

Wonderware® Application Server User's Guide

Invensys Systems, Inc.

Revision D

Last Revision: 10/27/09



Copyright

© 2009 Invensys Systems, Inc. All Rights Reserved.

All rights reserved. No part of this documentation shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Invensys Systems, Inc. No copyright or patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this documentation, the publisher and the author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

The information in this documentation is subject to change without notice and does not represent a commitment on the part of Invensys Systems, Inc. The software described in this documentation is furnished under a license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of these agreements.

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

<http://www.wonderware.com>

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

Trademarks

All terms mentioned in this documentation that are known to be trademarks or service marks have been appropriately capitalized. Invensys Systems, Inc. cannot attest to the accuracy of this information. Use of a term in this documentation should not be regarded as affecting the validity of any trademark or service mark.

Alarm Logger, ActiveFactory, ArcestrA, Avantis, DBDump, DBLoad, DT Analyst, Factelligence, FactoryFocus, FactoryOffice, FactorySuite, FactorySuite A², InBatch, InControl, IndustrialRAD, IndustrialSQL Server, InTouch, MaintenanceSuite, MuniSuite, QI Analyst, SCADAAlarm, SCADASuite, SuiteLink, SuiteVoyager, WindowMaker, WindowViewer, Wonderware, Wonderware Factelligence, and Wonderware Logger are trademarks of Invensys plc, its subsidiaries and affiliates. All other brands may be trademarks of their respective owners.

Contents

Welcome.....	13
Documentation Conventions.....	13
Technical Support	14
Chapter 1 Getting Started with the IDE	15
What's a Galaxy?	15
Creating a New Galaxy	17
Connecting to an Existing Galaxy	19
Getting Around the IDE.....	20
Using the Template Toolbox.....	22
Using the Graphics Toolbox.....	22
Using IDE Application Views	23
Model View	23
Deployment View.....	26
Derivation View.....	29
Operations View	30
Customizing Your Workspace.....	31
Docking Views	31
Floating Views	31
Hiding Views.....	32
Resetting the Workspace	32
Synchronizing the Views.....	32
Configuring User Information	33
Logging on and Logging off.....	35
Changing Users	36

Chapter 2	Getting Started with Objects	37
	About Templates and Instances	39
	Instances	39
	Templates	39
	Propagation.....	40
	About Base Templates	41
	Application Templates	42
	Device Integration Templates	42
	System Templates	42
	About Derived Templates	44
	Viewing Object Properties	45
Chapter 3	Working with Objects	47
	Managing Toolsets	47
	Creating Toolsets.....	48
	Creating Child Toolsets	49
	Deleting Toolsets	49
	Creating Derived Templates.....	50
	Deriving Templates from Another Derived Template..	51
	Creating Contained Templates.....	52
	ApplicationObject Containment	55
	Using Contained Names	58
	Containment Examples.....	60
	Viewing Containment Relationships.....	62
	Renaming Contained Objects	62
	Editing Objects	63
	Getting Help	65
	Help File Structure	65
	About the General Editor Layout.....	66
	Locking and Unlocking Template Attributes	67
	Setting Object Security	70
	Group Locking/Security	71
	About the Object Information Page.....	72
	Customizing Help.....	73
	Finding the Help Folders	73
	About the Scripts Page.....	74
	About the UDAs Page	75
	About the Extensions Page	77
	Referencing Objects Using the Galaxy Browser.....	79
	Browsing for Attributes	79
	Viewing Attribute Details in the Galaxy Browser ...	81
	Browsing for Graphics	84

Browsing for Element Properties	85
Creating a Filter for the Galaxy Browser	86
Changing How Information is Shown in the Galaxy Browser.....	88
Chapter 4 Managing Objects.....	89
Checking Objects Out.....	89
Checking In Objects	90
Validating Objects	91
Validating Scripts and Other External Components ...	92
Validating Manually	93
Creating Instances	94
Renaming Objects.....	95
Deleting Objects	97
Exporting Objects.....	97
Exporting Script Function Libraries	98
Importing Objects.....	99
Importing Script Function Libraries	101
After You Import	101
Chapter 5 Enhancing Objects	103
Creating and Working with UDAs	103
UDAs and Scripting	104
UDA Naming Conventions.....	105
Writing and Editing Scripts.....	106
About Scripts	107
Script Execution	107
Locking Scripts	112
Creating and Working with Extensions.....	114
About Extension Inheritance.....	114
Using the InputOutput Extension.....	117
When Objects Are On Scan.....	118
Using InputOutput Extensions in Scripts	118
Using the Input Extension.....	119
Using the Output Extension.....	120
Working with Outputs	121
Quality of Input, InputOutput and Output Extensions	123
Using the Alarm Extension	124
Using the History Extension	125
Using the Boolean Label Extension	126
Creating and Working with Graphics	127

Adding Graphics	128
Modifying Graphics	128
Renaming Graphics.....	128
Deleting Graphics.....	129
Chapter 6 Deploying and Running an Application.....	131
Planning for Deployment	131
Determining Galaxy Status	133
Configuring Advanced Communication Management ..	134
Selecting Advanced Communication Management	136
Configuring Scan Modes	136
Deploying Objects.....	139
Deployment Error Messages.....	143
Redeploying Objects	144
Undeploying Objects	144
Uploading Run-time Configuration.....	145
Undeployment Situations	147
Chapter 7 Working with History	149
Application Server History Components	150
Sending Historical Data Between Application Server and Wonderware Historian	151
Saving Historical Data During Communication Outages.....	151
Collecting Late Data	152
Saving Object Attribute Data to the Historian	152
Saving Process Values as Historical Data	153
Saving Data Quality as Historical Data	154
Additional Quality Data Saved to the Historian	154
Saving the Data Timestamp as Historical Data.....	156
Saving Alarms and Events as Historical Data	158
Deploying and Undeploying Attributes	158
Saving Historical Data During Run Time	159
Advanced Communication Management	159
Store Forward Mode.....	160
Configuring Common Historical Attributes	160
Configuring System Objects to Store Historical Data... 164	
Configuring the WinPlatform Object to Store Historical Data.....	165
Configuring an AppEngine Object to Store Historical Data.....	169
Setting Up an AppEngine for Late Data.....	169

Configuring an Area Object to Save Alarm Counts as Historical Data	171
Configuring Application Objects to Save Historical Data.....	173
Chapter 8 Working with Alarms and Events	175
Understanding Events	176
Types of Events.....	176
Understanding Alarms.....	178
Types of Alarms	181
State Alarms	181
Limit Alarms.....	182
Target Deviation Alarms	183
Rate of Change Alarms	184
Statistical Alarms.....	185
Setting Alarm State with Object Attributes	185
AlarmModeCmd Attribute	185
AlarmInhibit Attribute	185
AlarmMode Attribute.....	186
_AlarmModeEnum Attribute.....	186
Setting Alarm State for Individual Alarms	187
Enabling, Silencing, and Disabling Alarms.....	188
Enabling Alarms.....	190
Silencing Alarms	190
Disabling Alarms.....	190
Throttling Alarms.....	191
Propagating Timestamps with Alarms and Events ...	192
Alarms or Events Become Active.....	192
Alarms Become Disabled	192
Alarms Revert to Normal.....	193
Alarm Acknowledgement	193
Configuring Alarms.....	194
Configuring Alarming for System Objects	194
Configuring WinPlatform Object Alarms.....	196
Configuring Alarms for an AppEngine Object.....	198
Configuring Alarms and Events for Application Objects	200
Setting Alarms on the Extension Page	203
Distributing Alarms and Events	204
Subscribing to Alarms and Events from a Client	205
Using InTouch HMI as the Alarm and Event Client .	206
Understanding the Syntax of Alarm Queries	206

Alarm Query Syntax when Register Using Galaxy_<GalaxyName> is Enabled	206
Examples of Alarm Queries	207
Alarm Requirements for InTouch Client Applications	209
Alarms and Events in the InTouch HMI and in Application Server	210
Chapter 9 Working with References	213
Using Message Exchange and Attributes	214
Reference Strings	214
Relative References	215
Property References	216
Handling Time Zones with the Time Property	217
Preserving Time Stamps from the Publishing Source	217
Arrays	218
Formatting Reference Strings	219
Using Literals	219
Viewing Attributes in Objects	222
Viewing References and Cross References	223
Finding Objects	225
Using Galaxy References in InTouch	226
Chapter 10 Working with Security	231
About Security	232
About Authentication Modes	233
Multiple Accounts Per User	233
Changing Security Settings	234
About Security Groups	235
About Roles	236
About Users	237
Configuring Security	237
Assigning Users to Roles	242
Deleting Security Groups	244
Deleting Roles	244
Deleting Users	244
About OS Group-based Security	245
Connecting to a Remote Node for the First Time	245
Cached Data at Log In	245
Mixed or Native Domains	246
Using InTouch Access Levels Security	246

Chapter 11 Working with Languages.....	247
Defining and Configuring Galaxy Languages	248
Graphics Language Switching.....	248
Alarm Comment Language Switching	248
Workflow	248
Configuring Languages for a Galaxy.....	250
Adding a Language to a Galaxy.....	250
Removing a Language from a Galaxy	251
Modifying the Font for a Language	252
Changing the Default Language for a Galaxy	253
Exporting Symbol Text for Offline Translation.....	255
Types of Language Dictionary Files	255
Exporting Language Data for All Symbols in a Galaxy.....	256
Exporting Language Data for Specific Objects	257
Exporting Symbol Language Data for a Managed InTouch Application.....	258
Exporting Symbol Language Data for a Published InTouch Application.....	259
Exporting Symbol Text to an Existing Dictionary File	260
Translating Exported Symbol Language Files	261
Translating Exported Symbol Text Dictionary Files	261
Importing Translated Symbol Language Files	262
Importing Translated Symbol Dictionary Files	263
Examples of Symbol or Object Mismatch Handling during Language Imports.....	265
Language Data Handling for Galaxy Operations.....	267
Exporting Alarm Comments for Offline Translation	269
Guidelines and Recommendations	269
Organizing Your Export.....	269
Translation File Formatting and Editing	269
Reimporting	269
About the Alarm Comments Language File	270
Exporting Alarm Comments from Very Large alaxies.....	271
Exporting All Galaxy Alarm Comments	271
Exporting Alarm Comments by Area	272
Using File Names	272
Exporting Objects Not Assigned to an Area	272
Translating Exported Alarm Comment Language Files.....	274

