"
                               FT="FaultTolerant"/>
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller to be updated.<br>If this attribute is missing or blank, the most recently referenced controller will be used. If no controller had been previously referenced within the XML data, an error will result.<br>A query filter is not supported for this command. |
| DESCRPValue | Specifies Description text for this controller. |
| TCPIP_Address | Specifies the TCP/IP address for this controller. Care should be taken to ensure that a valid TCP/IP address is specified, or an error will result during hardware validation.<br>Validation will catch duplicate and improperly formatted TCP/IP addresses. |
| FTMAC_Address | Specifies the FTMAC address for this controller. Care should be taken to ensure that a valid FTMAC address is specified, or an error will result during hardware validation.<br>Validation will catch duplicate and improperly formatted FTMAC addresses. |
| FaultTolerant | Specifies the fault tolerance for this controller. Valid values are:<br>**S** – Single<br>**F** – Fault-Tolerant |

### Examples:

Update the TCP/IP address for controller **CP0100** to **151.128.152.002**:

```
<UpdateControllerAttributes Controller="CP0100"
TCPIP="151.128.152.002"/>
```

Update the FTMAC address for controller **CP0100** to **C0000A**, and specify a fault-tolerance of **S**:

```
<UpdateControllerAttributes Controller="CP0100" FTMAC="C0000A"
FT="S />
```

# UpdateDeclaration Command

The **UpdateDeclaration** command allows you to update a declaration value on a strategy.

### Syntax:

```
<UpdateDeclaration   Decl="DeclName"
                     Strategy="StrategyName"
                     Value="DeclValue"/>
```

| Attribute | Description |
|---|---|
| DeclName | The name of the declaration whose value is to be updated. |
| StrategyName | The name of the strategy that this declaration is to be modified in.<br>If **StrategyName** is missing, or blank, the declaration will be modified in the most recently referenced strategy. If no strategy had been previously referenced, an error will result. |
| Value | The value that the declaration is to be updated to.<br>If the declaration is an input declaration, the declaration will be updated with the exact string as entered for **DeclValue**.<br>If the declaration is an output declaration, the relative address "**Me.**" will be automatically placed in front of the value. |

**Examples:**

Update the output declaration **Output** on strategy template **$Cascade** to point to **Me.SECONDARY.OUT**:

```
<UpdateDeclaration Decl="Output" Strategy="$Cascade"
Value="SECONDARY.OUT"/>
```

Update the input declaration **My_Input** on the strategy **Strategy_001** to point to the attribute **COMPND_001.Strategy_002.My_Output.**

```
<UpdateDeclaration Decl="My_Input" Strategy="Strategy_001"
Value="COMPND_001.Strategy_002.My_Output"/>
```

# UpdateECBAttribute Command

The **UpdateECBAttribute** command allows you to update an attribute value on an ECB template or instance.

**Syntax:**

```
<UpdateECBAttribute   Compound="CompoundName"
                      ECB="ECBName"
                      ParmName="ParmName"
                      ParmValue="ParmValue"/>

                      Or

<UpdateECBAttribute   Filter="FilterName"
                      ECB="ECBName"
                      ParmName="ParmName"
                      ParmValue="ParmValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound in which the ECB resides. This attribute should not be specified if **FilterName** is given. |
| ECBName | The name of the ECB in which the attribute exists. **ECBName** may specify either a template or an instance. If **ECBName** refers to an ECB template, then **CompoundName** is ignored.<br>If **ECBName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute to be updated in the ECB. **ParmName** must be the name of a valid control attribute. If not found, an error will result. |
| ParmValue | The value that the attribute is to be updated to. |
| FilterName | The name of the query filter that will be used to determine what ECBs in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **CompoundName** and **ECBName** should not be.<br>The objects returned by the filter must be either ECB templates or compound instances.<br>The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |

### Examples:

Update the **PHASE** attribute on the **$MY_ECB** template to 2:

```
<UpdateECBAttribute ECB="$MY_ECB" ParmName="PHASE"
ParmValue="2"/>
```

Update the **PHASE** attribute on the ECB named **F00002** contained within the compound **CP0002_ECB**:

```
<UpdateECBAttribute Compound="CP0002_ECB” ECB="F00002"
ParmName="PHASE" ParmValue="2"/>
```

Update the **PHASE** attribute on all ECBs derived from **$ECB1_001** that are in the ECB compound **CP0003_ECB**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0003_ECB"/>

<BlockQueryFilter Filter="Filter1"
Condition="DerivedOrInstantiatedFrom" Value="$ECB1_001"/>

<UpdateECBAttribute Filter="Filter1" ParmName="PHASE"
ParmValue="3"/>
```

# UpdateECBHistory Command

The **UpdateECBHistory** command allows you to update historization settings for an ECB template or instance compound attribute.

**Syntax:**

```
<UpdateECBHistory    Compound="CompoundName"
                     ECB="TargetName"
                     ParmName="ParmName"
                     Enable="EnableFlag"
                     EnableLocked="EnableLockedFlag"
                     Description="Description"
                     DescriptionLocked="DescriptionLockedFlag"
                     EngUnits="EngUnits"
                     EngUnitsLocked="EngUnitsLockedFlag"
                     Deadband="Deadband"
                     DeadbandLocked="DeadbandLockedFlag"
                     Period="Period"
                     PeriodLocked="PeriodLockedFlag"
                     TrendHigh="TrendHigh"
                     TrendHighLocked="TrendHighLockedFlag"
                     TrendLow="TrendLow"
                     TrendLowLocked="TrendLowLockedFlag"
                     ScanRate="ScanRate"
                     ScanRateLocked="ScanRateLockedFlag"
                     OnMessage="OnMessage"
                     OnMessageLocked="OnMessageLockedFlag"
                     OffMessage="OffMessage"
                     OffMessageLocked="OffMessageLockedFlag"/>
```

Or

```
<UpdateECBHistory   Filter="FilterName"
                    ParmName="ParmName"
                    Cascade="CascadeValue"
                    Enable="EnableFlag"
                    EnableLocked="EnableLockedFlag"
                    Description="Description"
                    DescriptionLocked="DescriptionLockedFlag"
                    EngUnits="EngUnits"
                    EngUnitsLocked="EngUnitsLockedFlag"
                    Deadband="Deadband"
                    DeadbandLocked="DeadbandLockedFlag"
                    Period="Period"
                    PeriodLocked="PeriodLockedFlag"
                    TrendHigh="TrendHigh"
                    TrendHighLocked="TrendHighLockedFlag"
                    TrendLow="TrendLow"
                    TrendLowLocked="TrendLowLockedFlag"
                    ScanRate="ScanRate"
                    ScanRateLocked="ScanRateLockedFlag"
                    OnMessage="OnMessage"
                    OnMessageLocked="OnMessageLockedFlag"
                    OffMessage="OffMessage"
                    OffMessageLocked="OffMessageLockedFlag"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound containing the ECB to be updated. **CompoundName** should not be specified if **FilterName** is given, or if the ECB being updated is an ECB template. |
| ECBName | The name of the ECB instance or template to be updated. **ECBName** should not be specified if **FilterName** is given. |
| CascadeValue | Specifies if this command should be applied to all assigned objects. If the filter used for this command returns Compounds, Controllers, or Equipment Units, setting CascadeValue to “Yes” will apply this command to all contained ECBs under the objects returned by the QueryFilter. The default value for this option is “No”. |

| Attribute | Description |
|---|---|
| ParmName | The name of the attribute on the ECB for which historization settings are to be updated. **ParmName** must be the name of a valid control attribute. If not found, an error will result. **ParmName** will be ignored if the **FilterName** is specified and the filter contains an AttributeQueryFilter component and the AttributeQueryFilter specification will be used. If a **FilterName** is supplied, no AttributeQueryFilter is specified, and no ParmName is given then the historization settings will be applied to all parameters of the filtered ECBs. |
| FilterName | The name of the filter that will be used to determine what ECBs get updated. This attribute should not be specified if **CompoundName** and **ECBName** are given. |
| EnableFlag | Indicates whether or not history should be enabled for this attribute. **EnableFlag** must be **True** or **False**. If missing and **ParmName** is specified, default value is **False**.<br>If set to **False**, then no other attribute properties should be set. History will be disabled, and all historization settings will be lost. |
| EnableLockedFlag | Indicates whether or not the **EnableFlag** setting is locked – valid only for compound templates. **EnableLockedFlag** must be **True** or **False**. If it is missing and **ParmName** is specified, the default value is **False**. |
| Description | A description of the attribute for historical trending. The maximum length for this field is 512 characters. |
| DescriptionLockedFlag | Indicates whether or not the **Description** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **DescriptionLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| EngUnits | Engineering units for deadband, high and low trends. The maximum length for this field is 32 characters. |
| EngUnitsLockedFlag | Indicates whether or not the **EngUnits** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **EngUnitsLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

| Attribute | Description |
|---|---|
| Deadband | For numerical attributes, the threshold value in engineering units between the new and last-stored values before storing the new value to history. For example, a deadband value of 5 means that the new value will not be historized until it changes more than 5 units from the previous value. The default value is zero (0). Not applicable for string, Boolean, or enumeration attributes. |
| DeadbandLockedFlag | Indicates whether or not the **Deadband** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| Period | The Force Storage Period, or the time interval (in milliseconds) after which the value must be historized even if the value has not changed. A value of zero (0) disables this feature. |
| PeriodLockedFlag | Indicates whether or not the **Period** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendHigh | For numerical attributes, the default top of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendHighLockedFlag | Indicates whether or not the **TrendHigh** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| TrendLow | For numerical attributes, the default bottom of the trend scale. Not applicable for string, Boolean, or enumeration attributes. |
| TrendLowLockedFlag | Indicates whether or not the **TrendLow** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. This attribute must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| ScanRate | The scan rate, in milliseconds. Allowed values are 100, 500, 1000, 1500, …, 9500, 10000. |
| ScanRateLockedFlag | Indicates whether or not the **ScanRate** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **ScanRateLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

| Attribute | Description |
|---|---|
| OnMessage | Custom message string for Boolean data type only. Empty if data type other than Boolean, or if not desired. **True** is the default string. Historian clients (for example, trends) display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |
| OnMessageLockedFlag | Indicates whether or not the **OnMessage** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **OnMessageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |
| OffMessage | Custom message string for Boolean data type only. Empty if data type is other than Boolean, or is not desired. **False** is the default string. Historian clients such as trends, display the names of the two Boolean states on the trends value axis. Maximum length for this field is 15 characters. |
| OffMessageLockedFlag | Indicates whether or not the **OffMessage** setting is locked – valid only for compound templates, and **EnableLockedFlag** is **True**. **OffMesageLockedFlag** must be **True** or **False**. If missing and **ParmName** is specified, the default value is **False**. |

### Examples:

Update the historization settings for the **ACHNGE** attribute on the ECB template **$MY_ECB200**:

```
<UpdateECBHistory ECB="$MY_ECB200" ParmName="ACHNGE"
Description="Test description" Deadband="2" TrendHigh="100.0"
TrendLow="10.2" EngUnits="%" Period="10" ScanRate="100"
Enable="True" EnableLocked="True"/>
```

Update the historization settings for the **ACHNGE** attribute of all ECBs (including the primary ECB) in compound **CP0100_ECB**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="CP0100_ECB"/>

<UpdateECBHistory Filter="Filter1" Cascade="Yes"
ParmName="ACHNGE" Description="Test description" Deadband="2"
TrendHigh="100.0" TrendLow="10.2" EngUnits="%" Period="10"
ScanRate="100" Enable="True />
```

Disable historization settings for the **ACHNGE** attribute on the ECB instance F00001:

```
<UpdateECBHistory Compound="COMPND_001" ECB="F00001"
ParmName="ACHNGE" Enable="False"/>
```

Update the historization settings for the **ACHNGE** and **ACTION** attributes on all ECBs contained by compounds named like "COMPND_H%", derived from base template $ECB200:

```
<QueryFilter Filter="Filter2" Condition="NamedLike"
Value="COMPND_H%"/>

<BlockQueryFilter Filter="Filter2" Condition="BasedOn"
Value="$ECB200"/>

<AttributeQueryFilter Filter="Filter2" Condition="AttributeName"
Value="ACHNGE"/>

<AttributeQueryFilter Filter="Filter2" Condition="AttributeName"
Value="ACTION"/>

<UpdateECBHistory Filter="Filter2" Enable="True" Description="New
description" Deadband="5" TrendHigh="134.0" TrendLow="24.2"
EngUnits="pct" Period="8" ScanRate="500" />
```

# UpdateECBSecurity Command

The **UpdateECBSecurity** command allows you to update the security setting for one or more ECB attributes.

**Syntax:**

```
<UpdateECBSecurity    Compound="CompoundName"
                      ECB="ECBName"
                      ParmName="ParmName"
                      Security="Security"/>

                      Or

<UpdateECBSecurity    Filter="FilterName"
                      ParmName="ParmName"
                      Security="Security"
                      Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound in which the ECB resides. **CompoundName** may only be an instance. If **CompoundName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced compound within the command file. |
| ECBName | The name of the ECB in which the attribute exists. **ECBName** may specify either a template or an instance. If **ECBName** refers to an ECB template, then **CompoundName** is ignored. If **ECBName** is not specified or blank, then **FilterName** must be specified. |

| Attribute | Description |
|---|---|
| ParmName | The name of the attribute on the ECB for which the security setting is to be updated. **ParmName** must be the name of a valid control attribute. If not found, an error will result. |
| Security | The security classification to use for this attribute. This attribute must be set to one of the following: **FreeAccess**, **Operate**, **Tune**, **Configure**, **SecuredWrite**, **VerifiedWrite**, **ReadOnly**. |
| FilterName | The name of the query filter that will be used to determine what ECBs in the Galaxy are to be considered for this operation. If **FilterName** is specified, then **CompoundName** and **ECBName** should NOT be.<br><br>NOTE: If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either ECB templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either ECB templates or instances. The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName** and **ParmValue**. |
| CascadeValue | Specifies if this command should be applied to all assigned objects. If the filter used for this command returns Compounds, Controllers, or Equipment Units, setting **CascadeValue** to "**Yes**" will apply this command to all contained ECBs under the objects returned by the **QueryFilter**. The default value for this option is "**No**". |

### Examples:

Update the security for the **ACTION** attribute in the ECB template for **MyECB2** to **FreeAccess**:

```
<UpdateECBSecurity ECB="$MyECB2" ParmName="ACTION"
Security="FreeAccess"/>
```

Update the security for the **CHAN** attribute in ECB **F00013** to **ReadOnly**:

```
<UpdateECBSecurity Compound="FCP280_ECB" ECB="F00013"
ParmName="ACTION" Security="ReadOnly"/>
```

Update the security for the ACTION attribute in all instances of ECB02 within **FCP280_ECB** to Configure:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="FCP280_ECB"/>
```

```
<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$ECB2"/>

<UpdateECBSecurity Filter="Filter1" ParmName="ACTION"
Security="Configure" Cascade="Yes" />
```

# UpdateFBMAttributes Command

The **UpdateFBMAttributes** command allows you to update non-software attributes for an FBM. Any or all attributes may be specified in a single command.