Importing Translated Alarm Comment Language Files	276
Re-exporting Alarm Comments	277
Exporting New Untranslated Alarm Comments	278
Exporting Modified Existing Alarm Comments	278
Testing the Language Switching Functionality at Run Time	278
Chapter 12 Managing Galaxies	279
Backing Up and Restoring Galaxies	279
Changing Galaxies	280
Deleting a Galaxy	281
Exporting a Galaxy Dump File.....	282
Looking at the Galaxy Text Dump File Structure.....	283
Host Attributes	283
About Quotation Marks and Carriage Returns	284
Time Formats in Excel	284
Importing a Galaxy Load File.....	285
Synchronizing Time across a Galaxy	286
Using Time Synchronization in Windows Domains ...	286
Synchronization Schedule.....	287
Required Software	287
Hosting Multiple Galaxies in One Galaxy Repository ..	288
Managing Licensing Issues.....	289
Viewing License and End-User License Agreement Information	289
Updating a License.....	294
Disk Space Requirements	295
Managing Communication between Galaxy Nodes.....	295
About ArcestrA User Accounts.....	296
Using Multiple Network Interface Cards	297
Defining the Order of the NIC.....	297
Configuring the IP Address and DNS Settings	298
Configuring Multiple NICs for the Vista and Windows Server 2008 Operating Systems	299
Chapter 13 Working with Redundancy	303
About Redundancy	304
Configuring AppEngine Redundancy.....	304
Redundancy during Run Time.....	306
Working with AppEngine Redundancy.....	307

- Configuring the Redundancy Message Channel..... 308
- Configuring Redundancy 309
 - Configuring Redundancy in Templates..... 310
 - Deleting Redundant AppEngines 311
- Deploying AppEngine Objects 311
 - Configuration Requirements 312
 - Undeploying AppEngine Objects 313
- During Deployment 314
 - Objects at Run Time..... 314
- During Run Time..... 314
 - AppEngine Redundancy States 315
- Troubleshooting..... 317
 - Generating Alarms 319
 - Generating History 320
- Working with Data Acquisition Redundancy 321
 - Configuring Data Acquisition Redundancy 321
 - Deploying Redundant DIOjects 322
 - What Happens in Run Time 322
 - RedundantDIOject and PLC Connectivity 322

Glossary..... 323

Index 333

Welcome

This guide describes how to use the ArchestraA Integrated Development Environment (IDE) to develop and manage Application Server applications.

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

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

In some areas of the Wonderware® Application Server, you can also right-click to open a menu. The items listed on this menu change, depending on where you are in the product. All items listed on this menu are available as items on the main menus.

Documentation Conventions

This documentation uses the following conventions:

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

Technical Support

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

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

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

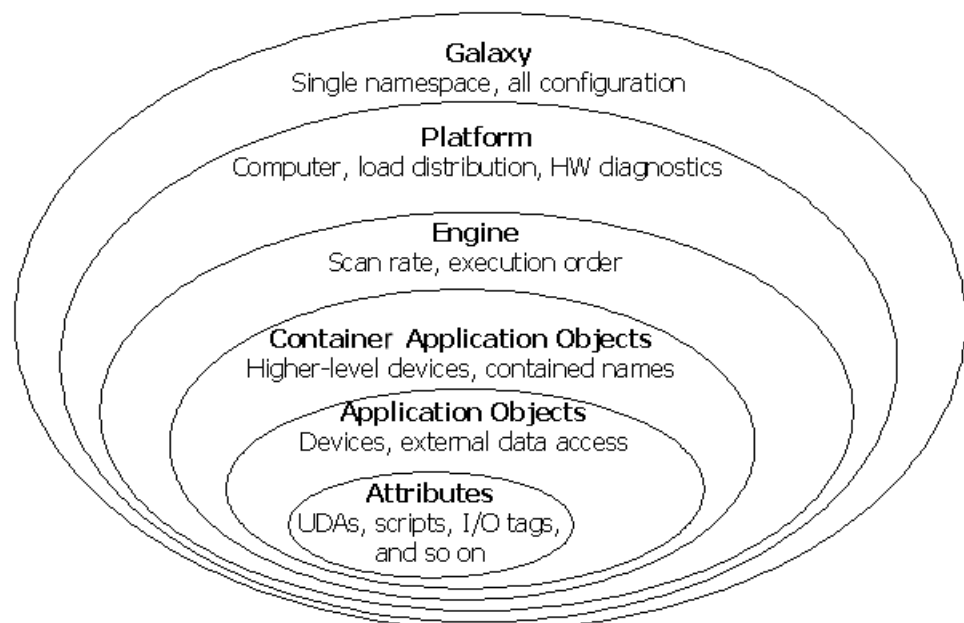
Chapter 1

Getting Started with the IDE

This section explains how to create and open a Galaxy. It also briefly describes the ArchestrA Integrated Development Environment (IDE).

What's a Galaxy?

A Galaxy represents your entire production environment, including all computers and components that run your application. A Galaxy is a collection of platforms, engines, templates, instances, and attributes you define as the parts of your specific application. Persistent information about this collection of objects is stored in a Galaxy database.



A Galaxy database resides on a single network computer. A Galaxy database can reside on any computer on your network with the SQL Server, Bootstrap, and Galaxy Repository software installed. But, you cannot store parts of a Galaxy database on several computers.

A Galaxy Repository (GR) is the name of the single computer where the Galaxy database is located.

You can deploy Galaxy components, such as platforms and engines, on multiple computers to share the work load while applications are running. For more information, see *Deploying and Running an Application* on page 131.

A Galaxy's namespace is the set of unique object and attribute identifiers. The namespace and the values of each of its identifiers define a Wonderware Application Server application, and can be accessed by clients of the configuration system as well as the Application Server Message Exchange in a deployed system.

A key benefit of the Application Server namespace is that it allows Application Server objects and process data to be referenced by scripts and animation links from any computer in the Galaxy without the reference needing to specify the object's location.

Galaxies also include security, which is turned off by default. Using security allows you to limit what users can do. You can add more users, security roles, and security groups later if you want. For more information, see *Working with Security* on page 231.

When you start the IDE, you must select an existing Galaxy or create a new Galaxy. You cannot open the IDE without opening a Galaxy.

Before you can open the IDE, you must also have a valid license. For more information about licensing issues, see *Managing Licensing Issues* on page 289.

Creating a New Galaxy

Each time you start the ArcestrA IDE, you must connect to a Galaxy. You must either create a new Galaxy or select an existing Galaxy before opening the IDE. Also, you must have a valid Wonderware license installed before opening the IDE.

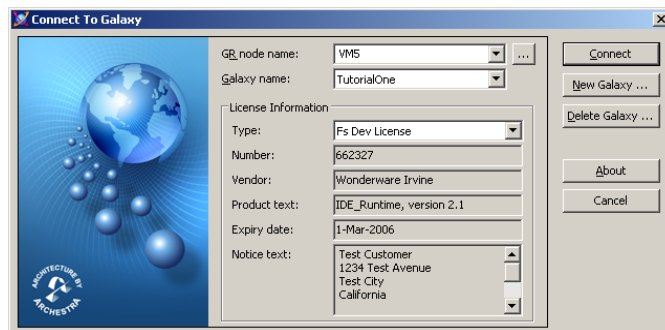
Creating a new Galaxy requires you to specify a Galaxy Repository (GR) node name and the name of the Galaxy. The Galaxy database is created and is ready for you to connect to and use.

You can only create a new Galaxy on a computer with the Bootstrap and the Galaxy Repository software installed.

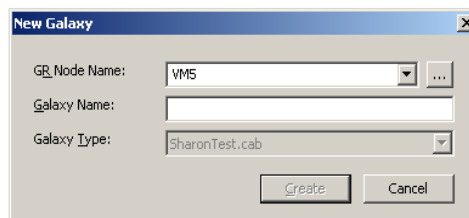
New Galaxies are created without security. To learn more about setting security for your Galaxy, see *Working with Security* on page 231.

To create a new Galaxy

- 1 On the **Start** menu, point to **Programs, Wonderware** and click **ArcestrA IDE**. The **Connect to Galaxy** dialog box appears.



- 2 Click **New Galaxy**.



3 Specify the properties of your new Galaxy:

- In the **GR Node Name** list, type, or select the name of a computer that has the Galaxy Repository software installed. If necessary, click the **Browse** button to locate the node where the Galaxy Repository is installed.
- In the **Galaxy Name** box, type the name of the Galaxy you want to create within that Galaxy Repository. A Galaxy name can be up to 32 alphanumeric characters, including _ (underscore), \$, and #. The first character must be a letter. A Galaxy name cannot contain a blank space.
- In the **Galaxy Type** list, select the name of the already created backup Galaxy files, which are located at BackupGalaxies folder. The selected file is used as a template to create a new Galaxy. The system restores the selected backup Galaxy and renames it to the Galaxy name that you provided in the Galaxy Name box.

Note You cannot use the following reserved names as Galaxy names: Me, MyContainer, MyArea, MyHost, MyPlatform, MyEngine and System. You cannot use a name that conflicts with an existing object in the backup Galaxy file.

- 4** Click **Create**. The **Create Galaxy** dialog box opens, showing the Galaxy database being created.
- 5** After the Galaxy database is created, click **Close**. You are ready to open the Galaxy and begin creating your application. For more information about opening an existing Galaxy, see [Connecting to an Existing Galaxy](#) on page 19.

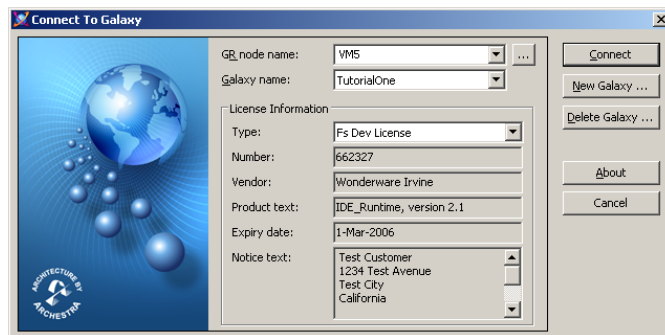
Connecting to an Existing Galaxy

Selecting an existing Galaxy lets you open a previously created Galaxy so you can work in it.

If security is enabled for an existing Galaxy, you cannot open it without logging in. If you do not have log on rights to a Galaxy, you cannot log in to that Galaxy. For more information about security, see *Working with Security* on page 231.

To connect to an existing Galaxy

- 1 On the **Start** menu, point to **Programs, Wonderware** and click **Archestra IDE**. The **Connect to Galaxy** dialog box appears.



- 2 Do the following:



- In the **GR node name** list, select the name of a computer you previously connected to. Click the **Browse** button to browse and select from the available domains and nodes on your network.
- In the **Galaxy name** list, select the name of the Galaxy on that GR node.
- Click **Connect**.

If the Galaxy you select has security enabled, the **Login** dialog box appears. Type your user name and password and click **OK**.

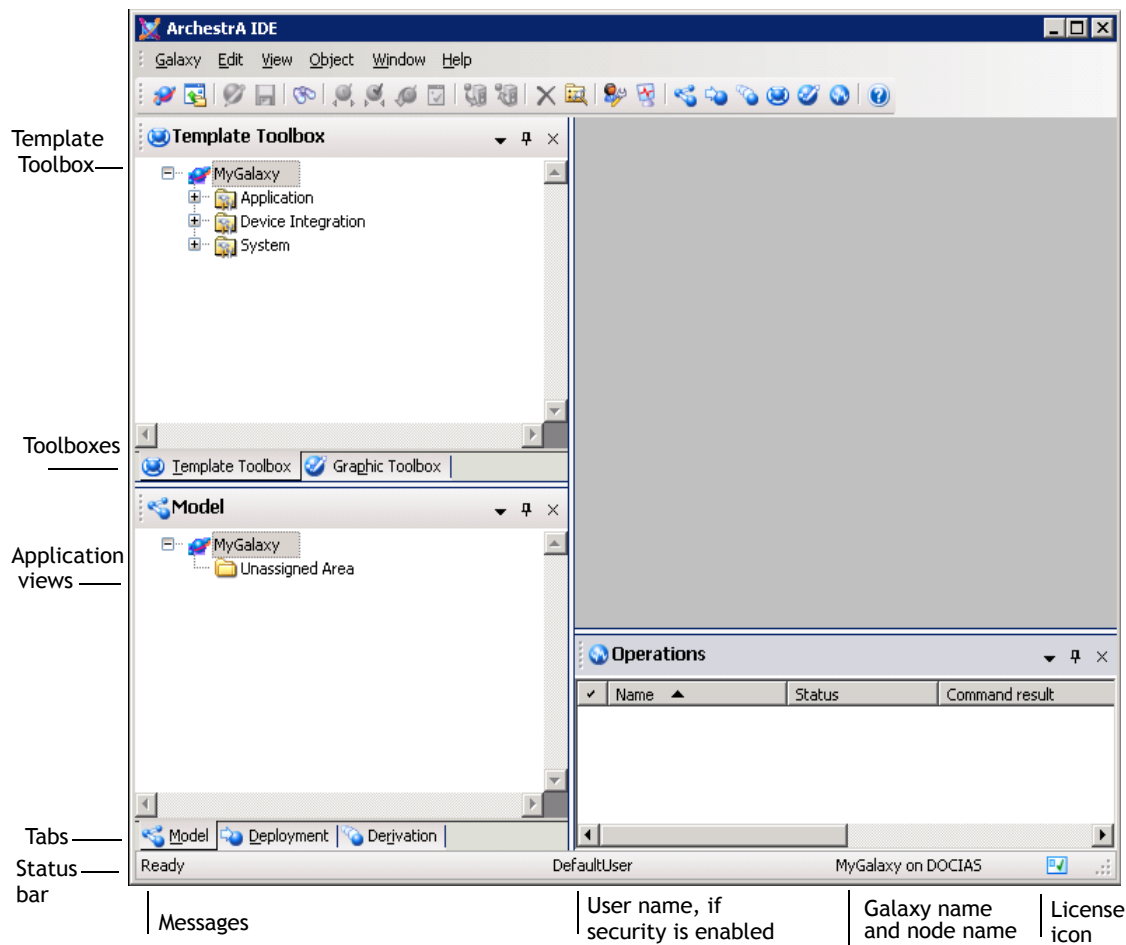
The Galaxy opens in the IDE. You are ready to start working with your Galaxy.

Getting Around the IDE

The IDE is the integrated design and development tool from which all ArcestrA objects are configured and deployed to target computers. You work from the IDE to create, configure, and maintain the objects that comprise your application and the underlying infrastructure that supports your application.

Using the ArcestrA IDE, you can import new types of objects in to the Galaxy Repository, configure new ones, and deploy them to computers in your network. Multiple users can work concurrently on different sets of objects from different ArcestrA IDEs.

After you open a Galaxy, the IDE opens and shows the different views of your Galaxy.



For a complete discussion of items like templates and instances, see About Templates and Instances on page 39.

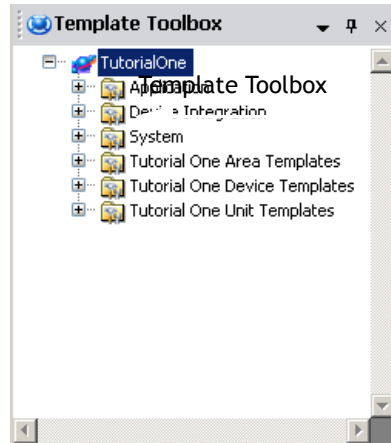
Views in the IDE include:

Template Toolbox	Expand the top level folders to see the different templates in the toolboxes.
Graphic Toolbox	Contains the global ArcestraA graphics that can be used in the Galaxy. For information, see the <i>Creating and Managing ArcestraA Graphic's User Guide</i> .
Application views	Click the tabs at the bottom to open: <ul style="list-style-type: none">• Model view - the object relationship to the automation scheme layout. The objects are organized into Areas that typically represent the physical plant layout.• Deployment view - the object relationship to the computers that comprise the deployed system that the objects run on.• Derivation view - the derivation path from base template to the instances. This View allows a user to see all object instances that were based on a given template. All templates and instances appear in this view.
Status bar	Shows messages, user name, Galaxy name and node, and license information. Turn off the Status bar by clicking Status bar on the View menu.

Using the Template Toolbox

The Template Toolbox lists template toolsets, which contain object templates. The Template Toolbox shows a tree view of template categories in the Galaxy. Double-click a category to expand the toolset and show the templates within it.

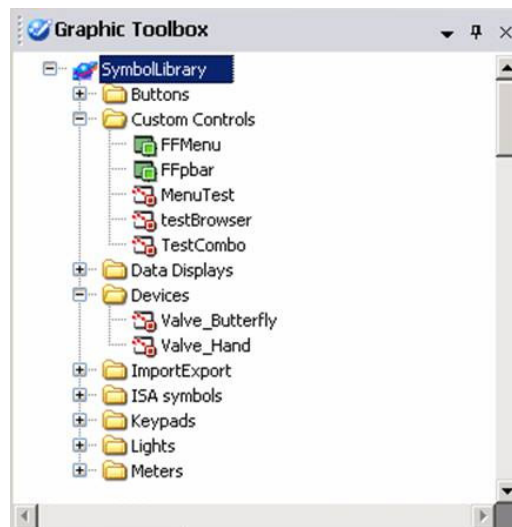
A new Galaxy is automatically populated with base templates.



Using the Graphics Toolbox

The Graphic Toolbox shows a treeview of toolsets that contains ArchestrA symbols and Clients Controls.

Double-click the toolsets to open them and reveal the graphics they contain. A new Galaxy is automatically populated with a library of Graphics organized in toolsets.



Using IDE Application Views

Base templates and non-base templates are included with Application Server. Non-base templates include: \$Boolean, \$Double, \$Float, \$Integer, and \$String. They are derived from \$FieldReference. The templates are automatically imported into the IDE when you first create a Galaxy.

Base templates appear in the Template Toolbox and in the Derivation view with a \$ as the first character of their name. You cannot directly modify base templates but you can use them to create your own objects or derived templates.

When you move from one view to another, the selected object in the first view is selected in the second view, if the object exists in that view. For example, templates are not shown in the Deployment view and Model view.

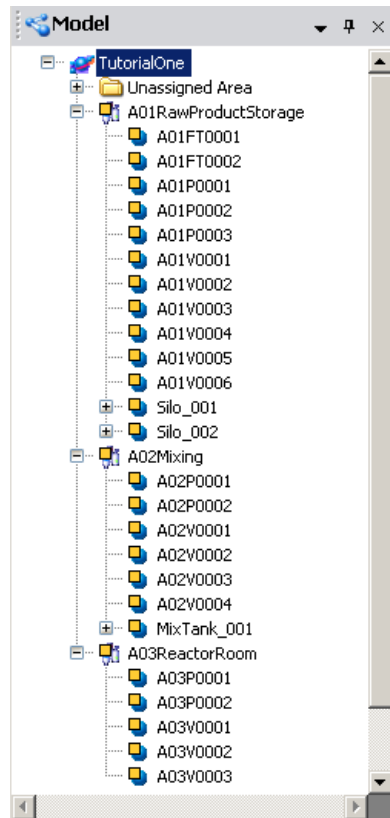
Model View

The Model view shows objects in terms of their physical or containment relationships, and allows you to organize them through a folder structure. This Model view most accurately represents an application perspective of the processes that users are emulating: for instance, specific process areas, tanks, valves, pumps and their relationships based on containment.

For more information about containment, see [Creating Derived Templates](#) on page 50.

Note You must undeploy an object before reassigning it to another object.

The tree structure acts like a standard Microsoft Windows Explorer tree. Initially, it shows a simple hierarchy: <Galaxy name> and the Unassigned Area folder.



In the Model view, all objects are grouped by areas and by containment relationship. The Model view shows these relationships in the following ways:

- The top of the tree is the Galaxy.
- Top-level Areas are shown under the Galaxy.
- Within each Area, contained Areas are listed. Areas support hierarchical composition; that is, they support sub-Area construction. Areas can be nested only 10 levels (after the sub-area is 10-levels deep, you cannot add another sub-level).

- Objects that belong to an Area are listed under the Area.
- Objects contained by other objects are listed under their respective containers. Multiple levels are allowed. For more information about containment, see [Creating Derived Templates](#) on page 50.

Note Objects belong to the same Area as the object that contains them.

Some object's hierarchical, or contained, names are truncated if you have multiple levels shown. To view the entire hierarchical name, select the object and click **Properties** on the **Galaxy** menu. The entire hierarchical name is shown in the **Properties** dialog box. For more information about hierarchical names, see [Using Contained Names](#) on page 58.

- Objects that currently do not belong to an Area are listed under Unassigned Area. Containment relationships between parent and child objects are shown there.

In each branch of the tree, objects are listed in alphabetical order. Default objects are shown in bold.