### Syntax:

```
<UpdateFBMAttributes    FBM="FBMName"
                        DESCRP="DESCRPValue"
                        FT="FaultTolerant"
                        IOM="IOMPackage"/>
```

| Attribute | Description |
|---|---|
| FBMName | The name of the FBM to be updated.<br>If this attribute is missing or blank, the most recently referenced FBM will be used. If no FBM had been previously referenced within the XML data, an error will result.<br>A query filter is not supported for this command. |
| DESCRPValue | Specifies Description text for this FBM. |
| FaultTolerant | Specifies the fault tolerance for this FBM. Valid values are:<br>**S** – Single<br>**D** – Dual |
| IOMPackage | Specifies a new IOM package for those FBM types that support a variety of IOM packages.<br>**Note** Changing the IOM type automatically changes the type of ECB that's associated with the specified FBM.<br>If the IOM type specified is not valid for that FBM, a warning will result. |

### Examples:

Update the fault tolerance for FBM **F00001** to dual (D):

```
<UpdateFBMAttributes FBM="F00001" FT="D"/>
```

Update the IOM software package for FBM **F00002** to iom08 (note that this will automatically change the ECB associated with F0002 to be of type ECB08):

```
<UpdateFBMAttributes FBM="F00001" IOM="iom08"/>
```

# UpdateFCMAttributes Command

The **UpdateFCMAttributes** command allows you to update non-software attributes for an FCM. Any or all attributes may be specified in a single command.

### Syntax:

```
<UpdateFCMAttributes   FCM="FCMName"
                       DESCRP="DESCRPValue"
                       TCPIP="TCPIP_Address"
                       FTMAC="FTMAC_Address"
                       FT="FaultTolerant"/>
```

| Attribute | Description |
|---|---|
| FCMName | The name of the FCM to be updated.<br>If **FCMName** is missing or blank, the most recently referenced FCM will be used. If no FCM had been previously referenced within the XML data, an error will result.<br>A query filter is not supported for this command. |
| DESCRPValue | Specifies Description text for this FCM. |
| TCPIP_Address | Specifies the TCP/IP address for this FCM. Care should be taken to ensure that a valid TCP/IP address is specified, or an error will result during hardware validation.<br>Validation will catch duplicate and improperly formatted TCP/IP addresses. |
| FTMAC_Address | Specifies the FTMAC address for this FCM. Care should be taken to ensure that a valid FTMAC address is specified, or an error will result during hardware validation.<br>Validation will catch duplicate and improperly formatted FTMAC addresses. |
| FaultTolerant | Specifies the fault tolerance for this FCM. Valid values are:<br>**S** – Single<br>**D** – Dual |

### Examples:

Update the TCPIP address for FCM **FCM100** to **151.128.152.002**:

```
<UpdateFCMAttributes FCM="FCM100" TCPIP="151.128.152.002"/>
```

Update the FTMAC address for FCM **FCM100** to **C0000A**, and specify a fault tolerance of **S**:

```
<UpdateFCMAttributes FCM="FCM100" FTMAC="C0000A"
FT="S" />
```

# UpdateNetworkPrinterAttributes Command

The **UpdateNetworkPrinterAttributes** command allows you to update non-software attributes for a network printer. Any or all attributes may be specified in a single command.

### Syntax:

```
<UpdateNetworkPrinter    Printer="PrinterName"
Attributes               DESCRP="DESCRPValue"
                         TCPIP="TCPIP_Address"
                         FTMAC="FTMAC_Address"/>
```

| Attribute | Description |
|---|---|
| PrinterName | The name of the network printer to update. If this attribute is missing or blank, the most recently referenced printer will be used. If no printer had been previously referenced within the XML data, an error will result. The use of a query filter is not supported for this command. |
| DESCRPValue | Specifies the description text for this network printer. |
| TCPIP_Address | Specifies the TCP/IP address for this network printer. Care should be taken to ensure that a valid TCP/IP address is specified. |
| FTMAC_Address | Specifies the FTMAC address for this network printer. Care should be taken to ensure that a valid FTMAC address is specified, or an error will occur during hardware validation. Validation will catch duplicate and improperly formatted FTMAC addresses. |

### Examples:

Update the TCPIP address for network printer PRTNET **PR0001** to **151.128.152.002**:

```
<UpdateNetworkPrinterAttributes Printer="PR0001"
TCPIP="151.128.152.002"/>
```

Update the FTMAC address for network printer PRTNET **PR0001** to **C0000A**:

```
<UpdateNetworkPrinterAttributes Printer="PR0001"
FTMAC="C0000A"/>
```

# UpdateNodeAttributes Command

The **UpdateNodeAttributes** command allows you to update non-software attributes for a NODE. All attributes may be specified in a single command.

### Syntax:

```
<UpdateNodeAttributes      Node="NodeName"
                           DESCRP="Description"
                           NSAP ="NSAP_Address"/>
```

| Attribute | Description |
|---|---|
| NodeName | The name of the node to be updated.<br>If this attribute is missing or blank, the most recently referenced node will be used. If no node had been previously referenced within the XML data, an error will occur.<br>The use of a query filter is not supported for this command. |
| Description | Specifies description text for this node. |
| NSAP_Address | Specifies the NSAP for this NODE. Validation errors are thrown if an invalid NSAP Address is set. |

### Examples:

Update the NSAP Address for node **N00001** to **000002**:

```
<UpdateNodeAttributes Node="N00001" NSAP="1000002"/>
```

Update the NSAP Address for node **N00001** to **000002** and node **N00002** to **000002**:

```
<UpdateNodeAttributes Node="N00001" NSAP="000002"/>
<UpdateNodeAttributes Node="N00002" NSAP="000002"/>
<UpdateNodeAttributes Node="N00002" DESCRP="DESCRPValue"/>
```

**Note** This command fails displaying the following message “Updating NSAP=000002... Another object has claimed this NSAP value (000002)”.

# UpdateObjectAttribute Command

The **UpdateObjectAttribute** command allows you to update an attribute value on one or more objects – template or instance.

### Syntax:

```
<UpdateObjectAttribute  Object="ObjectName"
                        AttrName="AttrName"
                        AttrValue="AttrValue"/>
                        Or
<UpdateObjectAttribute  Filter="FilterName"
                        AttrName="AttrName"
                        AttrValue="AttrValue"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object to update. **ObjectName** may be a template or instance.<br>**ObjectName** should not be specified if **FilterName** is given. |
| AttrName | The name of the attribute to be updated in the object. If not found, an error will result. |
| AttrValue | The value that the attribute is to be updated to. |
| FilterName | The name of the filter that will be used to determine what objects get updated. **FilterName** should not be specified if **ObjectName** is given. |

### Examples:

Update the **PV.EngUnits** attribute on the **Boolean_001** object to **PSI**:

```
<UpdateObjectAttribute Object="Boolean_001"
AttrName="PV.EngUnits" AttrValue="PSI"/>
```

Update the **ExecutionRelativeOrder** attribute on the **Boolean_001** object to **Before**. For an enumerated value such as this attribute, the value specified must be a valid value for that enumeration, or an error will result.

```
<UpdateObjectAttribute Object="Boolean_001"
AttrName="ExecutionRelativeOrder" AttrValue="Before"/>
```

Update the **PV.EngUnits** attribute on all templates and instances based on **$Boolean:**

```
<QueryFilter Filter="UOAFilter" Condition="BasedOn"
Value="$Boolean" />

<UpdateObjectAttribute Filter="UOAFilter" AttrName="PV.EngUnits"
AttrValue="NewValue" />
```

# UpdateSecurityGroup Command

The **UpdateSecurityGroup** command allows you to update the security group for any object within the Galaxy – works with ArchestrA objects as well as any control object.

### Syntax:

```
<UpdateSecurityGroup    Name="ObjectName"
                        Group="SecurityGroup"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object to be placed into the security group indicated by **SecurityGroup**.<br>If the object name begins with a dollar sign (‘**$**’), it is assumed to be a template. May not refer to a base template.<br>If the object name contains a period (‘**.**’), it is assumed to be the hierarchical name of the object (for example, **COMPND_001.MyStrategy**). |
| SecurityGroup | The name of the security group in which to place the object.<br>**SecurityGroup** should be the name of an existing security group in the Galaxy (for example, Operators).<br>If the security group does not exist, then the object will not appear within the security group until the security group is created within the Galaxy using the **Galaxy > Configure > Security** menu pick. |

### Examples:

Update the security group for compound template **$MY_COMPND**:

```
<UpdateSecurityGroup Name="$MY_COMPND" Group="Operators"/>
```

Update the security group for the compound instance **COMPND_001**:

```
<UpdateSecurityGroup Name="COMPND_001 " Group="Operators"/>
```

**Note** The **UpdateSecurityGroup** command requires that the object for which the security group is changing be checked-in. In order to ensure that it is checked-in, bracket the command with the appropriate **Reset** command, for example:

```
<Reset Compound=""/>
<UpdateSecurityGroup Name="COMPND_001" Group="Operators"/>
<Reset Compound=""/>
```

# UpdateSoftwareAttribute Command

The **UpdateSoftwareAttribute** command allows you to update an attribute value in a software package.

### Syntax:

```
<UpdateSoftwareAttribute  Workstation="WorkstationName"
                          Controller="ControllerName"
                          Printer="PrinterName"
                          FBM="FBMName"
                          FCM="FCMName"
                          SoftwarePackage="SoftwarePackageName"
                          ParmName="ParmName"
                          ParmValue="ParmValue"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The name of the workstation on which the software package resides. |
| ControllerName | The name of the controller on which the software package resides. |
| FBMName | The name of the FBM on which the software package resides. |
| FCMName | The name of the FCM on which the software package resides. |
| ParmName | The name of the attribute to be updated in the software package. **ParmName** must be the name of a valid software package attribute. If not found, an error will result. |
| ParmValue | The value that the attribute is to be updated to. |
| PrinterName | The name of the network printer on which the software package resides. |
| SoftwarePackage Name | The name of the software package on the workstation or controller to be updated. |

**Note** Either **WorkstationName**, **ControllerName**, **FBMName**, **FCMName**, or **PrinterName** can be specified. If none are specified, the last referenced Workstation is assumed. If these attributes are omitted and no previous workstation is referenced, the command fails.

### Examples:

Update the parameter **IPCNAM** on the **ASMON7** software package on workstation **AW0001** to the value **SYSMO1**:

```
<UpdateSoftwareAttribute Workstation="AW0001"
SoftwarePackage="ASMON7" ParmName="IPCNAM"
ParmValue="SYSMO1"/>
```

Update the parameter **CPBPC** on the **OS1C70** software package on controller **CP0100** to the value **2**:

```
<UpdateSoftwareAttribute Controller="CP0100"
SoftwarePackage="OS1C70" ParmName="CPBPC" ParmValue="2"/>
```

Update the parameter **P1LN** on the NETPRT software package on PRTNET **PR0001** to the value **LP05**:

```
<UpdateSoftwareAttribute Printer="PR0001"
SoftwarePackage="NETPRT" ParmName="P1LN" ParmValue="LP05"/>
```

The list of appropriate software packages and allowable attribute names and values is shown on the following pages.

**Note** When providing attribute values for enumerations, you must specify the value to the left of the equals ("=") sign. For example, if specifying a BPC of 100 ms for a controller, you must specify CPBPC=1, not 100. When such a list is provided in the table(s) below, the default value will be highlighted.

Also, when providing attribute values for attributes that specify the letterbug or logical name of another object, that object must exist in the database, or the value will not be applied. For example, when specifying the WP message backup MSGB1 of an AW70, the value must specify the name of a WP logical name that has already been specified on an existing AW.

**Table 15-1. Attributes for OS7AW1 – AW70 Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| OS7AW1 (AW70) | MTK | Specifies whether or not this AW is a Master Timekeeper (**Y**/**N**) |
| OS7AW1 (AW70) | PBMTK | Specifies whether or not this AW is a backup MTK (**Y**/**N**) |
| OS7AW1 (AW70) | GPS | Specifies whether or not this AW has a GPS for time synch (**Y**/**N**) |
| OS7AW1 (AW70) | APLN | AP logical host name. Initialized to the name of the AW. |
| OS7AW1 (AW70) | APLHLB | Logical host letterbug. Initialized to the name of the AW. |
| OS7AW1 (AW70) | DATIM | Date format (**0**=US, **1**=European) |
| OS7AW1 (AW70) | MSGLN | WP Logical Name |
| OS7AW1 (AW70) | MSGB1 | WP message backup (must be existing WP logical name) |
| OS7AW1 (AW70) | PnB1 | Port **n** backup device (must be existing message destination) |
| OS7AW1 (AW70) | PnBAU | Port **n** baud rate (**1**=19200, **2**=9600, **3**=5600, **4**=4800, **5**=1200, **6**=600, **7**=300). "**n**" ranges from 1 to 6. |

**Table 15-1. Attributes for OS7AW1 – AW70 Operating System (Continued)**

| Software Package | Attribute | Description |
|---|---|---|
| OS7AW1 (AW70) | PnLN | Port **n** Logical Name |
| OS7AW1 (AW70) | PP1LN | Parallel Port Logical Name |

**Table 15-2. Attributes for OS1C80 – FCP280 Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| OS1C80 (FCP280) | CPBPC | Basic process cycle (0.5=50 ms, 1=100 ms, **2**=200 ms, 5=500 ms, 10=1 sec) |
| OS1C80 (FCP280) | OMSCAN | Scan rate for OM (1=100 ms, **5**=500 ms) |
| OS1C80 (FCP280) | MMHLN0 | Logical name of primary destination message manager (must be existing Message Manager logical name) |
| OS1C80 (FCP280) | MMHLN1 | Logical name of backup destination message manager (must be existing Message Manager logical name) |

**Table 15-3. Attributes for OS1C70 – FCP270 Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| OS1C70 (FCP270) | CPBPC | Basic process cycle (0.5=50 ms, 1=100 ms, **2**=200 ms, 5=500 ms, 10=1 sec) |
| OS1C70 (FCP270) | OMSCAN | Scan rate for OM (1=100 ms, **5**=500 ms) |
| OS1C70 (FCP270) | MMHLN0 | Logical name of primary destination message manager (must be existing Message Manager logical name) |
| OS1C70 (FCP270) | MMHLN1 | Logical name of backup destination message manager (must be existing Message Manager logical name) |

**Table 15-4. Attributes for OS1Z70 – ZCP270 Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| OS1Z70 (ZCP270) | CPBPC | Basic process cycle (**0.5**=50 ms, **1**=100 ms, **2**=200 ms, **5**=500 ms, **10**=1 sec) |
| OS1Z70 (ZCP270) | OMSCAN | Scan rate for OM (**1**=100 ms, **5**=500 ms) |

**Table 15-4. Attributes for OS1Z70 – ZCP270 Operating System (Continued)**

| | | |
|---|---|---|
| OS1Z70 (ZCP270) | MMHLN0 | Logical name of primary destination message manager (must be existing Message Manager logical name) |
| OS1Z70 (ZCP270) | MMHLN1 | Logical name of backup destination message manager (must be existing Message Manager logical name) |

**Table 15-5. Attributes for AADM7 – Additional Display Managers**

| Software Package | Attribute | Description |
|---|---|---|
| AADM7 (AW70) | NDMRW | Number of additional display managers (**01-32**). Note that values less than 10 must be preceded by zero (for example, 01, 02, and so forth.) |

**Table 15-6. Attributes for AHIST7 – AIM*Historian**

| Software Package | Attribute | Description |
|---|---|---|
| AHIST7 (AW70) | HSNAME | Historian name – must be lowercase, 6 chars (for example, **hist01**) |

**Table 15-7. Attributes for AMSGM7 – Message Manager**

| Software Package | Attribute | Description |
|---|---|---|
| AMSGM7 (AW70) | MMLN | Message manager logical name. Initialized to the name of the AW. |
| AMSGM7 (AW70) | REDUNT | Indicates whether message manager is redundant (**Y**/**N**) |
| AMSGM7 (AW70) | BKPLN | Backup message manager logical name (must be the logical name of an existing Message Manager). Only specified in redundant. |
| AMSGM7 (AW70) | MDSLN | Array of controller letterbugs (FCP270s or ZCP270s) that messages go to from this Message Manager.<br>Must be comma-separated list of letterbugs, up to 256 existing FCPs or ZCPs (for example, “CP0100,CP0200,CP0300”).<br>The same letterbug cannot appear in the list of more than one Message Manager. |

**Table 15-8. Attributes for ASMON7 – System Monitor**

| Software Package | Attribute | Description |
|---|---|---|
| ASMON7 (AW70) | IPCNAM | System Monitor name – 6 characters |
| ASMON7 (AW70) | SMHDB | Name of associated Historian database. Must be name of an instance of AHIST7. If no AIM*Historian database is configured, this should not be specified. |
| ASMON7 (AW70) | PR1PR0 | Priority 1 Printer 0 (must be existing port logical name) |
| ASMON7 (AW70) | PR1PR1 | Priority 1 Printer 1 (must be existing port logical name) |
| ASMON7 (AW70) | PR2PR0 | Priority 2 Printer 0 (must be existing port logical name) |
| ASMON7 (AW70) | PR2PR1 | Priority 2 Printer 1 (must be existing port logical name) |
| ASMON7 (AW70) | SMGC | System Monitor WPs.<br>Must be comma-separated list of up to 6 existing WP logical names (for example, “AW0001,AW0002”) |
| ASMON7 (AW70) | SMNWP | Notification WPs<br>Must be comma-separated list of up to 6 existing WP logical names (for example, “AW0001,AW0002”) |
| ASMON7 (AW70) | SMSTM | Stations to monitor.<br>Must be comma-separated list of up to 64 existing stations and switches to be monitored by this system monitor (for example, “CP0100,SW0001,CP0200”) |

**Table 15-9. Attributes for OS7PC1 - WSTA70 and WSVR70 Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| OS7PC1 (WSTA70, WSVR70) | NICTYP | Specifies the type of network card for the station (1C=Single Copper, 2C=Dual Copper, 1F=Single Fiber, 2F=Dual Fiber) |
| OS7PC1 (WSTA70, WSVR70) | MTK | Specifies whether this workstation is a master timekeeper (Y,N) |

**Table 15-9. Attributes for OS7PC1 - WSTA70 and WSVR70 Operating System (Continued)**

| Software Package | Attribute | Description |
|---|---|---|
| OS7PC1 (WSTA70, WSVR70) | PBMTK | Specifies whether his workstation is a backup MTK (Y/N) |
| OS7PC1 (WSTA70, WSVR70) | GPS | Specifies whether this workstation has a GPS for time synch (Y/N) |
| OS7PC1 (WSTA70, WSVR70) | DATIM | Date format (0=US, 1=European, 3-ISO) |
| OS7PC1 (WSTA70, WSVR70 | P1LN | USBPrinter1 logical name) |
| OS7PC1 (WSTA70, WSVR70) | P1B1 | USBPrinter1 backup logical name |
| OS7PC1 (WSTA70, WSVR70) | P2LN | USBPrinter2 logical name |
| OS7PC1 (WSTA70, WSVR70) | P2B1 | USBPrinter2 backup logical name |
| OS7PC1 (WSTA70, WSVR70) | P3LN | USBPrinter3 logical name |
| OS7PC1 (WSTA70, WSVR70) | P3B1 | USBPrinter3 backup logical name |
| OS7PC1 (WSTA70, WSVR70) | APLN | AP logical host name. Initialized to the name of the workstation. |
| OS7PC1 (WSTA70, WSVR70) | APLHLB | Logical host letterbug. Initialized to the name of the workstation. |
| OS7PC1 (WSTA70, WSVR70) | MSGLN | WP logical name |
| OS7PC1 (WSTA70, WSVR70) | MSGB1 | WP message backup (must be existing WP logical name) |

**Table 15-10. Attributes for NETPRT - PRTNET Operating System**

| Software Package | Attribute | Description |
|---|---|---|
| NETPRT (PRTNET) | P1LN | Network printer logical name |
| NETPRT (PRTNET) | P1B1 | Network printer backup logical name |

# UpdateStrategyDiagram Command

The **UpdateStrategyDiagram** command provides a mechanism to ensure that the Visio diagram that represents a strategy in the Galaxy is up-to-date. The UpdateStrategyDiagram creates a strategy diagram for a given strategy or updates any existing diagram that is not up-to-date.

**Syntax:**

```
<UpdateStrategyDiagram  Strategy="StrategyName"
                        SwitchAppearanceObjects="SwitchAO"
                        LayoutDiagram="Layout"
                        PreserveUserObjects="PreserveUserObjs"
                        PrintTemplate="PrintTemplateName"
                        ApplyEditorPreferences="ApplyEditorPreferences"
                        UpgradeToShowPageBreaks="UpgradeToShowPageBreaks"
                        ManageLayout="ManageLayout"
                        Filter="FilterName"
                        Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy whose associated Visio diagram is to be updated. attribute should not be specified if FilterName is given. |

| Attribute | Description |
|---|---|
| SwitchAO | This attribute allows the user to change Block Appearance Objects on the strategy based on information found in the IDE Tools/Editor Preferences Dialog. Valid values for SwitchAO are **Yes** and **No**. The default value is **No**.<br>When the value of SwitchAO is **Yes** and 'Use Logic Editor Appearance Objects' is checked in the IDETools/Editor Preferences dialog, the Appearance Object (AO) for a given block is switched to a logic diagram for block types that support logic diagrams and have defined RPN code. These block types are the CALC, CALCA, LOGIC, and MATH blocks. The x, y location of each AO on the strategy canvas will be retained. These block types do not have factory-supplied definitions in the IDE Dynamic Appearance Object Definitions Editor, but if a user-defined DAO definition exists and 'Use Dynamic Appearance Objects' is also checked in the Editor Preferences dialog, the Logic diagram AO will take precedence over the DAO. When 'Use Logic Editor Appearance Objects' is not checked in the Editor Preferences dialog, any Logic AO is switched back to a user-defined AO, if one exists, otherwise to the default 'green' AO.<br>The Editor Preferences Dialog also contains a check box labeled 'Show Logic Blocks in Color', to show logic shapes on the logic diagram filled either with color (checked) or white (unchecked). The current value of this setting is applied whenever a Logic Editor Appearance Object is updated, even if that AO already exists on the strategy.<br>When the value of the SwitchAO attribute is **Yes**, and 'Use Dynamic Appearance Objects' is checked in the Editor Preferences dialog, and the block type for a given block on the strategy is checked in the IDE Dynamic Appearance Object Definitions Editor, the AO for that block is switched to the corresponding Dynamic Appearance Object (DAO) if it is not already that DAO.<br>If the block type is not checked in the Dynamic Appearance Object Definitions Editor and the DAO currently exists on the strategy, it is switched back to a user-defined AO, if one exists, otherwise to the default 'green' AO. When 'Use Dynamic Appearance Objects' is not checked in the Editor Preferences dialog, any DAO is switched back to a user-defined AO, if one exists, otherwise to the default 'green' AO.<br>When the value of SwitchAO is **No** or is not specified, no changes are made to Block Appearance Objects on the strategy.<br><br>**Note** Whenever the value of SwitchAO is **Yes**, it is recommended to also specify Layout='LayoutBlocksAndLines', otherwise, errors reporting overlapping shapes could occur. |

| Attribute | Description |
|---|---|
| Layout | This attribute provides the user with layout options for a strategy. Valid options are:<br>• **LayoutNone**: No layout changes are made for the strategy.<br>• **LayoutLinesOnly**: Blocks are left in their original locations and lines on the strategy are re-routed.<br>• **LayoutBlocksAndLines**: Blocks on the strategy are re-positioned and lines are re-routed.<br><br>Non-control objects are not considered during layout and are left in their original locations relative to each other, but could be located differently relative to control objects.<br>There are cases when a Strategy Diagram does not yet exist. For instance after a strategy is generated from Direct Access or Bulk Data and has not yet been opened in the IDE Strategy Editor, or if the strategy is generated as above and then exported. When this export file is reimported, no Strategy Diagram will exist.<br>In these cases the diagram is created rather than updated, and blocks and lines must be laid out for the first time, ignoring the user's layout choice.<br>When the **LayoutLinesOnly** option is selected, errors can be returned to the user if the blocks are not already properly laid out (for example, if any blocks overlap each other).<br>The user's layout choice is executed whether or not Use Dynamic Appearance Objects is checked in the IDE Tools / Global Editor Preferences dialog. This allows the user to re-layout existing strategies.<br>The default value for this attribute is **LayoutNone**.<br><br>**Note** The Layout argument can change the ManageLayout setting for the strategy. When the Layout command is run with LayoutBlocksAndLines, the ManageLayout setting is left at **No**, regardless of what the setting was before the Layout command was run.<br>The user may want to include a command to set the ManageLayout option to **Yes** after running a LayoutBlocksAndLines command. |

| Attribute | Description |
|---|---|
| PreserveUser Objs | The user can choose whether to keep all non-control objects as they appear on the strategy, or remove them.<br>When the value of this attribute is **Yes**, all non-control objects are left undisturbed and in their original locations relative to each other, but could be located differently relative to control objects.<br>When the value of this attribute is **No**, all non-control objects are deleted from the strategy.<br>The PreserveUserObjects option is executed whether or not Use Dynamic Appearance Objects is selected in the Global Editor Preferences dialog of the IDE.<br>The default value for this attribute is **Yes**. |
| PrintTemplate Name | The name of the print template to use when updating a strategy. The ".vsd" extension will automatically be added if not provided within the command.<br>**Note** Supplying the appropriate value for PrintTemplateName is important because the Print Template is used during strategy diagram 'LayoutBlocksAndLines' to place shapes on each page so they will not overlap page boundaries.<br>**PrintTemplateName** value may be either the name of a standard print template, or a print template that has been customized by the user. Standard print templates include:<br>• IATemplate_Landscape_Ledger<br>• IATemplate_Landscape_Legal<br>• IATemplate_Landscape_Letter<br>• IATemplate_Portrait_Legal<br>• IATemplate_Portrait_Letter<br>• IATemplate_Portrait_Tabloid<br>If blank or missing, the print template will default to **IATemplate_Portrait_Letter.vsd**. |
| ApplyEditor Preferences | Specifies whether or not to apply line connection configuration changes, done in IDE Tools/Editor Preferences Dialog, to connection lines within the diagram. Valid values for ApplyEditorPreferences are Yes and No. The default value is No. Line connectors within Logic Editor Appearance Objects will also be updated. |

| Attribute | Description |
|---|---|
| UpgradeToShow PageBreaks | The page break and high resolution printout features have been disabled for strategy diagrams that were engineered and laid out prior to Control Editors v5.0.1, due to diagram resizing needed in support of showing the page break feature. In order to upgrade pre-existing diagrams to use these features, set the value of UpgradeToShowPageBreaks to “Yes”. Valid values for UpgradeToShowPageBreaks are “Yes” and “No”. The default value is “No”. Once the diagram has been updated, some minor layout modifications to adjust the print layout may need to be performed by the user since the diagram has been resized. Strategy Diagrams created with Control Editors v5.0.1, or later, automatically support page breaks and high resolution printouts. Setting argument UpgradeToShowPageBreaks to No will have no effect and will not remove these features or revert back an upgraded strategy diagram. |
| ManageLayout | Specifies whether the layout is managed by the strategy or user. If layout is managed by the strategy, a one-inch border is imposed between the page boundary and the appearance objects on the page allowing for routing of lines and positioning of declarations. If layout is managed by the user, the one-inch border is not enforced. Valid values for ManageLayout are **Yes**, **No**, and **Ignore**. If the user wants to manage the layout, set ManageLayout to **Yes**, otherwise set the value to **No**. If ManageLayout is set to **Ignore**, the argument is not processed by UpdateStrategyDiagram command. |
| FilterName | The name of the filter that is used to determine which strategy diagrams are updated. This attribute should not be specified if StrategyName is given. |
| CascadeValue | The CascadeValue specifies whether a cascade update is to be performed or not.<br>If CascadeValue is **Yes**, the diagrams of all contained strategies are updated along with the strategies specified by the command. |

### Examples:

Update Strategy “SELEX01”, switching Appearance Objects and deleting all user-defined objects. Line connection configuration will be applied to connection lines. All blocks and lines will be re-laid-out. If the Strategy Diagram does not yet exist, it will be generated:

```
<UpdateStrategyDiagram Strategy="SELEX01"
SwitchAppearanceObjects="Yes"
LayoutDiagram="LayoutBlocksandLines"
PrintTemplate="IATemplate_Landscape_Letter.vsd"
PreserveUserObjects="No"
ApplyEditorPreferences="Yes"
UpgradeToShowPageBreaks="Yes
ManageLayout="Yes/>
```

**Note** For more information on the Logic Editor Appearance Objects discussed in SwitchAO above, refer to “Dynamic Appearance Objects” in *Configuration Utilities User's Guide* (B0750AZ).