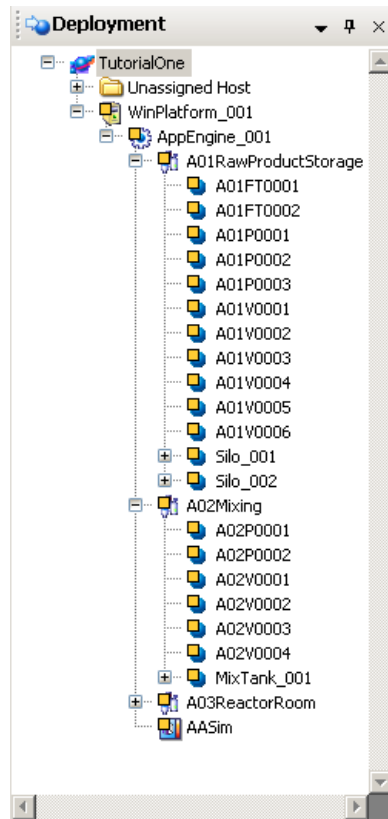
To assign an object to another, drag it onto the host object. If that object is an inappropriate assignment match, the international **Not** symbol appears. To unassign an object, drag it to the **Unassigned Host** folder.

Deployment View

The Deployment view shows instances only in terms of their assignment relationships. This view allows you to organize those objects through a folder structure.

This view shows which objects instances reside on which computers. In the ArchedrA environment, the physical location of object instances is not required to approximate the real-world environment it models. The Deployment view does not need to reflect your physical plant environment.

The tree structure acts like a standard Windows Explorer tree. It is initially divided into two hierarchical levels: <Galaxy Name> and the Unassigned Host folder.



In the Deployment view, objects appear in a tree according to their distribution relationships in a multi-node system in the following ways:

- The top of the tree is the Galaxy.
- WinPlatforms are shown under the Galaxy.
- Under each WinPlatform, assigned AppEngines are listed.
- Under each AppEngine, assigned Areas and DI Objects, such as DINetwork Objects, are listed.
- Under each Area, assigned ApplicationObjects are listed.
- Under each ApplicationObject, contained ApplicationObjects are listed. Multiple levels are allowed.
- Under each DINetwork Object, assigned DIDevice Objects are listed.
- Unassigned objects are grouped together in the **Unassigned Host** folder. Area and containment relationships are maintained in this view.

Important DINetwork objects have specific configuration limits such as whether more than one object can be deployed to a single WinPlatform. The IDE does not check for these limits. For more information about configuration limits, see the online help for the DINetwork object.

Under each branch of the tree, objects are listed in alphabetical order. Default objects are shown in bold.

For objects shown in any view, you see the following symbol in the corner of the object's icon:



Not deployed

[No
icon]

Deployed



Deployed, but pending configuration changes exist that have not been deployed.



Deployed, but software modifications exist that have not been deployed.



Applies only to redundant AppEngines. An AppEngine is undeployed, but its redundant pair is deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, but its redundant pair is not deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, its redundant pair is not deployed, and pending configuration changes exist that have not been deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, its redundant pair is not deployed, and software modifications exist that have not been deployed.

[No
icon]

Good



Warning



Error. The object is in an Error state and cannot be deployed.



InTouchViewApp application files are being asynchronously transferred to the target node. This icon is normally visible for only a few moments at the end of an InTouchViewApp deployment operation, unless the object is deployed on a slow network. This icon completely replaces the original while it is shown.



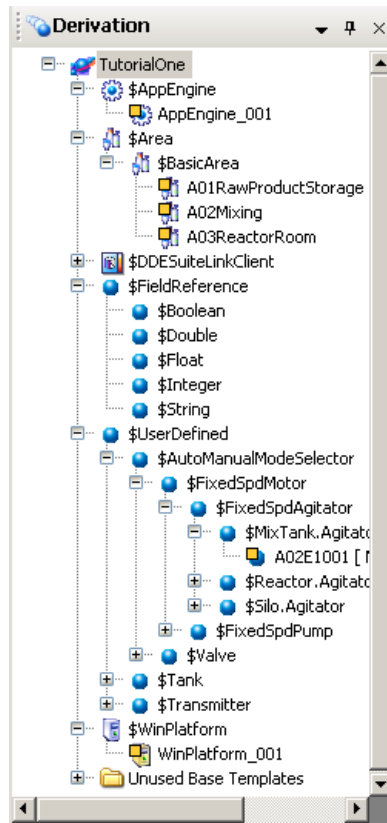
To assign an object to another, drag it onto the host object. If that object is an inappropriate assignment match, the international **Not** symbol appears. To unassign an object, drag it to the **Unassigned Host** folder.

Note You must undeploy an object that is currently deployed before reassigning it from one object to another.

Derivation View

The Derivation view shows objects and templates in terms of their parent/child relationship. An object derived from another object appears in a hierarchy level under it.

The tree structure acts like a standard Windows Explorer tree, and initially is divided into three hierarchical levels: <Galaxy Name>, <Used Base Templates>, and the Unused Base Templates folder.



In the Derivation view, objects appear according to their parent-child relationship in the following ways:

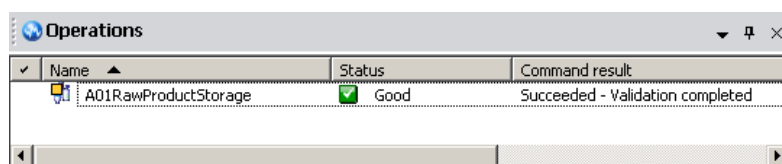
- The top of the tree is the Galaxy.
- Base templates with associated child objects, either derived templates or instances, are shown under the Galaxy.
- Under each base template, derived templates and instances created from the base template are listed. Multiple levels are allowed. Instances created from derived templates are listed under their parents.
- Templates with no associated derived templates or instances are grouped together in the **Unused Base Templates** folder.

Objects with names that start with a “\$” are templates, either base or derived. Under each branch of the tree, child objects are listed in alphabetical order.

As in other views, dragging one object onto another in the **Derivation view** associates the two objects based on the predefined rules of the object types. For example, you can drag ApplicationObjects onto other ApplicationObjects but you cannot drag ApplicationObjects to an Engine.

Operations View

The **Operations** view shows the results of validating the configuration of objects. You may need to open it before you see it. On the **View** menu, click **Operations**.



During the validation of an object, its icon and name appear with the status of the operation.

Important You can validate both templates and instances if they are checked in.

The status of the object (**Status** column) is shown with an icon and a descriptive word or phrase.

When validation is complete, the **Command Result** column shows a “Succeeded” or “Failed” message and additional information about the validation results. For more information about validating objects, see Validating Objects on page 91.

Note You can validate all objects in the Galaxy by running the Validate operation on the Galaxy object. In that case, Command Result messages are shown after all objects in the Galaxy are validated.

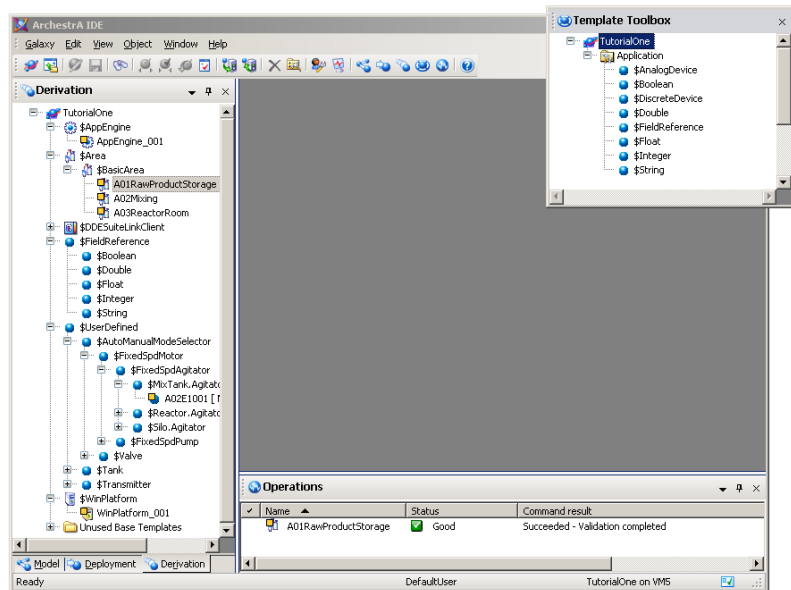
The **Operations view**, like the **Template Toolbox** and **Applications views**, is also updated as the status and conditions of objects in the Galaxy change.

Customizing Your Workspace

You can customize your workspace by docking and floating the IDE's views. You can also hide some of the views.

Docking Views

To dock a view, drag the view to the location you want it. For example, drag the title bar of the Template Toolbox and dock it under the Application views. You can also undock it by dropping it anywhere on your desktop. Drag it back to dock it again.



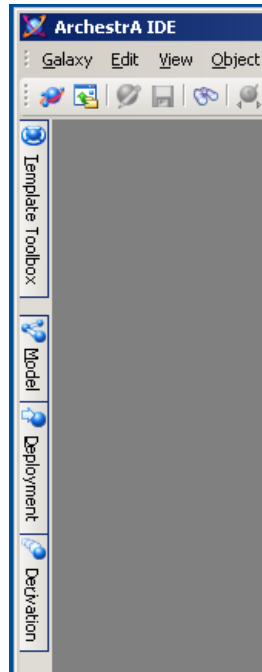
Floating Views

- ▼ You can also float a view. With your cursor over the view you want to float, click the arrow. On the menu that appears, click **Floating**.
- ▼ You can also float just one view. To float the **Derivation** view, click the arrow. On the menu that appears, click **Floating**. The view floats on your desktop. You can move it to another location or dock it.

Hiding Views



To hide a view, click the **Pin** icon. The views “hide” as tabs on the side of the window. The Operations view hides at the bottom of the window.



When you move the mouse over the tab, the view expands into the workspace.

Resetting the Workspace

You can easily reset the IDE workspace and restore views back to their default locations.

To return the views to the default docking

- ◆ On the **View** menu, click **Reset Layout**.

Synchronizing the Views

You can specify that a selected object stay selected as you move through the views. In any of the views, select the object you want to synchronize. On the **View** menu, click **Synchronize Views**. Now as you move from one view to another, that object stays selected.

Configuring User Information

You can configure options for a Galaxy, including prompts for check-in comments and user defaults. These user options apply to the currently open Galaxy and do not change options for other Galaxies.

If you specify a security group, that security group must already exist. For more information about security and security groups, see [Configuring Security](#) on page 237.

To configure user information

- 1 On the **Edit** menu, click **User Information**. The **Configure User Information** dialog box appears.

The screenshot shows the 'Configure User Information' dialog box with the following settings:

- Prompts:**
 - Ask for Check In Comments
 - Warn before launching an editor for a read-only object
 - Warn for insufficient permissions
 - Warn before launching an InTouchViewApp editor
- Initial scan state for deployed objects:**
 - On Scan
 - Off Scan
- Scan state defaults:**
 - Force Off Scan
 - Don't Force Off Scan
- Auto context selection:**
 - Automatically synchronize the current single object selection when opening a new view
- User Defaults:**
 - Platform:
 - Application Engine:
 - Area:
 - View Engine:
 - Security Group:

Buttons: OK, Cancel

- 2 In the **Prompts** area, do one or more of the following:
 - Select the **Ask for 'Check In' Comments** check box if you want to be prompted to type comments when checking in templates and objects.
 - Select the **Warn before launching an editor for a read-only object** check box if you want to see a prompt that informs you if you are opening an instance or template as read-only. The prompt warns you if you open an instance or template while someone else is working on it. If someone else is working in the instance or template, you cannot make changes.

- Select the **Warn for insufficient permissions** check box if you want to see a prompt that tells you if you have permission to create or modify InTouchView applications. These permissions authorize or prevent you from creating and modifying InTouch View Applications.
 - Select the **Warn before launching an InTouchViewApp editor** check box if you want to see a prompt each time you attempt to edit an InTouchView application instance. You can edit the associated template or cancel the operation. If you do not select this check box, the request to edit the InTouchViewApp instance is automatically redirected to the associated template.
- 3 Select the **Initial scan state for deployed objects**. You can select **On Scan** or **Off Scan**. You can change this setting on an individual basis in the **Deploy** dialog box when you deploy. For more information, see *Deploying and Running an Application* on page 131.
 - 4 Select the **Scan state defaults** when undeploying or redeploying instances. **Force Off Scan** will attempt to take the target object offscan when an already deployed object is redeployed. **Don't Force Off Scan** does not force the target to go off scan when you deploy.

Note Redeployment of objects that are currently deployed on-scan will be cancelled unless this option is selected.

- 5 Make your **Auto context selection**.
- 6 In the **User defaults** area, provide the following object names of Framework objects you select to be defaults with respect to assignment relationships.
 - Type the **Platform** name.
 - Type the **Application Engine** name.
 - Type the **Area** name.
 - Type the **View Engine** name.
 - Type your **Security Group** name, if any.
- 7 When you are done, click **OK**.

Logging on and Logging off

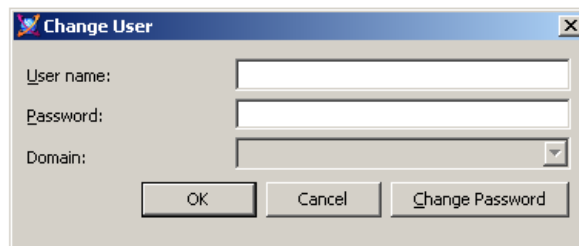
Some Galaxies have security associated with them. If you try to open a Galaxy with security, you need to log on to the Galaxy to open it.

Note If you do not have logon rights, you cannot open a Galaxy.

For information about setting up security in the ArchestrA environment, see *Working with Security* on page 231.

To log on to a Galaxy

- 1 When you open a security enabled Galaxy, the **Change User** dialog box appears.



- 2 Type your **user name** and **password**. If OS authentication security is enabled for the Galaxy, you must select the **Domain** on which your user account is located. If the list is unavailable, the selected Galaxy is on your local computer.
You can change your password after you type your user name and password. Click **Change Password**. Type the new password and then retype it.
- 3 Click **OK**. Your logon data is validated by the Galaxy Repository being accessed. Depending on operating system security, the IDE opens. If the GR does not recognize your user name or password, you are prompted to enter them.

Changing Users

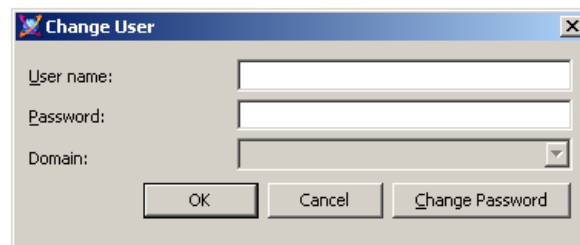
You can change users in a Galaxy at any time. Any security restrictions associated with a user change when the user logs on or logs off from the Galaxy. For more information about users and security, see *Configuring Security* on page 237.

If the Galaxy has not be configured to enable security, you see a message. All users in an open security environment are treated as the `DefaultUser` by the Galaxy. This means all users have full access to everything.

To change users

- 1 On the **Galaxy** menu, click **Change User**.

If security is enabled on the Galaxy, the **Change User** dialog box appears.



- 2 Enter your logon information and click **OK**.
 - If needed, click **Change Password** to change the password for the new user.
 - Type the new password and then retype it.
- 3 Click **OK**.

Chapter 2

Getting Started with Objects

Before you start modeling your application using the Application Server, you must understand templates and object instances.

Templates are elements in Application Server that contain common configuration parameters for objects instances that you use multiple times in your application.

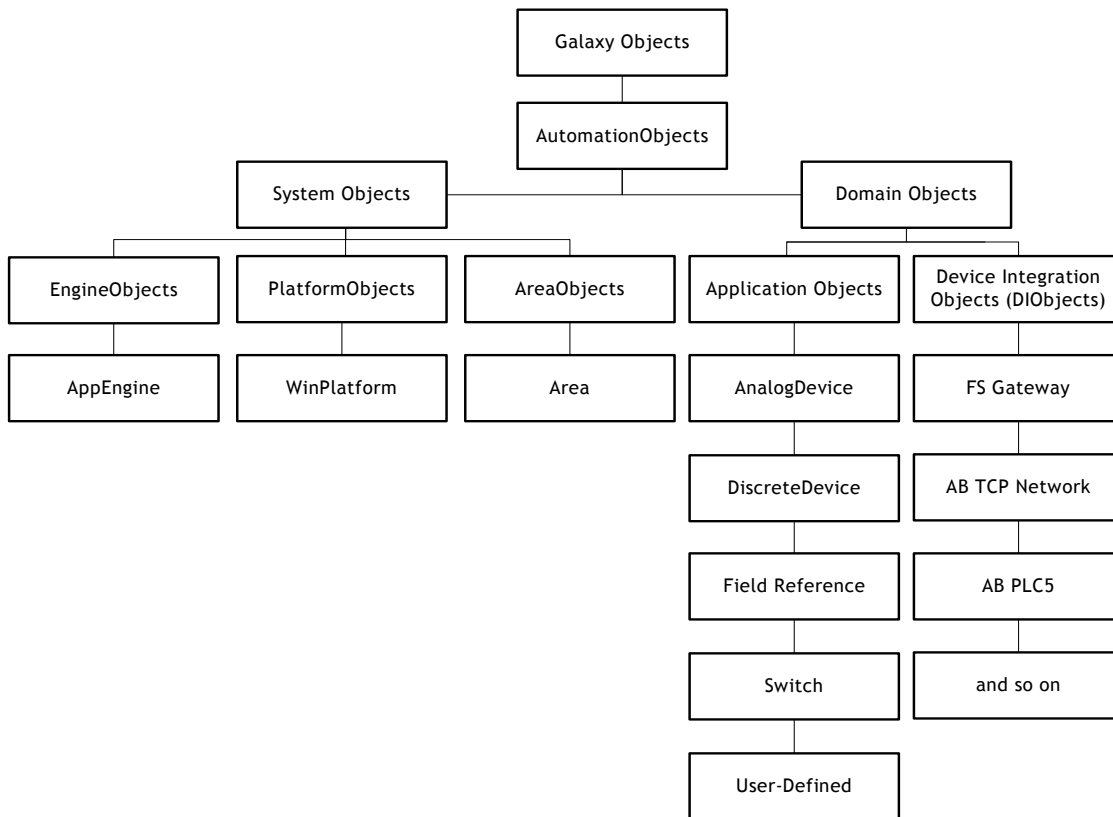
For example, you might create a template for valves. You configure the template with all the unique attributes for valves. You use that template to make object instances of valves. You can further configure and customize each object instance to represent a specific valve.

Object instances are the specific devices in your environment, such as diaphragm valves or very complex devices, like a reactor. You create an instance from a template and then customize the specific instance as needed.

Instances are deployed to the run-time environment. Templates exist in the development environment and cannot be deployed.

Creating templates and instances is very similar to object-oriented programming. For example, templates and instances have a parent/child relationship that involves inheriting attributes. There are differences, however, between object-oriented programming and creating templates and instances in Application Server.

Collectively, templates and instances are called objects. The following graphic shows the different kinds of objects and how they are organized.



If you are new to this kind of programming, the next section explains the basic concepts you need to know before you start. If you are familiar with object-oriented programming, the concepts in the next section may be familiar to you, but notice the important differences between object-oriented programming and Application Server.

About Templates and Instances

Understanding templates and instances is critical to working with Application Server.

Instances

Instances are the run-time objects created from templates in Application Server. Instances are the specific things in your environment like processes, valves, conveyer belts, holding tanks, and sensors. Instances can get information from sensors on the real-world device or from application logic in Application Server. Instances exist during run time.

In your environment, you may have a few instances or several thousand. Many of these instances may be similar or identical, such as valves or holding tanks. Creating a new valve object from scratch when you have several thousand identical valves is time-consuming. That's where templates come in.

Templates

Templates are high-level definitions of the devices in your environment. Templates are like a cookie cutter from which you can make many identical cookies.

You define a template for an object, like a valve, one time and then use that template when you need to define another instance of that item. Template names have a dollar sign (\$) as the first character of their name.

A template can specify application logic, alarms, security, and historical data for an object.

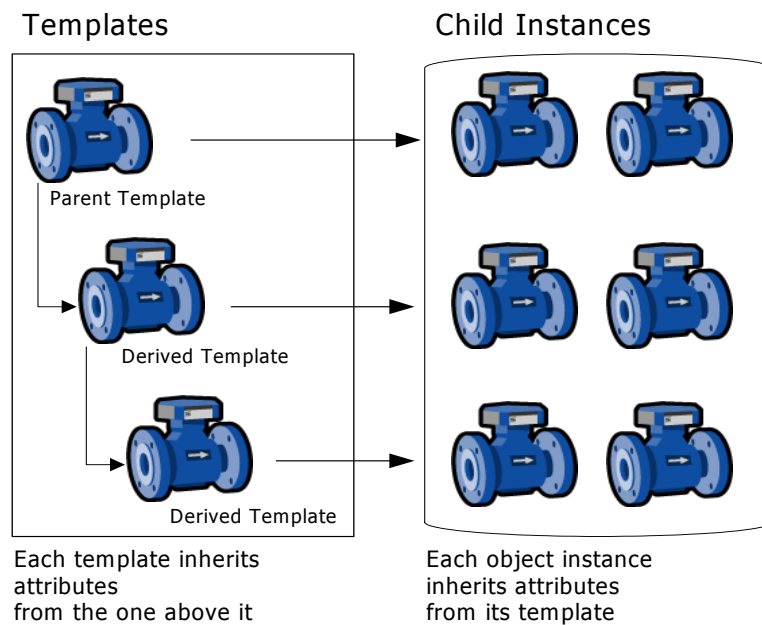
A template can also define an area of your environment. You can extend and customize a template by adding User Defined Attributes (UDAs), scripts, or extensions to meet the specific needs of your environment. Objects inherit attributes from their parents.