Update a Strategy Template and its contained strategies:

```
<UpdateStrategyDiagram Strategy="$MyTemplate1" Cascade="Yes"/>
```

Update all strategies assigned to a compound:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"/>
<QueryFilter Filter="Filter1" Condition="ContainedBy"Value="
COMPND_002"/>
<UpdateStrategyDiagram Filter="Filter1"/>
```

Below is a table that lists combinations of command option choices and their results. In the results column a ✓ indicates the result is applicable to corresponding command options, an ✗ indicates the result is not applicable.

This table assumes that a Strategy Diagram exists, Use Dynamic Appearance Objects is checked in the IDE Tools / Global Editor Preferences dialog, and block types for all blocks on the strategy are checked in the IDE Dynamic Appearance Object Definitions Editor.

| Command Options | | | Results | | | |
|---|---|---|---|---|---|---|
| Switch | Layout | Preserve User Objs | AOs Switched | Layout Lines Only | Layout Lines & Blocks | User Objs Preserved |
| Yes | None | Yes | ✓ | ✗ | ✗ | ✓ |
| Yes | None | No | ✓ | ✗ | ✗ | ✗ |
| Yes | Lines Only | Yes | ✓ | ✓ | ✗ | ✓ |
| Yes | Lines Only | No | ✓ | ✓ | ✗ | ✗ |
| Yes | Blocks and Lines | Yes | ✓ | ✗ | ✓ | ✓ |
| Yes | Blocks and Lines | No | ✓ | ✗ | ✓ | ✗ |
| No | None | Yes | ✗ | ✗ | ✗ | ✓ |
| No | None | No | ✗ | ✗ | ✗ | ✗ |
| No | Lines Only | Yes | ✗ | ✓ | ✗ | ✓ |
| No | Lines Only | No | ✗ | ✓ | ✗ | ✗ |
| No | Blocks and Lines | Yes | ✗ | ✗ | ✓ | ✓ |
| No | Blocks and Lines | No | ✗ | ✗ | ✓ | ✗ |

# UpdateSwitchAttributes Command

The **UpdateSwitchAttributes** command allows you to update non-software attributes for a switch. Any or all attributes may be specified in a single command.

### Syntax:

```
<UpdateSwitchAttributes    Switch="SwitchName"
                           TCPIP="TCPIP_Address"
                           Ports="Number_Ports"/>
```

| Attribute | Description |
|---|---|
| SwitchName | The name of the switch to be updated.<br>If this attribute is missing or blank, the most recently referenced switch will be used. If no switch had been previously referenced within the XML data, an error will result.<br>The use of a query filter is not supported for this command. |
| TCPIP_Address | Specifies the TCP/IP address for this switch. Care should be taken to ensure that a valid TCP/IP address is specified, or an error will result during hardware validation. Validation will catch duplicate and improperly formatted TCP/IP addresses. |
| Number_Ports | Specifies the number of ports for the switch.<br>This is always a safe operation when increasing the number of ports (for example, 8 to 16). However, if decreasing the number of ports, hardware previously connected to ports outside the new limit will be disconnected from the switch. |

### Examples:

Update the TCP/IP address for switch **SW0001** to **151.128.081.001** and specify the number of ports to be 16:

```
<UpdateSwitchAttributes Switch="SW0001" TCPIP="151.128.081.001"
Ports="16"/>
```

# UpdateUserAttribute Command

The **UpdateUserAttribute** command allows you to update a user attribute within a strategy.

### Syntax:

```
<UpdateUserAttribute    Attribute="AttributeName"
                        Value="AttributeValue"
                        Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| AttributeName | The name of the user attribute to update within the strategy. |
| AttributeValue | The new value to assign to the user attribute |
| StrategyName | The name of the strategy this user attribute is to be updated in. May be the name of an existing template (for example, **$My_Strategy**) or instance.<br>If **StrategyName** is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

### Examples:

Update a user attribute called **Project** on strategy **Strategy_001**:

```
<UpdateUserAttribute Attribute="Project" Value="New_Refinery"
Strategy="Strategy_001"/>
```

# UpdateWorkstationAttributes Command

The **UpdateWorkstationAttributes** command allows you to update non-software attributes for a workstation. Any or all attributes may be specified in a single command.

### Syntax:

```
<UpdateWorkstationAttributes   Workstation="WorkstationName"
                               DESCRP="DESCRPValue"
                               TCPIP="TCPIP_Address"
                               TCPIP2="TCPIP2_Address"
                               FT="FTValue"
                               FTMAC="FTMAC_Address"/>
```

| Attribute | Description |
|---|---|
| WorkstationName | The name of the workstation to be updated.<br>If this attribute is missing or blank, the most recently referenced workstation will be used. If no workstation had been previously referenced within the XML data, an error will result.<br>The use of a query filter is not supported for this command. |
| DESCRPValue | Specifies description text for this workstation. |
| TCPIP_Address | Specifies the TCP/IP address for this workstation. Care should be taken to ensure that a valid TCP/IP address is specified, or an error will result during hardware validation.<br>Validation will catch duplicate and improperly formatted TCP/IP addresses. |
| TCPIP2_Address | Specifies the TCP/IP2 address for this workstation (ATS). Care should be taken to ensure that a valid TCP/IP2 address is specified, or an error will occur during hardware validation.<br>Validation will catch duplicate and improperly formatted TCP/IP addresses. |
| FTMAC_Address | Specifies the FTMAC address for this workstation. Care should be taken to ensure that a valid FTMAC address is specified, or an error will occur during hardware validation.<br>Validation will catch duplicate and improperly formatted FTMAC addresses. |
| FT | Specifies the FT type for this workstation. The two values associated with FT are Single and Fault Tolerant. |

**Examples:**

Update the TCPIP address for workstation **AW0001** to **151.128.152.002**:

```
<UpdateWorkstationAttributes Workstation="AW0001"
TCPIP="151.128.152.002"/>
```

Update the TCPIP2 address for workstation **ATS001** to **151.128.008.066**:

```
<UpdateWorkstationAttributes Workstation="ATS001"
TCPIP2="151.128.008.066"/>
```

Update the FTMAC address for workstation **AW0001** to **C0000A**:

```
<UpdateWorkstationAttributes Workstation="AW0001"
FTMAC="C0000A"/>
```

CHAPTER 16

# Lock/Unlock Operations

This chapter describes lock and unlock operations, which allow you to lock/unlock an attribute on a block template or instance on a strategy template, lock/unlock an attribute on one or more compound templates, and lock/unlock a user attribute within a strategy template.

## Contents



# LockBlockAttribute Command

The **LockBlockAttribute** command allows you to lock an attribute on a block template or instance on a strategy template. If the attribute that is locked is on a block template, then the attribute becomes “locked-in-parent” for all instances of that block, no matter where they are. If the attribute that is locked is on a block instance contained within a strategy template, then the attribute becomes “locked-in-parent” for that block on all instances and templates derived from that strategy template.

**Syntax:**

```
<LockBlockAttribute    Strategy="StrategyName"
                       Block="BlockName"
                       ParmName="ParmName"
                       TypeName="TypeName"/>
```

Or

```
<LockBlockAttribute    Filter="FilterName"
                       ParmName="ParmName"
                       Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. This attribute must be a strategy template – locking of block attributes in a strategy instance is not supported.<br>If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file.<br>**StrategyName** is not required if **BlockName** is a block template. |
| BlockName | The name of the block in which the attribute exists.<br>**BlockName** may specify either a block template, or a block in a strategy template.<br>If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute to be locked in the block.<br>**ParmName** must be the name of a valid control block attribute. If not found, an error will result.<br>The attribute must not be “locked in parent” or a warning will result. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName** – not valid if a **FilterName** is specified.<br>**TypeName** defaults to **ContainedName** if not specified and no filter has been specified.<br>**Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block.<br>This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br><br>**Note** If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either block templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates.<br>The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName**. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

### Examples:

Lock the **HSCO1** attribute on the **MY_AIN** block within strategy **$MyStrategy1**:

```
<LockBlockAttribute Strategy="$MyStrategy1" Block="MY_AIN"
ParmName="HSCO1"/>
```

Lock the **HSCO1** attribute on the derived block template **$MY_AIN**:

```
<LockBlockAttribute Block="$MY_AIN" ParmName="HSCO1"/>
```

Lock the **HSCO1** attribute on all blocks derived from **$AIN** that are found in **$Strategy_002**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="$Strategy_002"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<LockBlockAttribute Filter="Filter1" ParmName="HSCO1"/>
```

Lock the **HSCO1** attribute on all blocks derived from **$AIN** that are found on any strategy templates:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"
Type="Templates" />

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<LockBlockAttribute Filter="Filter1" ParmName="HSCO1"
Cascade="Yes"/>
```

# LockCompoundAttribute Command

The **LockCompoundAttribute** command allows you to lock an attribute on one or more compound templates. Once locked, the attribute becomes “locked-in-parent” for that compound on all instances and templates derived from that compound template.

### Syntax:

```
<LockCompoundAttribute   Compound="CompoundName"
                         ParmName="ParmName"/>
                         Or
<LockCompoundAttribute   Filter="FilterName"
                         ParmName="ParmName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound. This attribute must be a compound template – locking of attributes in a compound instance is not supported.<br>This attribute should not be specified if **FilterName** is given. |
| ParmName | The name of the attribute to be locked in the compound. **ParmName** must be the name of a valid control attribute. If not found, an error will result.<br>The parameter must not be “Locked in Parent”, or a warning will result. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. **FilterName** should not be specified if **CompoundName** is given. |

### Examples:

Lock the **PHASE** attribute on the **$CMPND_001** compound template. Locking this attribute will cause all **PHASE** attributes to be marked “locked-in-parent” on all instances of **$CMPND_001**:

```
<LockCompoundAttribute Compound="$CMPND_001"
ParmName="PHASE"/>
```

Lock the **PHASE** attribute on all compound templates beginning with **"$COMP".** Locking this attribute will cause all **PHASE** attributes to be marked "locked-in-parent" on all instances of these compounds.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="$COMP%"/>

<LockCompoundAttribute Filter="Filter1" ParmName="PHASE"/>
```

# LockDeclaration Command

The **LockDeclaration** command allows you to lock a declaration within a strategy template.

### Syntax:

```
<LockDeclaration      Decl="DeclarationName"
                      Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| DeclarationName | The name of the declaration to lock within the strategy template. |
| StrategyName | The name of the strategy in which this declaration is to be locked. **StrategyName** must be the name of an existing template (for example, **$My_Strategy**). If the strategy specified is not a template, an error will result.<br>If this attribute is missing, or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

### Examples:

Lock a declaration called **Project** on strategy **$Strategy_001**:

```
<LockDeclaration Decl="Project" Strategy="$Strategy_001"/>
```

# LockObjectAttribute Command

The **LockObjectAttribute** command allows you to lock an attribute on one or more object templates. Once locked, the attribute becomes "locked-in-parent" for that object template on all instances and templates derived from that template.

### Syntax:

```
<LockObjectAttribute   Object="ObjectName"
                       AttrName="AttrName"/>
```

Or

```
<LockObjectAttribute  Filter="FilterName"
                      AttrName="AttrName"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object. This attribute must be a template – locking of attributes in an object instance is not supported. This attribute should not be specified if **FilterName** is given. |
| AttrName | The name of the attribute to be locked in the object. If not found, an error will result.<br>The parameter must not be “Locked in Parent”, or a warning will result. |
| FilterName | The name of the filter that will be used to determine what objects get updated. **FilterName** should not be specified if **ObjectName** is given. |

### Examples:

Lock the **PV.EngUnits** attribute of the template **$Boolean_001**. Locking this attribute will cause all **PV.EngUnits** attributes to be marked “locked-in-parent” on all instances of **$Boolean_001**:

```
<LockObjectAttribute Object="$Boolean_001"
AttrName="PV.EngUnits"/>
```

Lock the **PV.EngUnits** attribute of all templates that are based on **$Boolean**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Boolean"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%"/>
<LockObjectAttribute Filter="Filter1" AttrName="PV.EngUnits"/>
```

# LockUserAttribute Command

The **LockUserAttribute** command allows you to lock a user attribute within a strategy template.

### Syntax:

```
<LockUserAttribute  Attribute="AttributeName"
                    Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| AttributeName | The name of the user attribute to lock within the strategy template. |
| StrategyName | The name of the strategy this user attribute is to be locked in. **StrategyName** must be the name of an existing template (for example, **$My_Strategy**).<br>If the strategy specified is not a template, a warning will result.<br>If this attribute is missing, or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

**Examples:**

Lock a user attribute called **Project** on strategy $**Strategy_001**:

```
<LockUserAttribute Attribute="Project" Strategy="$Strategy_001"/>
```

# UnlockBlockAttribute Command

The **UnlockBlockAttribute** command allows you to unlock an attribute on a block template or instance on a strategy template. If the attribute that is being unlocked is on a block template, then the attribute becomes unlocked for all instances of that block, no matter where they are. If the attribute that is being unlocked is on a block instance contained within a strategy template, then the attribute becomes unlocked for that block on all instances and templates derived from that strategy template.

**Syntax:**

```
<UnlockBlockAttribute    Strategy="StrategyName"
                         Block="BlockName"
                         ParmName="ParmName"
                         TypeName="TypeName"/>
                         Or
<UnlockBlockAttribute    Filter="FilterName"
                         ParmName="ParmName"
                         Cascade="CascadeValue"/>
```

| Attribute | Description |
|---|---|
| StrategyName | The name of the strategy in which the block resides. This attribute must be a strategy template – unlocking of block attributes in a strategy instance is not supported.<br>If **StrategyName** is empty or blank, and no filter is specified, then it will default to the name of the most recently referenced strategy within the command file.<br>**StrategyName** is not required if **BlockName** is a block template. |
| BlockName | The name of the block in which the attribute exists. May specify either a block template, or a block in a strategy template.<br>If **BlockName** is not specified or blank, then **FilterName** must be specified. |
| ParmName | The name of the attribute to be unlocked in the block. **ParmName** must be the name of a valid control block attribute. If not found, an error will result.<br>The attribute must not be “locked in parent” or a warning will result. |
| TypeName | The type of name specified by **BlockName**. Values may be **Tagname** or **ContainedName** – not valid if a **FilterName** is specified.<br>**TypeName** defaults to **ContainedName** if not specified and no filter has been specified.<br>**Tagname** signifies the name by which the block is known to the Control Processor. **ContainedName** is the name by which the block is known to the strategy. The tagname of a block does not have to be the same as the contained name of a block.<br>This setting is ignored if **FilterName** is specified. |

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what blocks in the Galaxy are to be considered for this operation.<br>If **FilterName** is specified, then **StrategyName** and **BlockName** should not be.<br><br>**Note** If a filter is composed of only a **QueryFilter** specification, then the objects returned by the filter must be either block templates or strategies.<br><br>If the filter is a combination of a **QueryFilter** and **BlockQueryFilter**, then the objects returned by the filter must be either block or strategy templates.<br>The specification of **AttributeQueryFilter** for purposes of this command will be ignored, because of the dependency upon **ParmName**. |
| CascadeValue | Specifies whether or not contained or assigned objects are to be considered. Valid only when **FilterName** is specified.<br>Valid values are **Yes** or **No**. If not specified, default value is **No**.<br>If **CascadeValue** = **Yes**, then the operation will also consider all objects that are contained by, or assigned to, any of the objects that were immediately returned by the query filter. |

### Examples:

Unlock the **HSCO1** attribute on the **MY_AIN** block within strategy **$MyStrategy1**:

```
<UnlockBlockAttribute Strategy="$MyStrategy1" Block="MY_AIN"
ParmName="HSCO1"/>
```

Unlock the **HSCO1** attribute on the derived block template **$MY_AIN**:

```
<UnlockBlockAttribute Block="$MY_AIN" ParmName="HSCO1"/>
```

Unlock the **HSCO1** attribute on all blocks derived from **$AIN** that are found in **$Strategy_002** and any child strategies that might be contained by **$Strategy_002**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="$Strategy_002"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UnlockBlockAttribute Filter="Filter1" ParmName="HSCO1"
Cascade="Yes”/>
```

Unlock the **HSCO1** attribute on all blocks derived from **$AIN** that are found on any strategy templates:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Strategy"
Type="Templates" />
```

```
<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<UnlockBlockAttribute Filter="Filter1" ParmName="HSCO1"
Cascade="Yes"/>
```

# UnlockCompoundAttribute Command

The **UnlockCompoundAttribute** command allows you to unlock an attribute on one or more compound templates. Once unlocked, the attribute becomes unlocked for that compound on all instances and templates derived from that compound template.

### Syntax:

```
<UnlockCompoundAttribute  Compound="CompoundName"
                          ParmName="ParmName"/>
                          Or
<UnlockCompoundAttribute  Filter="FilterName"
                          ParmName="ParmName"/>
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound. **CompoundName** must be a compound template – unlocking attributes in a compound instance is not supported.<br>**CompoundName** should not be specified if **FilterName** is given. |
| ParmName | The name of the attribute to be unlocked in the compound. **ParmName** must be the name of a valid control attribute. If not found, an error will result.<br>The attribute must not be “Locked in Parent” or a warning will result. |
| FilterName | The name of the filter that will be used to determine what compounds get updated. **FilterName** should not be specified if **CompoundName** is given. |

### Examples:

Unlock the **PHASE** attribute on the **$CMPND_001** compound template. Unlocking this attribute will cause all **PHASE** attributes to be marked “unlocked” on all instances and derived compound templates of **$CMPND_001**:

```
<UnlockCompoundAttribute Compound="$CMPND_001"
ParmName="PHASE"/>
```

Unlock the **PHASE** attribute on all compound templates beginning with **$COMP**. Unlocking this attribute will cause all **PHASE** attributes to be marked “unlocked” on all instances and derived compound templates of these compounds.

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<QueryFilter Filter="Filter1" Condition="NamedLike"
Value="$COMP%"/>

<UnlockCompoundAttribute Filter="Filter1" ParmName="PHASE"/>
```

# UnlockDeclaration Command

The **UnlockDeclaration** command allows you to unlock a declaration within a strategy template.

### Syntax:

```
<UnlockDeclaration    Decl="DeclarationName"
                      Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| DeclarationName | The name of the declaration to unlock within the strategy template. |
| StrategyName | The name of the strategy this declaration is to be unlocked in. **StrategyName** must be the name of an existing template (for example, **$My_Strategy**). If the strategy specified is not a template, an error will result.<br>If this attribute is missing, or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

### Examples:

Unlock a declaration called **Project** on strategy **$Strategy_001**:

```
<UnlockDeclaration Decl="Project" Strategy="$Strategy_001"/>
```

# UnlockObjectAttribute Command

The **UnlockObjectAttribute** command allows you to unlock an attribute on one or more object templates. Once unlocked, the attribute becomes unlocked for that object on all instances and templates derived from that template.

### Syntax:

```
<UnlockObjectAttribute   Object="ObjectName"
                         AttrName="AttrName"/>
                         Or
<UnlockObjectAttribute   Filter="FilterName"
                         AttrName="AttrName"/>
```

| Attribute | Description |
|---|---|
| ObjectName | The name of the object. **ObjectName** must be a template – unlocking attributes in an object instance is not supported. **ObjectName** should not be specified if **FilterName** is given. |
| AttrName | The name of the attribute to be unlocked in the object. If not found, an error will result.<br>The attribute must not be "Locked in Parent" or a warning will result. |
| FilterName | The name of the filter that will be used to determine what objects get updated. **FilterName** should not be specified if **ObjectName** is given. |

### Examples:

Unlock the **PV.EngUnits** attribute on the **$Boolean_001** object template. Unlocking this attribute will cause all **PV.EngUnits** attributes to be marked "unlocked" on all instances of **$Boolean_001**:

```
<UnlockObjectAttribute Object="$Boolean_001"
AttrName="PV.EngUnits"/>
```

Unlock the **PV.EngUnits** attribute of all templates that are based on **$Boolean**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$Boolean"/>
<QueryFilter Filter="Filter1" Condition="NamedLike" Value="%"/>
<UnlockObjectAttribute Filter="Filter1" AttrName="PV.EngUnits"/>
```

# UnlockUserAttribute Command

The **UnlockUserAttribute** command allows you to unlock a user attribute within a strategy template.

### Syntax:

```
<UnlockUserAttribute    Attribute="AttributeName"
                        Strategy="StrategyName"/>
```

| Attribute | Description |
|---|---|
| AttributeName | The name of the user attribute to unlock within the strategy template.<br>**AttributeName** must be the name of a user attribute that is not already locked in parent. |
| StrategyName | The name of the strategy in which this user attribute is to be unlocked. **StrategyName** must be the name of an existing template (for example, **$My_Strategy**).<br>If the strategy specified is not a template, a warning will result.<br>If this attribute is missing or blank, the most recently referenced strategy will be used. If no strategy had been previously referenced, an error will result. |

### Examples:

Unlock a user attribute called **Project** on strategy **$Strategy_001**:

```
<UnlockUserAttribute Attribute="Project" Strategy="$Strategy_001"/>
```

CHAPTER 17

# Direct Access Support for Field Device Tool (FDT)

This chapter gives an introduction to the Direct Access Support for Field Device Tool (FDT) functions.

## Contents

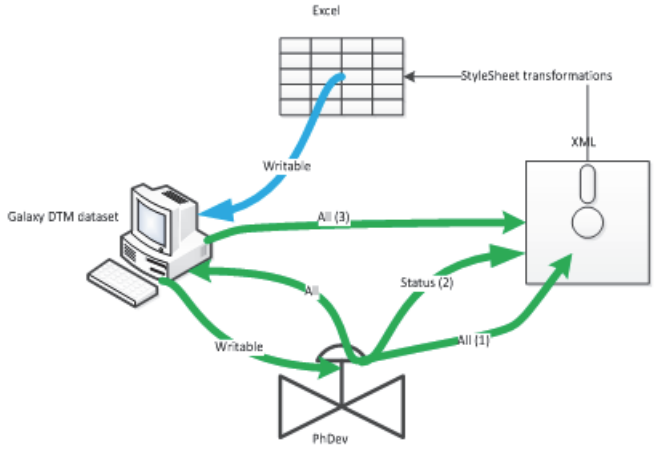


# Direct Access Support for Field Device Tool (FDT) Related Functions

FDT related functions are based on interfaces defined in the FDT specification. DTMs that implement these interfaces are used to:

- Read and write device parameters from/to the physically connected device
- Store or load the device dataset from the Galaxy
- Write and read the parameters to/from files. For example XML and Excel files

Figure 17-1 shows the possible flow of data between the physical device (PhDev), the Galaxy instance and files.




(1) All Parameters exposed via the FDT interface IDtmParameter.GetParameter of the HART instances using UDTM or vendor DTM, and DTM instance using the vendor DTM

(2) Status exposed via FDT interface IdtmOnlineDiagnosis.GetDeviceStatus of the HART instances using UDTM or vendor DTM, and DTM instance using the vendor DTM

(3) All parameters exposed through The FDT interface IDtmParameter.GetParameter

**Figure 17-1. Dataflow of FDT related functions**

The FDT related functions specified in this section are only supported for:

- HART device instances in the Network view connected to a CP hosted HART FBM
- HART device instances in the DTM Network view
- DTM device instances in the DTM Network view

The FDT functions are not supported for:

- Profibus device instances
- HART device instances connected to a Profibus Remote-I/O

# FDT Related Commands

**Syntax:**

```
<DTMOperation    Device="DeviceName"
                 DTMType="DTMTypeEnum"
                 OutputDirectory="OutputPath"
                 InputDirectory="InputPath"
                 DTMOperationSequence="DTMOperation"/>
```

For logging, the Direct Access LogMessage command and functions are used.

| Attribute | Description |
|---|---|
| DeviceName | The name of the HART or DTM device instance. |
| DTMTypeEnum | The DTM type does not need to be defined in each command. It needs to be defined only once in a script file, but can be redefined. |
| OutputPath | Full pathname where the results of the commands (XML, Excel files) are stored. The path does not need to be defined in each command. It can be defined only once in a script file or only when redefinition is needed.<br>The filenames are automatically generated using <DeviceName>_<DTMOperation>_<DateAndTime>. extension.<br>The DTMOperations are defined in the next section. |
| InputPath | Full pathname where the input files (XML, Excel, or XML transformation files) are stored. The path does not need to be defined in each command. It can be defined only once in a script file or only when redefinition is needed. |
| DTMOperation | FDT related function that is executed using the selected device instance and DTM.<br>Refer to “DTM Operations” on page 306 for more information. |

# DTM Operations

In the DTMOperationSequence, part of the FDT related commands, you can define the DTM operations, which are executed using the selected device instance and DTM.

The command keywords define the data, and a source and destination.

| DTM Operation Keywords | Description |
|---|---|
| AllParameters_PhDev_to_Galaxy | Combined command that connects, reads physical device parameters, and stores them to the Galaxy. |
| AllParameters_PhDev_to_File<br>Or<br>AllParameters_PhDev_to_File(XmlTransformationFile) | Combined command that connects, reads physical device parameters, and stores them into a file. Users can specify which XML transformation is used. If the XmlTransformationFile is omitted, only the XML file is written. |
| AllParameters_Galaxy_to_File<br>Or<br>AllParameters_Galaxy_to_File(XmlTransformationFile) | Combined command that reads the dataset from the Galaxy and stores them into a file. Users can specify which XML transformation is used. If the XmlTransformationFile is omitted, only the XML file is written. |
| WriteableParameter_Galaxy_to_PhDev | Combined command that connects to the device and loads parameters from the Galaxy to the physical device. |
| Status_PhDev_to_File<br>Or<br>Status_PhDev_to_File(XmlTransformationFile) | Combined command that connects to the device, reads from the device and writes the status to a file. Users can specify which XML transformation is used. If the XmlTransformationFile is omitted, only the XML file is written. |
| WriteableParameter_Excel_to_Galaxy (filename) | Reads parameters from an Excel file with a filename from the input path and stores it in the Galaxy. |

For efficiency reasons the following DTM operations can be combined into one operation sequence for a selected DTM:

- AllParameters_PhDev_to_Galaxy
- AllParameters_PhDev_to_File
- Status_PhDev_to_File
- ExecuteMethod_in_PhDev

**Example:**