Application Server comes with predefined templates, called base templates. You cannot modify base templates. All templates you create are derived from base templates.

You can also nest templates, or contain them. Contained templates consist of nested object templates that represent complex devices consisting of smaller, simpler devices, including valves. A reactor is a good candidate for containment.

Templates only exist in the development environment.

Using the Diaphragm valve template, you can quickly create a Diaphragm valve instance when you need another Diaphragm valve in your application.

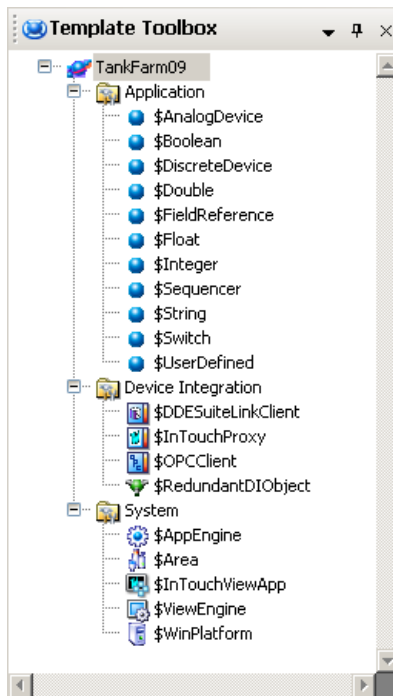


Propagation

If you need to change something about all diaphragm valves, you can change the template for the Diaphragm valve and all diaphragm valves in your application inherit the changes, assuming the attributes are locked in the parent template. This makes it easy to maintain and update your application.

About Base Templates

When you first open the IDE, you see the base templates in the **Template Toolbox**.



You cannot modify base templates. You use base templates to create derived templates, which are copies of the base templates. You modify your derived templates and create instances of them for your applications.

The template classes are as follows:

- Application templates

Use these templates to represent real devices in your Galaxy. These devices represent real objects in your environment. For example, use the DiscreteDevice base template to create a derived template for valves.

- Device integration templates

Use these templates to create instances that communicate with external devices. For example, use the DIOObject base template to create a derived template for a PLC device.

- System templates

Use these templates to define system instances, like other computers.

Application Templates

These base templates let you easily create devices in your Galaxy. They contain the properties you need to set for each kind of device. For example, a DiscreteDevice device contains all the settings you need to specify for an on/off device. Of course, using UDAs, scripts, and extension, you can extend and customize any device you select.

Device Integration Templates

These base templates represent the communication with external devices. External devices run on the application engine.

For example:

- DINetwork object – Refers to the object that represents the network interface port to the device through the Data Access Server. The object provides diagnostics, and configuration for that specific card.
- DIDevice object – Refers to the object that represents the actual external device (such as a PLC or RTU), which is associated to the DINetwork Object.

System Templates

These objects represent the parts of a Galaxy and not the domain they are monitoring/controlling. These base templates let you create more system level grouping and computers, such as areas you add objects to or another host AppEngine.

WinPlatform Object

The WinPlatform platform object is a key base object because you need a platform to host the objects you are modeling.

This object:

- Calculates various statistics for the node it is deployed to. These statistics are published in attributes.
- Monitors various statistics related to the node it is deployed to. These monitored attributes can be alarmed and historized.
- Start and stop engines, based on the engines startup type which are deployed to it.
- Monitor the running state of engines deployed to it. If the platform detects an engine failed, it can, optionally based on the value of the engine's restart attribute, restart the engine.

AppEngine Object

The AppEngine object must have a Platform on which to run. This object:

- Hosts ApplicationObjects, device integration objects and areas.
- Contains the logic to set up and initialize objects when they are deployed.
- Contains the logic to remove objects from the engine when they are undeployed.
- Determines the scan time which all objects within that particular engine run.

Area Object

All application objects belong to an area. Areas can contain sub-Areas. Areas provide a key organizational role in grouping alarm information and providing that information to those who use alarm/event clients to monitor their areas of responsibility.

The values of three Area object alarm attributes can be saved to the historian:

- Active alarm counter
- Unacknowledged alarm counter
- Disabled (or silenced) alarm counter

ViewEngine Object

The ViewEngine object must have a Platform on which to run. This object:

- Hosts InTouchViewApp objects.
- Contains the logic to set up and initialize objects when they are deployed.
- Contains the logic to remove objects when they are undeployed.
- Determines the scan time which all objects within that particular engine run.

InTouchViewApp Object

The InTouchViewApp object must have a ViewEngine on which to run. This object:

- Manages the synchronization and delivery of files required by the associated InTouch application.
- Provides run-time access to tags on the associated InTouch application.
- Starts WindowMaker for the associated InTouch application when edited.

About Derived Templates

All templates you create within the IDE are derived templates.

When creating your Galaxy application, plan ahead and create derived templates for devices of a certain type so you can use the templates to create instances from.

A new derived template is an exact copy of its parent template with the possible exceptions of locking and security and modified attribute values. You can lock attributes to prevent them from being modified in child templates.

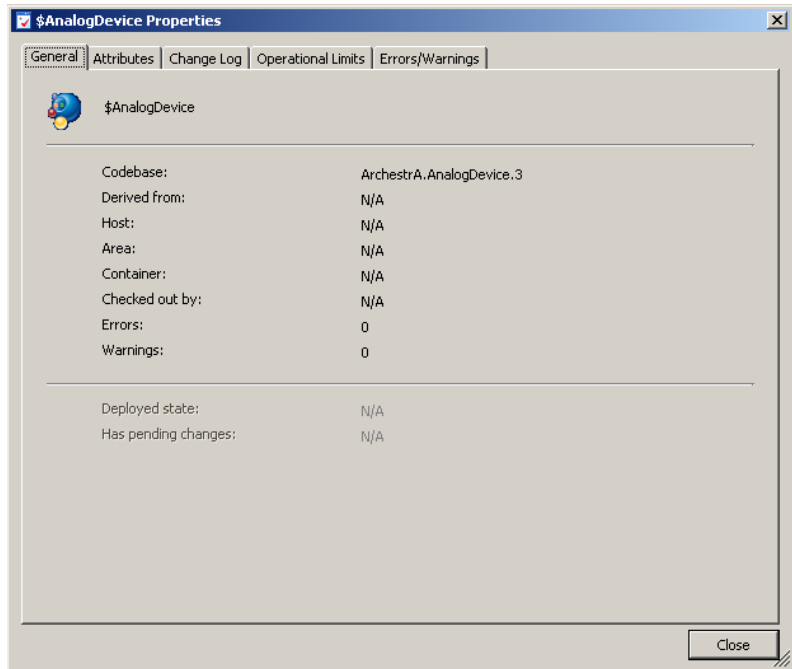
After you create a new derived template, you can customize it. For more information about customizing and extending templates, see [Creating Derived Templates](#) on page 50.

Every template has a set of attributes and default values. When you create an instance, attributes are inherited by the instance. In the instance, you can reconfigure many of the attributes inherited from the parent template if they are not locked on the parent template.

For more information about customizing instances, see [Working with Objects](#) on page 47.

Viewing Object Properties

You can view the properties of an object by right-clicking and clicking **Properties**. Object properties vary, depending on the type of selected object and whether it is a base template, a derived template, or an instance.



For more information about specifying the properties of objects, see [Working with Objects](#) on page 47.

Chapter 3

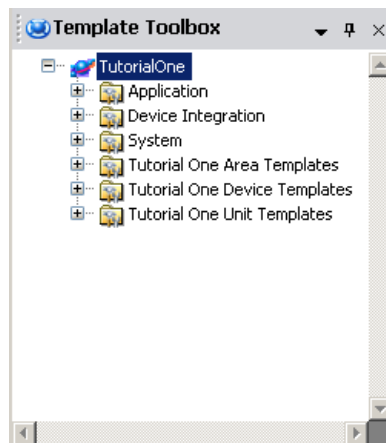
Working with Objects

You can work with objects using the Application Server development environment. Both templates and instances are collectively referred to as objects. For more information about what templates and instances are, see [About Templates and Instances](#) on page 39.

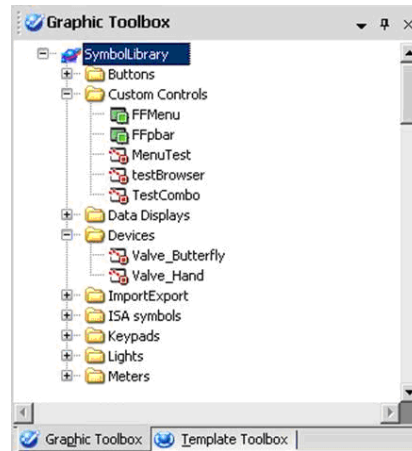
Managing Toolsets

Toolsets are high-level folders shown in the Application Toolbox. You can create toolsets to store the templates and graphics used in your Galaxy. You can also create toolsets within toolsets.

Use the Template Toolbox to view and organize object templates.



Use the Graphic Toolbox to view and organize Archestra Symbols.



You can move content between their respective toolsets. You can also show or hide toolsets to make the workspace less cluttered.

Creating Toolsets

When you create your own toolset, it must have a unique name. Toolset names are not case sensitive, so `Valves` is the same name as `valves`. A toolset name can be up to a maximum of 64 alphanumeric and special characters, including spaces, except \$.

To create a new toolset in the Toolbox

- 1 On the **Galaxy** menu, point to **New** and click either **Template Toolset** or **Graphic Toolset**.
- 2 Type a name for the new toolset.

A new toolset appears and is in focus. Now, you can drag templates into the new Template toolset, or you can drag graphics into the Graphics toolset.

Creating Child Toolsets

Toolsets can be created within existing toolsets. Nested toolsets help in further organizing templates and graphics. You can create a maximum of ten levels of toolset folders.

To create a child toolset

- 1 Select the parent toolset.
- 2 On the **Galaxy** menu, point to **New** and click either **Template Toolset** or **Graphic Toolset**.
- 3 Type a name for the new toolset.
A new toolset appears beneath the parent toolset and is in focus.
- 4 Drag templates into the new child Template toolset, or you can drag graphics into the Graphics toolset.

Deleting Toolsets

You can delete toolsets you no longer want or need. Before you start, make sure you move or delete all content from the toolset.

The toolset you want to delete must be empty, or it cannot be deleted.