```
<DTMOperation Device="MyHartDeviceInstance"
DTMType="UDTM"
OutputDirectory="D:\temp\Fox\Devices\Output"
InputDirectory=" D:\temp\Fox\Devices\Input"
DTMOperationSequence ="AllParameters_PhDev_to_Galaxy,
AllParameters_PhDev_to_File(ParamTransform.XSL),
Status_PhDev_to_File(StatusTransform.XSL)" />
```

**Note** Between any sequences of FDT commands, do not invoke other commands such as Create, Assign, Rename, and Delete.

# DTM Online Operations

The DTM operations AllParameters_PhDev_to_Galaxy, AllParameters_PhDev_to_File, WriteableParameter_Galaxy_to_PhDev, Status_PhDev_to_File, and ExecuteMethod_in_PhDev are online commands that must initialize the device DTM topology (needed to bring the selected DTM online) using the DTM datasets from the Galaxy. It brings the UDTM or Vendor DTM of the selected device instance to the online state (FDT communication set state). You do not need to provide the whole infrastructure information. The system will automatically set the necessary communication and gateway devices online.

These online operations can be used for HART instances in the DTM Network and Network view, and DTM instances in the DTM Network view.

HART instances that are assigned to a Profibus Remote-I/O in the network view are not supported.

Only one device DTM can be online/active at a time. If this command is called for a device instance and another device instance is still online, it sets the other DTM instance offline and brings the new DTM online.

# Read From Device Operations

The Read From device operations (AllParameters_PhDev_to_Galaxy, AllParameters_PhDev_to_File) must execute the FDT Upload function using the selected DTM and store the result in the defined destination.

The Read operations read parameters from the device if the device instance DTM is in an online state.

# Save to Galaxy Operations

The Galaxy Save operations (AllParameters_PhDev_to_Galaxy, WriteableParameter_Excel_to_Galaxy) will read the parameters from the source (physical device or file) and save the DTM dataset in the Galaxy instance object.

The selected instance is checked out. If check-out fails, an error is reported. Otherwise, the data is modified and then checked in.

# Write to Device Operations

The Download operation WriteableParameter_Galaxy_to_PhDev will execute the FDT Download function using the dataset stored in the Galaxy for the selected DTM.

This Download operation downloads/writes all writable parameters into the connected device using the dataset stored in the Galaxy for the selected DTM.

**Note** Before using this download command, you must ensure that the dataset in the Galaxy is a valid configuration for the connected device. To ensure a valid configuration, it is recommended to configure a device online until it is working as expected and then upload and store this dataset in the Galaxy instance object. With the **Copy DTM Configuration data to Template** function, you can update the template with valid configuration data and use it for other instances.

# Save Parameters to File Operations

The operations (AllParameters_PhDev_to_File, AllParameters_Galaxy_to_File) will get the parameters of a selected DTM and store the result into a file in the Output directory.

The operations (AllParameters_PhDev_to_File, AllParameters_Galaxy_to_File) use the FDT interface GetParameters to get the parameters from a DTM using the FDT DTMParameterSchema and store the result into an XML file. An optional XML transformation file allows transformation of the XML content into an Excel file.

The filenames are automatically generated using the name structure:

- <DeviceName>_AllParameters_PhDev_to_File_<DateAndTime>.extension
- <DeviceName>_AllParameters_Galaxy_to_File_<DateAndTime> .extension

The content (number of exposed parameters) depends on the (vendor) DTM implementation.

# Status Operations

The operation Status_PhDev_to_File will read the status information from the connected online device using the selected DTM and store the result into a file in the Output directory.

The operation Status_PhDev_to_File uses the FDT interface GetDeviceStatus to get the status from a DTM using the FDT DTMDeviceStatusSchema and stores the result into an XML file. An optional XML transformation file allows transformation of the XML content into an Excel file.

The implementation of the FDT interface that has to be used for this method is optional. Some vendor DTMs might not provide this functionality.

If a DTM supports this interface, it will read the current status from the device and provide this information in the XML file. The content (description of the device status) depends on the vendor DTM implementation.

# XML Transformation Operations

A transformation file contains the script, which is required to convert an xml document to Excel. Using the Transformation file, xml nodes can be represented in a tabular format. Nodes can be added or removed using the transformation file.

Device parameters or device status information can be saved in an Excel file using the AllParamToExcel.xsl and StatusToExcel.xsl files.

These files are located in the '[ArchestrA Framework DIRectory]\Bin\Invensys' directory under the DirectAccess folder. DTM operations that require the transformation file must place these files in the input directory. After the script has run successfully, excel files with *.XLS extension will be generated in the output Directory.

**Note** The transformation files AllParamToExcel.xsl and StatusToExcel.xsl are given as examples. New files can be created. The files provided can be used with the UDTM, but Vendor DTMs may not provide the required information and structure in the XML files and therefore the generated result may not be satisfactory.

To view this file in a proper format, open the generated Excel file and save it in the .xlsx format by following the below steps.

1. Double-click the *.XLS file generated.

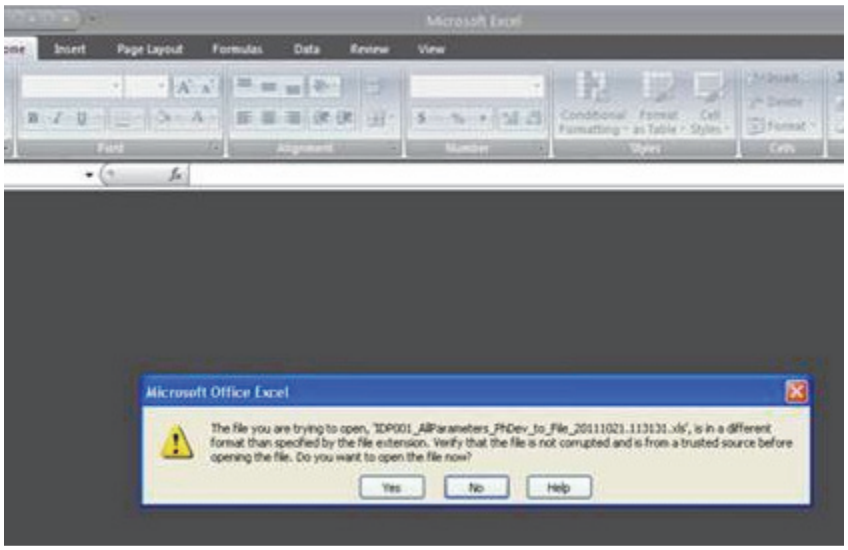



**Figure 17-2. Microsoft Office Excel Warning**

2. Click **Yes** on the warning message displayed above.

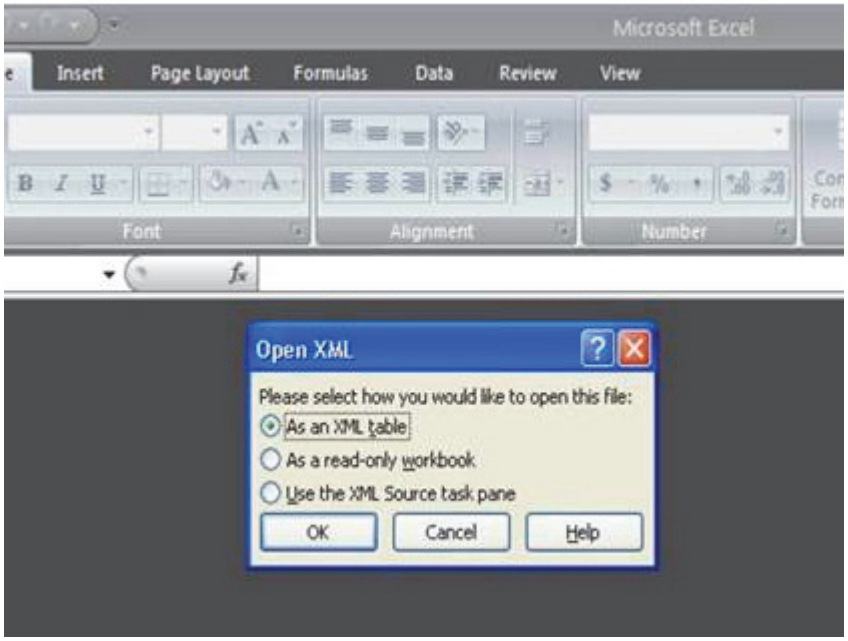



**Figure 17-3. Open XML**

3. Select **As an XML table** and click **OK**.
4. Click **Save** to save the file in *.xlsx format.

**Note** The file must be saved in the *.xlsx format and its name must match the XML file name. This is required if the file is used as an input for the WriteableParameter_Excel_to_Galaxy operation.

# WriteableParameter_Excel_to_Galaxy Operations

With this DTM operation modified parameters from the Excel file are saved to the Galaxy database.

To execute this DTM operation, proceed as follows:

- Install Microsoft Excel (version 2007 onwards) on the workstation.
- The *.xlsx and *.XML files must be placed in the input directory and have the same name.

**Note** For commands with Excel files, the input and output directory must have the same path.

CHAPTER 18

# Miscellaneous

This chapter discusses the MoveBlocks, UpdateRefs, SetExecutionOrder, ExecOrderItem and SaveAllExport commands.

## Contents



# ExecOrderItem Command

The **ExecOrderItem** command allows the user to specify the block, ECB or compound within the context of a **SetExecutionOrder** command (described in the previous section). This command can only be used nested within a **SetExecutionOrder** command.

### Syntax:

```
<ExecOrderItem      Compound="CompoundName"
                    ECB="ECBName"
                    Block="BlockName" />
```

| Attribute | Description |
|---|---|
| CompoundName | The name of the compound to be placed in the execution order of the currently executing controller **SetExecutionOrder** command. |
| ECBName | The name of the ECB to be placed in the execution order of the currently executing compound **SetExecutionOrder** command. |
| BlockName | The name of the block to be placed in the execution order of the currently executing strategy or compound **SetExecutionOrder** command. |

**Examples:**

Refer to the examples in “SetExecutionOrder Command” on page 320.

# MoveBlocks Command

The **MoveBlocks** command allows the user to move one or more blocks from one strategy instance to another. The filter argument is required and specifies the source for the move. The object level part of the filter must resolve to a single strategy. The block level part of the filter must resolve to one or more blocks within the source strategy. All blocks matching the filter will be moved to the target strategy, specified by the **StrategyName**. The target strategy must exist before this command executes. The target strategy will not be automatically created if it does not already exist.

All connections are retained through the move:

- Connections between moved blocks and unmoved blocks are retained by creating the necessary input and output declarations on the source and target strategies to reconnect the blocks.
- External sink connections to blocks are moved with the blocks. Connections made via input declarations to moved blocks are reestablished by making the same input declarations in the target strategy.
- External source connections to blocks are moved with the blocks by making corresponding output declarations, then scanning the entire Galaxy to find all references to the old output declaration and updating them. More details on this operation are provided below.

Once the block(s) have been moved to the target strategy, the Galaxy must be scanned to determine if there were any references to the moved block(s) that are now invalid. This operation can be very time consuming, especially if there are many strategies in the Galaxy. To optimize this operation, the user can opt to defer this processing until all the MoveBlocks commands are complete. Deferring this operation allows all of the broken references to be updated in one pass through the Galaxy.

This repair processing is done at the following times:

- Whenever a **MoveBlocks** command is executed **without** the UpdateRefs=“Defer” option. All prior moved blocks are updated at this time as well.
- When the UpdateRefs command is executed. All blocks moved with the UpdateRefs=“Defer” option since the last repair processing was done will be repaired.
- At the end of the currently executing script. All deferred block processing will be done.

**Note** The **MoveBlocks** command is intended to move blocks between strategies without changing the view of the blocks. In other words, the **COMPOUND:BLOCK** hierarchical name for each block must remain the same. Therefore, the command is only allowed if the source and target strategy are contained within the same compound.

**Note** This command can only be used to move blocks between "top level" strategies. It cannot be used to move blocks to/from nested strategies.

### Syntax:

```
<MoveBlocks          NewStrategy="StrategyName"
                     Filter="FilterName"
                     UpdateRefs="UpdateRefMode"/>
```

| Attribute | Description |
|---|---|
| StrategyName | Required. The name of the target strategy for the move. All blocks specified by the filter will be moved to this strategy.<br>The strategy must be contained within the same compound as the source strategy.<br>The strategy cannot be a nested strategy or derived from a template other than **$Strategy**. |
| FilterName | Required. The source strategy and blocks for the move.<br>The object portion of the filter must resolve to exactly one strategy.<br>The block portion of the filter may resolve to one or more blocks within that strategy. |
| UpdateRefMode | Optional. Indicates when the references to the moved block(s) are updated.<br>Options are:<br>"**Now**" - the scan is done immediately after the block(s) are moved (Default)<br>"**Defer**" - the scan is done when the **UpdateRefs** command is encountered or at the end of the current command script, whichever comes first. |

### Examples:

Move all blocks based on $AIN from **Strategy1** to **Strategy2**. **Strategy2** must already exist. The entire Galaxy will be scanned for references to the moved blocks and be updated appropriately:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy1"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<MoveBlocks NewStrategy="Strategy2" Filter="Filter1"/>
```

Move all blocks based on $AIN from **Strategy1** to **Strategy2**. Then move all blocks based on $AOUT from **Strategy1** to **Strategy2**. The entire Galaxy will be scanned for references to the moved blocks and be updated appropriately only after the second **MoveBlocks** command is executed:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy1"/>
```

```
<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<MoveBlocks NewStrategy="Strategy2" Filter="Filter1"
UpdateRefs="Defer" />

<QueryFilter Filter="Filter2" Condition="NameEquals"
Value="Strategy1"/>

<BlockQueryFilter Filter="Filter2" Condition="BasedOn"
Value="$AOUT"/>

<MoveBlocks NewStrategy="Strategy2" Filter="Filter2"/>
```

# MoveECBs Command

The **MoveECBs** command allows the user to move one or more ECBs from one compound instance to another. The ECB can be explicitly specified or one or more ECBs can be specified via the **Filter** argument. The object filter must resolve to one or more compounds. The block filter must resolve to one or more ECBs within the source compound(s). All ECBs matching the filter will be moved to the target compound, specified by **NewCompound**. The target compound must exist and be contained by the same controller before this command executes. The target compound will not be automatically created if it does not already exist.

**Note** I/O and block connections to the moved ECB(s) are retained through the move operation.

### Syntax:

```
<MoveECBs      NewCompound="NewCompoundName"
               Filter="FilterName" />
               Or
<MoveECBs      NewCompound="NewCompoundName"
               Compound="OldCompoundName"
               ECB="ECBName" />
```

| Attribute | Description |
|---|---|
| NewCompoundName | Optional. The name of the target compound for the move operation. All ECBs specified by the supplied arguments or filter will be moved to this compound. The compound must be contained within the same controller as the source compound. If **NewCompoundName** is not specified, the last referenced compound will be used. |
| FilterName | Specifies the source compound and ECB(s) for the move operation. The object filter must resolve to one or more compounds. The block filter must resolve to one or more blocks within the specified compound(s). If **FilterName** and **OldCompoundName**/**ECBName** are both specified, the explicitly specified **OldCompoundName**/**ECBName** are used. |
| OldCompoundName | Specifies the source compound for the move operation. The ECB specified by **ECBName** is moved from **OldCompoundName** to **NewCompoundName**. |
| ECBName | Specifies the ECB to be moved. If **OldCompoundName** is specified but **ECBName** is not, all ECBs in **OldCompoundName** will be moved to **NewCompoundName**. |

Either **NewCompoundName** or **OldCompoundName** must be specified. If neither is specified, the command will fail.

### Examples:

Move all blocks based on $ECB5 from **COMP1A** and **COMP1B** to **COMP2**. **COMP2** must already exist and be contained by the same controller as **COMP1**:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMP1A"/>

<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="COMP1B"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$ECB5"/>

<MoveECBs NewCompound="COMP2" Filter="Filter1"/>
```

Move ECB **F00001** from **CP6001_ECB** to **COMP2**:

```
<MoveECBs NewCompound="COMP2" Compound="CP6001_ECB"
ECB="F00001" />
```

Create new compound **COMP2** and move all ECBs from **CP0100_ECB** to **COMP2**:

```
<CreateCompound Compound="COMP2" Controller="CP0100" />

<MoveECBs Compound="CP0100_ECB" />
```

# SaveAllExport Command

The **SaveAllExport** command allows the user to select multiple controllers or compounds and export their control logic to a **SaveAll** format in a user-specified folder.

The Control Editors Library objects containing HLBL and SFC includes referenced by the **SaveAll** content can also be exported.

### Syntax:

```
<SaveAllExport      Filter="FilterName"
                    Compound="CompoundName"
                    Controller="ControllerName"
                    Path="ExportDirectory"
                    LibPath="LibraryExportDirectory"/>
```

**Note** The attributes **Filter**, **Controller** and **Compound** are mutually exclusive. That is, only one of the three attributes should be used to determine the objects in the Galaxy, to be considered for the **SaveAll** export.

| Attribute | Description |
|---|---|
| FilterName | The name of the query filter that will be used to determine what objects in the Galaxy are to be considered for the export.<br>This query filter must be created by the **QueryFilter** command.<br><br>**Note** The filter should contain either Controllers or compounds. Other than these objects **SaveAll** export will not be performed.<br>If the query filter results in the same object being found multiple times, it will be considered only once. |
| CompoundName | The name of the compound which is to be exported via a **SaveAll** export. |
| ControllerName | The name of the Controller which is to be exported via a **SaveAll** export. |

| Attribute | Description |
|---|---|
| ExportDirectory | The name of the directory to which objects are exported. During export, if this directory does not exist, it will be created. |
| LibraryExportDirectory | (Optional) The name of the directory to which the Library objects are exported.<br>During export, if this directory does not exist, it will be created.<br>When this attribute is not included in a **SaveAllExport** command, the command exports only the **SaveAll** content to the target export directory specified by Path="ExportDirectory" and any Control Editors Library objects containing HLBL and SFC includes referenced by the **SaveAll** command are ignored.<br>**Note** **ExportDirectory** and **LibraryExportDirectory** should be different. If they are the same, the Library object export will not be performed but the **SaveAll** export will be performed. |

**Examples:**

**SaveAll** export of all FCP280s in the configuration to the folder **D:\temp**.

```
<QueryFilter Filter="Filter1" Condition="BasedOn" Value="$FCP280"/>

<SaveAllExport Filter="Filter1" Path="D:\temp" />
```

**SaveAll** export of Controller **FDA100** in the configuration to the folder **D:\temp**:

```
<SaveAllExport Controller="FDA100" Path="D:\temp" />
```

**SaveAll** export of all compounds in the configuration to the folder **D:\temp** and also a Library object export of library objects of type **HLBL includes** and/or **SFC includes** referred to by the compounds that are to be exported via a **SaveAll** export to the folder **D:\temp_lib**:

```
<QueryFilter Filter="Filter1" Condition="BasedOn"
Value="$COMPND"/>

<SaveAllExport Filter="Filter1" Path="D:\temp" LibPath="D:\temp_lib"
/>
```

**SaveAll** export of compound **COMPND_001** in the configuration to the folder **D:\temp**:

```
<SaveAllExport Compound="COMPND_001" Path="D:\temp" />
```

# SetExecutionOrder Command

The **SetExecutionOrder** command allows the user to set block, ECB, or compound execution orders. Block execution order may be set for strategies or compounds. ECB execution order may be set for compounds. Compound execution order may be set for controllers. Exactly one of the available attributes (**Controller**, **Compound** or **Strategy**) may be used for this command.