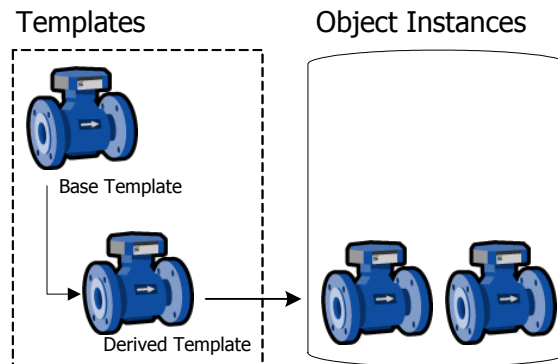
To delete a toolset

- 1 Select the toolset you want to delete.
- 2 On the **Edit** menu, click **Delete**.
- 3 Click **Yes** to delete the toolset.

Creating Derived Templates

All templates you create are derived templates. A derived template inherits attributes and behaviors from the parent template. You cannot change the attributes in a base template.

After you create a derived template, you can customize and modify its attributes. If you change locked attributes in the parent template, the changes propagate to the derived template.



After you create derived templates, you can create instances of the templates. You can change and modify unlocked attributes in the instances, making adjustments to meet the needs of the specific object you are modeling.

For example, your plant processes can use several models of a pump made by a single vendor. Each pump model has unique characteristics that map to different attribute values of the DiscreteDevice base template.

After you create a derived template, you can customize it.

To derive a template from another template

- 1 Select the base template to use as the parent template in the **Template Toolbox** or **Derivation** views pane.
- 2 On the **Galaxy** menu, click **New** and click **Derived Template**. A derived template is created in the same toolset as its parent and placed in name edit mode. The default name is the same as the parent template followed by a numeric sequence.

- 3 Rename the derived template, if needed. Template names can be up to 32 alphanumeric or special characters, including the required \$ as the first character. The second character cannot be \$ and the name must include at least one letter. Template names cannot contain spaces.

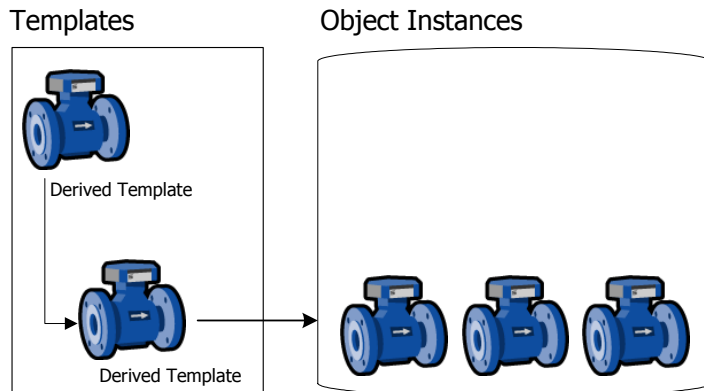
Note You cannot use the following reserved names as template names: Me, MyContainer, MyArea, MyHost, MyPlatform, MyEngine and System.

- 4 You are ready to customize your new template. For more information, see [Editing Objects](#) on page 63.

Deriving Templates from Another Derived Template

You can create derived templates from other derived templates. The child template inherits attributes from all parent templates. Any changed attributes in the immediate parent overrides attributes changes in grandparent levels.

If you change locked attributes in the parent template, the locked attributes propagate to the derived template.



A derived template is an exact copy of its parent with the exceptions of locking, security, and the unlocked attributes that have been edited. If you create a new derived template from an existing container template, the new derived template has the same contained templates.

A good practice is to create a hierarchy of derived templates until you reach logical endpoints. Then create instances from each unique derived template.

To create a derived template from a derived template

- 1 In the **Template Toolbox** or **Derivation view**, select the derived template you want to use as the parent template.
- 2 On the **Galaxy** menu, click **New** and click **Derived Template**. A derived template is created in the same toolset as its parent. You can edit the name of the new derived template. The default name is the same as the parent template followed by a numeric sequence.
Template names can be up to 32 alphanumeric or special characters, including the required \$ as the first character. The second character cannot be \$ and the name must include at least one letter. Template names cannot contain spaces.

Note You cannot use the following reserved names as template names: Me, MyContainer, MyArea, MyHost, MyPlatform, MyEngine and System.

- 3 You can create another derived template by repeating the steps above, or you can customize your new derived template. For more information about customizing your template, see *Editing Objects* on page 63.

Creating Contained Templates

Containment is the relationship in which one object includes another. Containment relationships organize objects in a hierarchy. You can build objects that represent complex devices consisting of smaller, simpler devices.

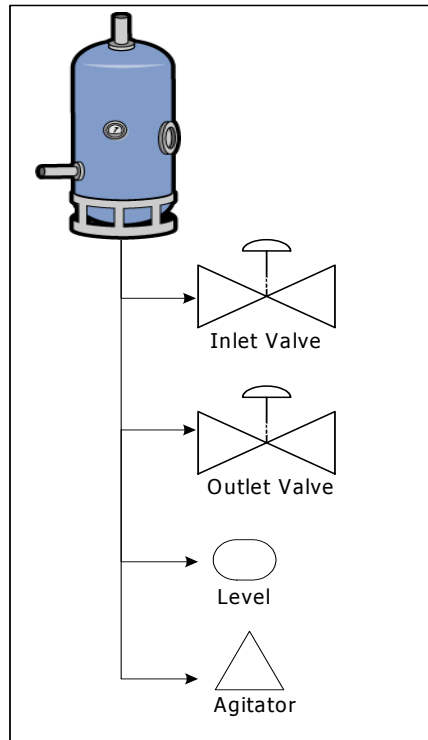
In scripts, these objects can be referred to by the name that derives from the containment relationship. This name is called a hierarchical name.

An object can have three kinds of names if it is contained by another object. The three names include:

Name	Description
Tag Name	The unique name of the individual object. For example, <code>Valve1</code> .
Contained Name	The name of the object within the context of its container object. For example, the object whose Tag name is <code>Valve1</code> may also be referred to as <code>Tank1.Outlet</code> , if <code>Tank1</code> contains it and it has the contained name "Outlet".
Hierarchical Name	<p>Hierarchical names that are fully-qualified names of a contained object include the name of the objects that contain it.</p> <p>Because the object that contains it may also be contained, there are potentially multiple hierarchical names that refer to the same object.</p> <p>For example, if:</p> <p>"Reactor1" contains <code>Tank1</code> (also known within <code>Reactor1</code> by its contained name "SurgeTank").</p> <p>"Tank1" contains <code>Valve1</code> (also known within <code>Tank1</code> by its contained name "Outlet").</p> <p><code>Valve1</code> could be referred to as:</p> <p>"Valve1"</p> <p>"Tank1.Outlet"</p> <p>"Reactor1.SurgeTank.Outlet".</p>

Note Base templates cannot be contained by another template, either as the container or as the template being contained. You can only use containment with derived templates.

Higher level objects contain lower level objects. This allows you to more closely model complex plant equipment, like tank systems. You can nest templates to 10 levels.



Note Objects can only contain objects like themselves. For example, ApplicationObjects can only be contained by other ApplicationObjects. Areas can only contain other Areas.

ApplicationObject Containment

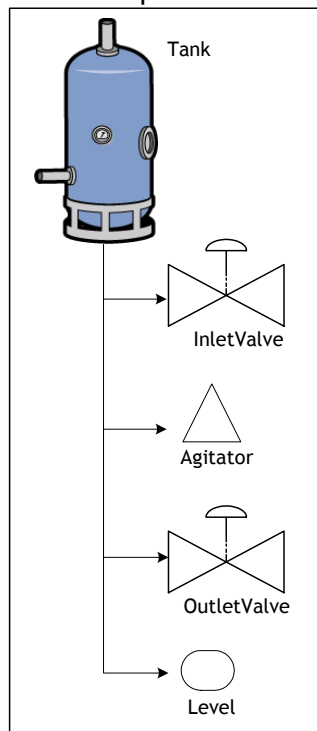
ApplicationObjects can be contained by other ApplicationObjects. This provides context for the contained object and a naming hierarchy that provides a powerful tool for referencing objects.

Note Base templates cannot be contained by another template, either as the container or as the template being contained. You can only use containment with derived templates.

An example of a containment hierarchy is a tank that contains the following objects:

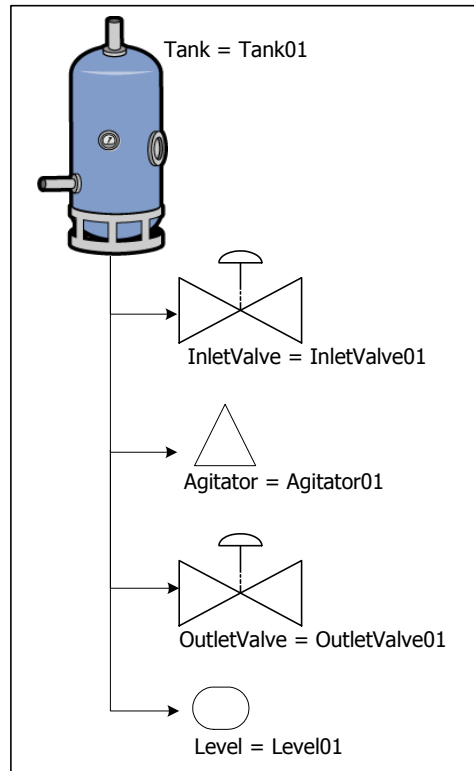
- Inlet Valve
- Agitator
- Outlet Valve
- Level

Tank Template



To enable referencing and flexibility within scripting, these objects can be referenced in several different ways. Each object has a unique tag name, such as:

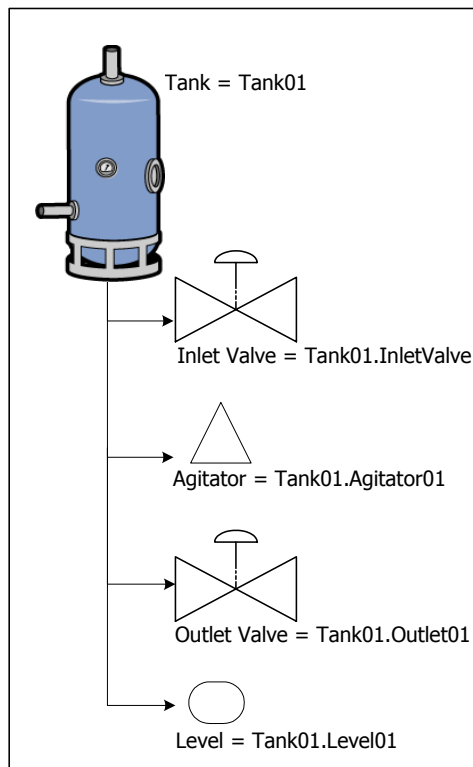
- Inlet Valve = InletValve01
- Agitator = Agitator01
- Outlet Valve = OutletValve01
- Level = Level01



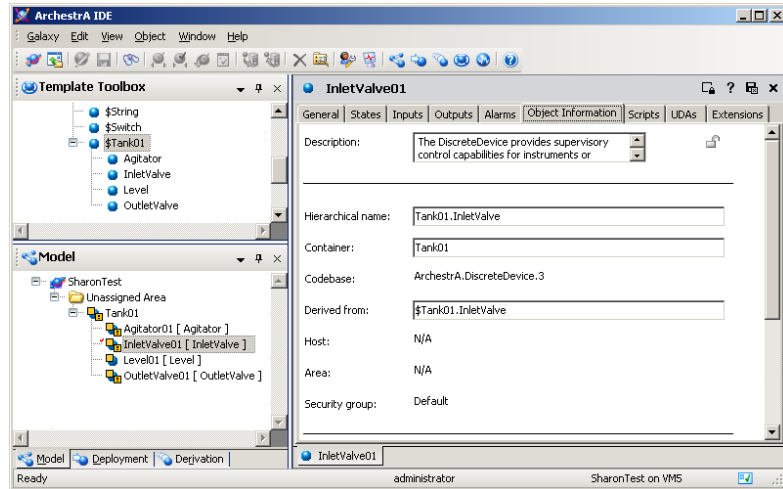
Within the context of each hierarchy, the contained names are unique, in that the names only refer to this tank system and the contained objects.