Objects specified will be ordered in the target object. All objects in the target execution order that are not specified will be appended to the execution order after the specified objects. See the examples below for more details.

**Note** Setting the execution order with the Direct Access application may log the warning message, "Warning: Execution order is not validated" when any of the following conditions exist:

- The numbers for execution order specified in Direct Access script are not contiguous, 1-x
- There are blocks in the target compound that are not defined in the Direct Access script
- There are blocks in the target compound that are not being generated at this time

### Syntax:

```
<SetExecutionOrder    Controller="ControllerName"
                      Compound="CompoundName"
                      Strategy="StrategyName" />
```

| Attribute | Description |
|---|---|
| ControllerName | The name of the controller for which the compound execution order is set. |
| CompoundName | The name of the compound for which either the block execution order or the ECB execution order is set. |
| StrategyName | The name of the strategy for which the block execution order is set. |

### Examples:

Set the relative execution order for blocks in a strategy instance:

```
<SetExecutionOrder Strategy="Strategy_001">
    <ExecOrderItem Block="AIN_1"/>
    <ExecOrderItem Block="PIDA_1"/>
    <ExecOrderItem Block="AOUT_1"/>
</SetExecutionOrder>
```

Set the relative execution order for blocks in a strategy template:

```
<SetExecutionOrder Strategy="$Strat_Template1">
    <ExecOrderItem Block="AIN_1"/>
    <ExecOrderItem Block="PIDA_1"/>
    <ExecOrderItem Block="AOUT_1"/>
</SetExecutionOrder>
```

Set the execution order for blocks in a compound:

```
<SetExecutionOrder Compound="COMP01">
    <ExecOrderItem Block="AIN_1"/>
    <ExecOrderItem Block="PIDA_1"/>
    <ExecOrderItem Block="AOUT_1"/>
</SetExecutionOrder>
```

Set the execution order for ECBs in a compound:

```
<SetExecutionOrder Compound="COMP01">
    <ExecOrderItem ECB="F00001"/>
    <ExecOrderItem ECB="F00002"/>
    <ExecOrderItem ECB="DEV001"/>
</SetExecutionOrder>
```

Set the execution order for compounds in a controller:

```
<SetExecutionOrder Controller="FCP001">
    <ExecOrderItem Compound="COMP01"/>
    <ExecOrderItem Compound="COMP02"/>
    <ExecOrderItem Compound="COMP03"/>
</SetExecutionOrder>
```

Partially specify the relative execution order for blocks in a strategy instance:

Current execution order for **Strategy_001**:

AIN_1
AIN_2
PIDA_1
PIDA_2
AOUT_1
AOUT_2

Execute the following command:

```
<SetExecutionOrder Strategy="Strategy_001">
    <ExecOrderItem Block="AIN_1"/>
    <ExecOrderItem Block="PIDA_1"/>
```

```
    <ExecOrderItem Block="AOUT_1"/>
</SetExecutionOrder>
```

Resulting execution order:

AIN_1

PIDA_1

AOUT_1

AIN_2

PIDA_2

AOUT_2

# UpdateRefs Command

The **UpdateRefs** command causes all of the strategies in the Galaxy to be scanned for references to blocks moved by previously executed **MoveBlocks** commands. Any broken references are repaired.

See the documentation for “MoveBlocks Command” on page 314 for other times at which this processing can occur.

### Syntax:

```
<UpdateRefs />
```

### Examples:

Move all blocks based on $AIN from **Strategy1** to **Strategy2**. Then move all blocks based on $AOUT from **Strategy1** to **Strategy2**. The entire Galaxy will be scanned for references to the moved blocks and be updated appropriately only when the **UpdateRefs** command is executed:

```
<QueryFilter Filter="Filter1" Condition="NameEquals"
Value="Strategy1"/>

<BlockQueryFilter Filter="Filter1" Condition="BasedOn"
Value="$AIN"/>

<MoveBlocks NewStrategy="Strategy2" Filter="Filter1"
UpdateRefs="Defer" />

<QueryFilter Filter="Filter2" Condition="NameEquals"
Value="Strategy1"/>

<BlockQueryFilter Filter="Filter2" Condition="BasedOn"
Value="$AOUT"/>

<MoveBlocks NewStrategy="Strategy2" Filter="Filter2"
UpdateRefs="Defer" />

<UpdateRefs />
```

CHAPTER 19

# Examples

This chapter gives several examples for using Direct Access software.

## Contents



## Create Child Strategy Template

This example creates a PID cascade loop as a child strategy template, then creates two parent strategies, each of which incorporates the child strategy template.

Depictions of the strategies created are shown below:

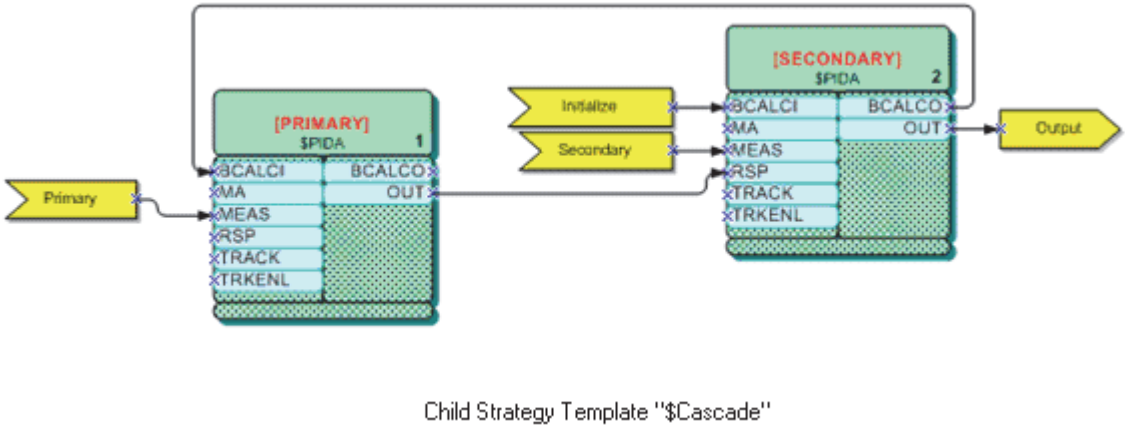



**Figure 19-1. Example of Creating a Child Strategy Template**

In the child strategy depicted above, two PIDA blocks are created called PRIMARY and SECONDARY. Four I/O declarations are then created, and attached to the PIDA blocks. The commands necessary to create this child strategy template are:

```
<CreateStrategy Template="$Strategy" Strategy="$Cascade"/>

<CreateBlock Template="$PIDA" Block="PRIMARY"
Strategy="$Cascade"/>

<CreateBlock Template="$PIDA" Block="SECONDARY"
Strategy="$Cascade"/>

<CreateDeclaration Decl="Primary" Type="Input" Strategy="$Cascade"
RefreshAO="false"/>

<CreateDeclaration Decl="Secondary" Type="Input"
Strategy="$Cascade" RefreshAO="false"/>

<CreateDeclaration Decl="Initialize" Type="Input" Strategy="$Cascade"
RefreshAO="false"/>

<CreateDeclaration Decl="Output" Type="Output" Strategy="$Cascade"
RefreshAO="true"/>

<CreateBlockAddressCxn Strategy="$Cascade" Sink="SECONDARY"
SinkParm="RSP" SinkValue="PRIMARY.OUT"/>

<CreateBlockAddressCxn Strategy="$Cascade" Sink="PRIMARY"
SinkParm="BCALCI" SinkValue="SECONDARY.BCALCO"/>

<CreateBlockAddressCxn Strategy="$Cascade" Sink="PRIMARY"
SinkParm="MEAS" SinkValue="Primary"/>

<CreateBlockAddressCxn Strategy="$Cascade" Sink="SECONDARY"
SinkParm="MEAS" SinkValue="Secondary"/>

<CreateBlockAddressCxn Strategy="$Cascade" Sink="SECONDARY"
SinkParm="BCALCI" SinkValue="Initialize"/>

<UpdateDeclaration Decl="Output" Strategy="$Cascade"
Value="SECONDARY.OUT"/>
```

These commands assume that the **$Strategy** and **$PIDA** templates already existed in the galaxy.

Note that if opened with the Strategy Editor, the child strategy template created above will not look exactly like the depiction. This is because there is no way to programmatically place blocks and declarations at ideal locations. When the user opens a strategy created with Direct Access (or with Bulk Generation), he or she is responsible for actually moving the blocks and declarations around the canvas to be aesthetically pleasing.

# Create Strategy with Child Strategy Instance

In this example, a repeat cycle is performed to generate two strategies which incorporate the child strategy created in the previous example.

A depiction of the first strategy is shown below:

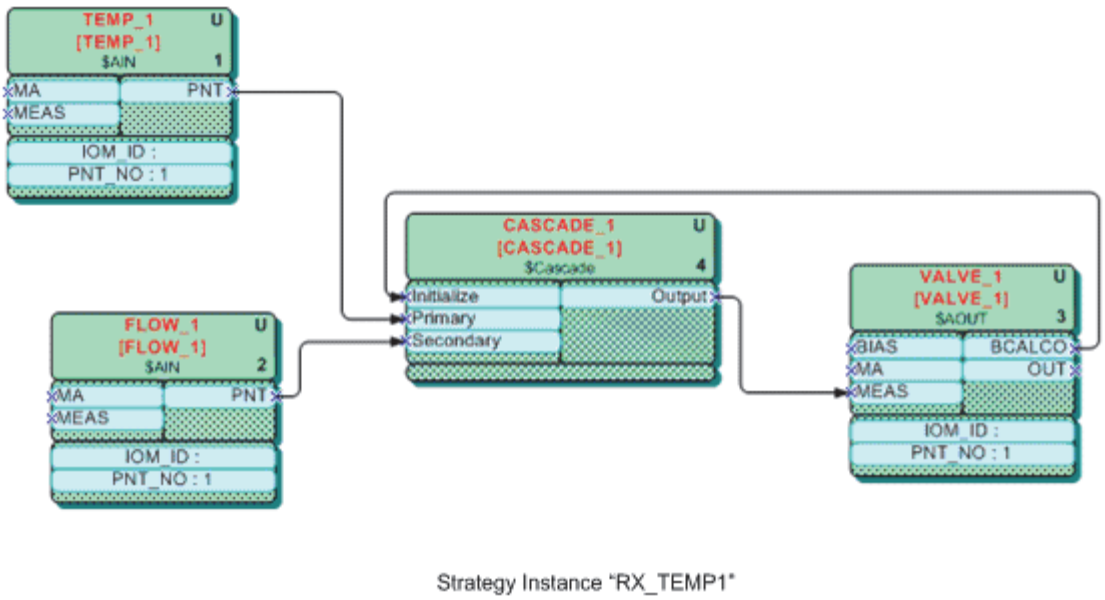



**Figure 19-2. Example of Creating a Strategy with a Child Strategy Instance**

In this strategy, two AIN blocks are used for the Primary and Secondary connections to the child strategy **CASCADE_1**. The output of the child strategy is routed to the **MEAS** attribute of the **AOUT** block, which in turn feeds back to the **Initialize** attribute of the child strategy. The commands necessary to produce this strategy are:

```
<!-- now, create two reactor control loops, RX_TEMP1 and RX_TEMP2--
>

<!-- each using the cascade created above for a child strategy -->

<PerformOperation Repeat="2" Start="1">

<CreateCompound Template="$COMPND"
Compound="COMPND_00^"/>

<CreateStrategy Template="$Strategy" Strategy="RX_TEMP^"
Compound="COMPND_00^"/>

<CreateBlock Template="$AIN" Block="TEMP_^"
Strategy="RX_TEMP^"/>

<CreateBlock Template="$AIN" Block="FLOW_^"
Strategy="RX_TEMP^"/>

<CreateBlock Template="$AOUT" Block="VALVE_^"
Strategy="RX_TEMP^"/>

<CreateStrategy Template="$Cascade" Strategy="CASCADE_^"
ParentStrategy="RX_TEMP^"/>

<UpdateDeclaration Decl="Primary"
Strategy="COMPND_00^.RX_TEMP^.CASCADE_^"
Value="TEMP_^.PNT"/>
```

```
<UpdateDeclaration Decl="Secondary"
Strategy="COMPND_00^.RX_TEMP^.CASCADE_^"
Value="FLOW_^.PNT"/>

<UpdateDeclaration Decl="Initialize"
Strategy="COMPND_00^.RX_TEMP^.CASCADE_^"
Value="VALVE_^.BCALCO"/>

<CreateBlockAddressCxn Strategy="COMPND_00^.RX_TEMP^"
Sink="VALVE_^" SinkParm="MEAS"
SinkValue="CASCADE_^.Output"/>

</PerformOperation>
```

Notice the use of the caret (‘^’) to produce unique object IDs – something very important when executing commands within a **PerformOperation** repeat cycle. Without the use of the caret, it would be almost impossible to create connections to the correct object(s).

# Import Base Templates

In this example, all the files needed to generate a complete database, with all the base templates used in the Control Software, are imported. This includes any hidden objects that we need for global data operations.

```
<!-- import base support objects -->

<ImportOptions Directory="C:\InFusion\bin"/>


<!-- wildcards are allowed - only one * or ? per line -->

<ImportFile Files="*.aaPDF"/>

<ImportFile Files="IA_Blocks\*.aaPDF"/>

<ImportFile Files="IA_ECBs\*.aaPDF"/>

<ImportFile Files="IA_Hardware\*.aaPDF"/>
```

# Import Galaxy Dump Files

In this example, ArchestrA objects that were previously exported via Galaxy Dump are imported into the Galaxy:

```
<!-- import directory -->

<ImportOptions Directory="C:\temp\DumpFiles"/>


<!-- wildcards are allowed - only one * or ? per line -->

<ImportFile Files="*.csv"/>
```

CHAPTER 20

# Troubleshooting

This chapter gives several methods to troubleshoot errors that may appear when using Direct Access software.

## Contents



# Exporting to SQL Tables

The following error message may appear if you attempt to export Control Software data to SQL tables:

```
SQL exception: SQL Server blocked access to STATEMENT
'OpenRowset/OpenDatasource' of component 'Ad Hoc
Distributed Queries' because this component is turned
off as part of the security configuration for this
server.
```

This error appears when you are trying to export to SQL tables because write access to the database for this type of operation is currently disabled.

To enable this feature for Foxboro Control Software v3.1 to v4.x and Foxboro Evo Control Software v5.0 or later, perform the following steps on the Galaxy server:

1. Invoke the SQL Server Management Studio.
   Connect to “server <workstation name>”, not “<workstation name>\INFUSIONSECURITY”.
2. Right-click on the workstation name and select **Facets**.
3. In the View Facets dialog box, change “Facet” type from **Server** to **Surface Area Configuration**.
4. Change **AdHocRemoteQueriesEnabled** status from **False** to **True**.
5. Click **OK** to close the View Facets dialog box.

To enable this feature for InFusion software v2.5 and earlier, perform the following steps:

1. Invoke the SQL Server Surface Area Configuration (**Start** > **Programs** > **Microsoft SQL Server 2005** > **Configuration Tools** > **SQL Server Surface Area Configuration**).
2. Click the link named **Surface Area Configuration for Features**.
3. In the dialog box that appears, expand the tree **MSSQLSERVER** > **Database Engine**.
4. Select the tree node **Ad Hoc Remote Queries**.
5. Select **Enable OPENROWSET and OPENDATASOURCE support**.
6. Click **OK** to close the dialog box.
7. In the SQL Server Surface Area Configuration dialog box, click the link **Surface Area Configuration for Services and Connections**.
8. In the dialog box that opens, expand the tree **MSSQLSERVER** > **Database Engine** > **Remote Connections**.
9. Select **Local and remote connections**.
10. Select **Using TCP/IP only**.
11. Click **OK** to close the dialog box.
12. Close the SQL Server Surface Area Configuration dialog box.

Direct Access software will now be able to write to SQL tables.

# Executing a Direct Access Script

An error message similar to the following may appear when you attempt to execute a Direct Access script:

```
Exception: An error occurred while parsing EntityName.
Line ##, position ##.
```

This error usually indicates some invalid XML was provided to Direct Access. Check the following:

1. XML Commands are paired. If the second of the following commands is not present, this would be an invalid XML construct.

   <DirectAccess>

   </DirectAccess>

2. One-line XML commands have the closing "/":

   <CreateBlock Block="$BLK" Template="$AIN" />

3. You have no “special” XML characters in your data. The characters in the table below must be replaced with their XML equivalents when they are used in a Direct Access XML command file.

| Character | XML Equivalent |
|---|---|
| & | &amp; |
| < | < |
| > | > |
| " | " |
| ' | &#39; |

# Opening Direct Access Generated .xls Files with Microsoft Excel 2010

When opening files with the extension “.xls” generated by Direct Access using Microsoft Excel 2010, the following message may be displayed:

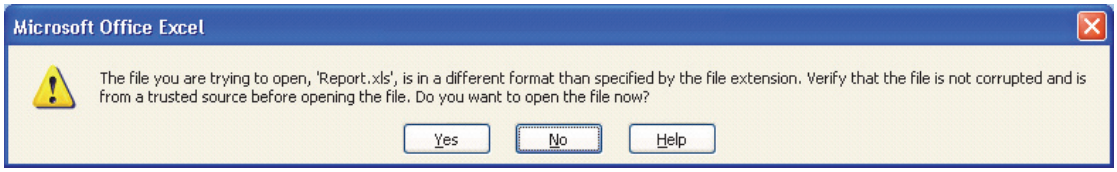



**Figure 20-1. Microsoft Excel 2010 Query**

This is a known issue with Excel 2010. Click “**Yes**” and the file will be opened. This issue does not occur when opening files with the extension “.xlsx”.

**Note** For legacy Foxboro Control Software (FCS) v3.1 or earlier, the command uses Excel 2007, while for FCS v4.x and Foxboro Evo Control Software v5.0 or later, the command uses Excel 2010.

CHAPTER 21

# SaveAll Validation Utility

This chapter describes the SaveAll Validation Utility, which allows you to compare the contents of a SaveAll with a Direct Access “BlockDetail” report.

## Contents



## Introduction

The SaveAll Validation utility compares the contents of an SaveAll with a Direct Access “Block Detail” report, generated from a Control Editors database. The utility writes a CSV file that contains differences between non-default parameters in the SaveAll and the I/A Series Galaxy. Figure 21-1 shows the SaveAll Validation interface.

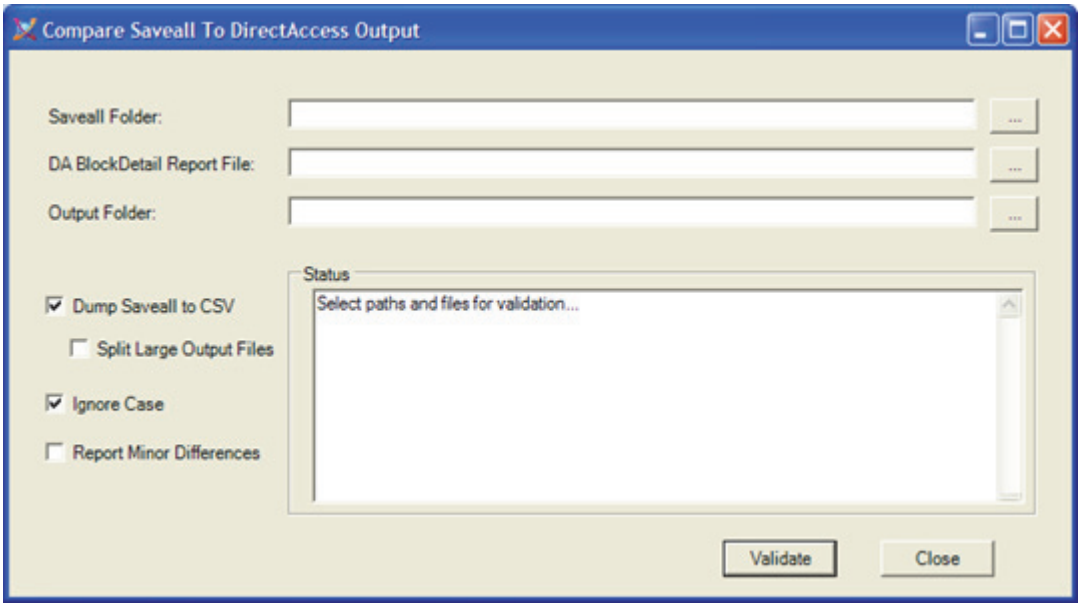



**Figure 21-1. SaveAll Validation Dialog Box**

The SaveAll Validation utility is installed as part of the Control Software. No additional steps are required to install the utility.

# Executing the SaveAll Validation Utility

To execute the SaveAll Validation Utility, double-click the following executable:

**ValidateSaveall.exe**

This executable is located in the <*IAS install folder*>\framework\bin\Invensys directory by default. (is typically D:\Program Files\Archestra\framework\bin\Invensys.)

**Note** The SaveAll Validation utility does not support FOUNDATION™ fieldbus-specific information, and any files created from this utility will not contain any information regarding FOUNDATION fieldbus devices.

# Usage

The following describes the required values and options available when running the SaveAll Validation utility:

- **SaveAll Folder** (required) – This is the path to the root of the SaveAll. This path should point to the folder containing the ".Compound_Dir" file, normally A:\ if reading the SaveAll from diskette.
- **DA Block Detail Report File** (required) – This is the path to the CSV file generated by running a Direct Access Block Detail Report. An example of the Direct Access commands is given later in this document.

**Warning** Do **not** edit this file with Microsoft® Excel! Microsoft Excel will change some floating point numbers. (For example, 32.0 will be come 32.) This will cause extra differences to be reported. Alternatively the. csv file can be renamed to a .txt file, then imported to Microsoft Excel, forcing ALL fields to text type.

- **OutputFolder** (required) – This is the folder where the results of the comparison will be written. The output of this utility will be one or more CSV files.
- **Dump SaveAll to CSV** (optional) – When this option is selected, the contents of the SaveAll are also written to CSV files for reference.
  - *SaveallCompounds.csv* contains the non-default parameter values for all Compounds in the SaveAll.
  - *SaveallBlocks.csv* contains the non-default parameter values for all Blocks in the SaveAll.
- **Split Large Output Files** (optional) – Some editors (for example, Microsoft Excel software) capable of editing CSV files cannot handle more than 256 columns of data. Choosing this option will split the output

into separate files. Each file will contain no more than 250 columns, and key columns (Compound, Block, and so forth) are repeated in each file.

- **Ignore Case** (optional) – All data reported by Direct Access software for control parameters is reported as uppercase characters. Depending on the source and content of the SaveAll, the SaveAll may contain lowercase characters. Choosing this option allows these mismatches to be ignored.
- **Report Minor Differences** (optional) – If selected, minor differences in connection strings and IOM_ID values are reported as differences. Minor differences are:
  - IOMID: Saveall = DEV001, Galaxy = CP0100_ECB:DEV001 (strings are different, but values are equivalent)
  - IOMID: Saveall = :DEV001, Galaxy = SAMECOMPOUND:DEV001 (strings are different, but values are equivalent)
  - Block Connection: Saveall = :BLOCK.PARM, Galaxy = SAMECOMPOUND:BLOCK.PARM (strings are different, but values are equivalent)
- **Status** – This is where errors and general status information are reported to the user.
- **Validate Button** – Begins the validation process.
- **Close Button** – Closes the dialog box.

# Direct Access Block Detail Report Example

```
<?xml version="1.0" encoding="utf-8"?>
<DirectAccess>
<QueryFilter Filter="Filter1" Condition="NameEquals" Value="CP0100" />
<ReportOptions Directory="C:\Temp" DumpOptions="Source" />
<ProduceReport Type="CSV" ReportName="BlockDetail" FileName="CP0100"
Filter="Filter1" Cascade="Yes"/>
</DirectAccess>
```

**Figure 21-2. Direct Access Block Detail Report Example**

The script in Figure 21-2 generates the output file required to validate a SaveAll import for control station CP0100. The individual commands are described below:

- `<?xml>` – Defines the character encoding and XML version for the file content. This line can be copied verbatim.
- `<DirectAccess>` – The script to execute in Direct Access must always be wrapped by this tag and its closing tag: </DirectAccess>
- `<QueryFilter>` – This defines the objects to be reported. For Saveall Validation purposes, the name of the Control Station containing the Saveall to be validated is recommended.

- `<ReportOptions>` – The name of the folder where the Block Detail Report is to be written and indicates that block source is to be reported
- `<ProduceReport>` – Generates the report.
  - Type must be “CSV”
  - ReportName must be “BlockDetail”
  - Filter must be the name of the QueryFilter in line 3
  - Cascade must be “Yes” so all contained objects of the named Control station are reported
  - FileName can be any user-defined name. The output file will be <FileName>.csv.

In the example in Figure 21-2, the output file will be CP0100.csv. CP0100.csv is the file required as input to the SaveAll Validation utility.

# Sample Output

Missing Block in Database: CPDK11_STA.STATION

Missing Block in Database: CPDK11_ECB.PRIMARY_ECB

| Compound | Strategy | Block | Parameter | Saveall Value | Database Value |
|---|---|---|---|---|---|
| FD1_SD_ESD | FD1_LO_TEMP_ESDCP | LO_TEMP | BI01 | FD1_ESD:TID181A_ALM.LAIND | |
| CONT_SD | CT_1 | CT_1 | RI01 | FA58:PIA356.PNT | |
| CONT_SD | CT_9 | CT_9 | BI11 | FA59:TIA202_AHH.HHAIND | |
| CONT_SD | CT_9 | CT_9 | BI13 | FA59:TIA204_AHH.HHAIND | |
| FD1_SD_ESD | PID10_ALM | FD1_PAHD | SETPT | FD1_ESD:PID10_ALM.HABLIM | |
| FD1_SD_ESD | PID10_ALM | FD1_PAHD | MEAS | FD1_SD_ESD:PID256_ALM.HABLIM | :PID256_ALM.HABLIM |
| FD1_SD_ESD | XYXD36 | XYXD36 | BI01 | FD1_ESD:PID10_A.HAIND | |
| PDA_SD | F012674 | HSA185C | IOM_ID | P01103 | CPDK11_ECB:P01103 |
| PDA_SD | HSA185B_F012674 | HSA185B | IOM_ID | SD0201 | CPDK11_ECB:SD0201 |
| CONT_SD | CT_1 | CT_1 | RI03 | FA59:PIA357.PNT | |
| CONT_SD | CT_1 | CT_1 | BI01 | FA58:PIA356_AHH.HHAIND | |
| CONT_SD | CT_1 | CT_1 | BI05 | FA59:PIA357_AHH.HHAIND | |
| FD19_SD_ESD | XYXD110 | XYXD110 | BI07 | FD19_ESD:HID286A.CIN | |
| FD1_SD | S_2113 | XID90A | IOM_ID | SD0301 | CPDK11_ECB:SD0301 |

| | | | | | |
|---|---|---|---|---|---|
| FD2_SD | HSD134_2115 | XSD115 | IOM_ID | SD0116 | CPDK11_ECB:SD0116 |
| FD19_SD_ESD | FCD3_PURG | XYXD32_PGL | BI03 | FD19_ESD:PID334_PRG.HAIND | |

**Figure 21-3. Sample BlockDiffs.csv Output**

Figure 21-3 shows sample output for the BlockDiffs.csv file. Things to note:

- First two rows show a sample of missing Blocks in the Galaxy. Missing Blocks can be caused by 2 things:
  - The Block truly does not exist
  - There were no parameters in the block that had non-default parameter values
- Column Headers
- Compound/Strategy/Block – The block that had the difference
- Parameter – Parameter with difference
- SaveAll Value – Value extracted from SaveAll image
- Database Value – Value from Galaxy
- Unresolved Connections – Most connection strings that get reported as differences are reported because the source end of the connection does not yet exist. When the other end of the connection is created, these parameters will no longer show as differences.
- Minor Differences – Rows highlighted in yellow are rows that would only appear as differences if the “Report Minor Differences” option is checked. Note that this output file is a CSV file and rows will not actually be highlighted in the real output. This was done here for clarity.

Missing Compound in Database: CPDK11_ECB

Missing Compound in Database: CPDK11_STA

| Compound | Parameter | Saveall Value | Database Value |
|---|---|---|---|
| BATCH_SD | DESCRP | IA COMPOUND | IA COMPOUND TEXT CHANGED |
| BATCH_SD | GR1DV3 | WPPD03 | WPPD04 |

**Figure 21-4. Sample CompoundDiffs.csv Output**

Figure 21-4 shows sample output from the CompoundDiffs.csv file. Things to note:

- Missing Compounds – Here, again, if there were no non-default parameter values, compounds may show up as missing.
- All other information is similar to that of the BlockDiffs.csv output.

The DirectAccess.exe and ValidateSaveall.exe files are located here:

D:\Program Files\ArchestrA\Framework\Bin\Invensys\DirectAccess.exe

D:\Program Files\ArchestrA\Framework\Bin\Invensys\ValidateSaveall.exe

# Index

T

Invensys Systems, Inc.
10900 Equity Drive
Houston, TX 77041
United States of America
http://www.invensys.com

---

Global Customer Support
Inside U.S.: 1-866-746-6477
Outside U.S.: 1-508-549-2424
Website: https://support.ips.invensys.com