So if the tank is named `Tank01`, the contained names are:

- `Tank01.Inlet`
- `Tank01.Agitator`
- `Tank01.Outlet`
- `Tank01.Level`



This naming convention adds context to the instances contained by Tank01.



Additionally, you can use containment references in scripts such as:

- `Me.Outlet`: Allows a script running within the parent object to generically reference its child outlet instance.
- `MyContainer.Inlet`: Allows a script running in any of the children instances to reference another child instance named `Inlet` that belongs to the same parent.

Using Contained Names

The contained name of a contained object only has to be unique in the context of its container.

An object can have three kinds of names, depending if it is contained by another object. The three names include:

Name	Description
Tag name	The unique name of the individual object. For example, <code>Valve1</code> .
Contained name	The name of the object within the context of its container object. For example, the object whose tag name is <code>Valve1</code> may also be referred to as <code>Tank1.Outlet</code> , if <code>Tank1</code> contains it and it has the contained name "Outlet".

Name	Description
Hierarchical name	<p data-bbox="891 268 1403 401">Hierarchical names that are fully-qualified names of a contained object include the name of the objects that contain it.</p> <p data-bbox="891 422 1403 554">Because the object that contains it may also be contained, there are potentially multiple hierarchical names that refer to the same object.</p> <p data-bbox="891 575 1094 602">For example, if:</p> <p data-bbox="891 623 1403 716">"Reactor1" contains Tank1 (also known within Reactor1 by its contained name "SurgeTank").</p> <p data-bbox="891 737 1403 829">"Tank1" contains Valve1 (also known within Tank1 by its contained name "Outlet").</p> <p data-bbox="891 850 1284 877">Valve1 could be referred to as:</p> <p data-bbox="891 898 1003 926">"Valve1"</p> <p data-bbox="891 947 1089 974">"Tank1.Outlet"</p> <p data-bbox="891 995 1279 1022">"Reactor1.SurgeTank.Outlet".</p>

For example, an instance of a \$Tank is named Tank01. An instance of \$Valve called Valve01 is contained within the instance of \$Tank.

Change the contained name of Valve01 to InletValve. Now Valve01 can also be referred to by its hierarchical name Reactor1.InletValve. The name of the contained object can be changed, though, within the scope of the hierarchy.

Contained names can be up to 32 alphanumeric or special characters. The second character cannot be \$ and the name must include at least one letter. You cannot use spaces.

Containment Examples

You can have a Tank object that contains two DiscreteDevice objects that represent its inlet and outlet valves.

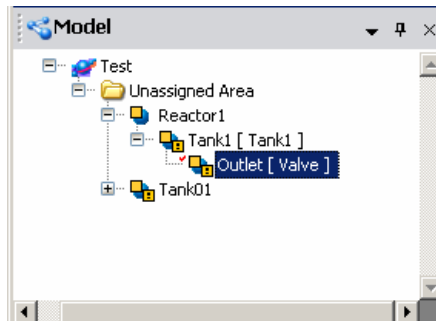
Note Base templates cannot be contained by another template, either as the container or as the template being contained. You can only use containment with derived templates.

To implement containment

- 1 Create the following instances: Tank1 from \$UserDefined and Valve from \$DiscreteDevice. Valve has only one name, Valve.
- 2 In the **Model** or **Deployment view**, drag Valve on to Tank.

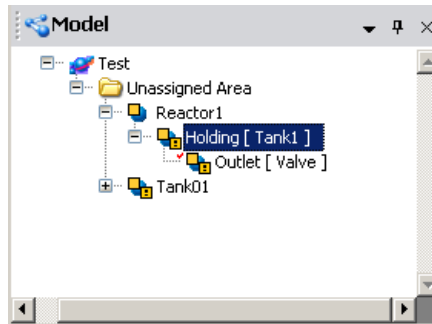
Note If Tank1 already contains an object with a contained name of Valve, the Galaxy generates a unique contained name for the newly contained object, such as Valve_001.

- 3 Change the contained name of Valve within Tank1 to Outlet. Valve can now be referred to by its tagname, Valve, as well as its hierarchical name, Tank1.Outlet.
- 4 Create an instance called Reactor1 from \$UserDefined.
- 5 In the **Model** or **Deployment view**, drag Tank1 onto Reactor1.



- 6 Change the contained name of Tank1 to Holding. Tank1 now has two names, Tank1 and Reactor1.Tank. Also, Valve1 has a three-part hierarchical name: Reactor1.Tank.Outlet.

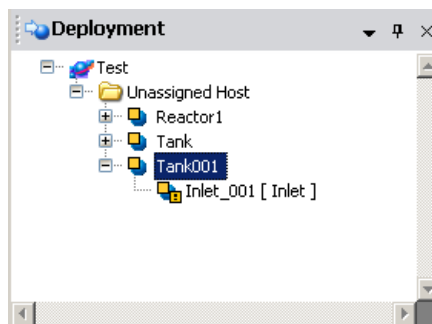
For the three objects in this example (Reactor1 containing Tank1 containing Valve1), the following naming hierarchy exists:



To implement template-level containment

Note Contained Templates do not have tagnames. When an instance hierarchy is created from a template and its contained children, unique tagnames will be created for the instances based on their contained names.

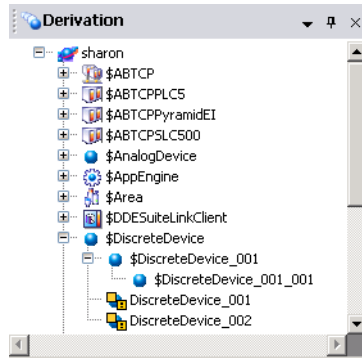
- 1 Create the following derived templates: \$Tank from \$UserDefined and \$Valve from \$DiscreteDevice.
- 2 Derive \$Inlet from \$Valve.
- 3 In the **Template Toolbox**, drag \$Inlet on to \$Tank. If \$Tank already contains a template named Inlet, the Galaxy generates a unique tagname for the new contained template, such as Inlet_001. The contained template now has a hierarchical name \$Tank.Inlet.
- 4 Create an instance (Tank1) of \$Tank.
- 5 The **Model** and **Deployment** views show an instance Tank1 that contains an instance called Inlet.



Viewing Containment Relationships

Containment relationships appear for templates in the **Template Toolbox**. For instances, the relationship appears in both the **Model** and **Deployment views**.

In the **Derivation** view, if a template contains other templates, you can expand it to show the containment under that template.



The **Derivation view** shows templates and instances with regard to containment in the following ways:

- Non-contained instances show their tagnames.
- Contained instances show their tagnames and hierarchical names.
- Non-contained templates show their template name.
- Contained templates show their hierarchical name.

Renaming Contained Objects

Before you rename a contained name of an object, make sure that the object is not checked out to another user or currently deployed.

The new contained name must comply with naming restrictions. Template names can be up to 32 alphanumeric or special characters, including the required \$ as the first character. The second character cannot be \$ and the name must include at least one letter. You cannot use spaces.

Contained names also cannot be the same contained name as an existing contained object within the same level of hierarchy in the containment relationship.

WARNING! Be careful renaming contained objects. References from other objects to the object being renamed are not automatically updated with the new name. You must update the references. Objects with broken references receive bad quality data at run-time.

To rename an object's contained name

- 1 Select the object in an Application view.
- 2 On the **Edit** menu, click **Rename Contained Name**.
- 3 Type a new contained name.
All IDEs connected to the Galaxy show the object's new contained name.

Editing Objects

Using the Object Editor, you define attributes specific to an object.

The Object Editor shows object extension pages that are common to all objects and may also show you pages that are unique to the object. See *Enhancing Objects* on page 103 for more information about the **Scripts**, **UDAs**, and **Extensions** pages. Click the tab of each page to open that page.

When you open the Object Editor in non-ReadOnly mode, the object is checked out. No one else can edit an object while you are working with it. If someone else is already working on it, you can open it to view but you cannot make changes.

When editing an object, you may see attribute text boxes showing a --- (dashdashdash). The --- is a placeholder reference that does not cause the associated object to be placed in a warning configuration status when it is validated. You may also see attribute text boxes showing a ---.--- (dashdashdash dot dashdashdash). You need to provide a valid reference in the text box. The ---.--- placeholder causes the associated object to be placed in a warning configuration status when the associated object is validated.

When you are finished editing an object and save it, the configuration data for the object is validated. If errors or warnings are identified during validation, a message appears. You can cancel the save or save the object configuration as it is.

- If you cancel, the Object Editor remains open so you can correct the problems.
- If you save the configuration as it is, the object is placed into a bad or warning state. The object's status is marked in the Galaxy database as Good, Warning or Error. Error means the object is undeployable.

To edit an object in the Object Editor

- 1 Select the object.
- 2 On the **Galaxy** menu, click **Open**. A red check mark appears next to the object's icon indicating it is checked out and the Object Editor opens.
- 3 Make your changes. For more information about locking attributes, see About the UDAs Page on page 75. For more information about setting security, see Setting Object Security on page 70.
- 4 When you finish configuring the object, click **Save** or **Close** on the **Galaxy** menu.
 - **Save** keeps the editor open and saves all configuration changes to the Galaxy database.
 - **Close** closes the editor.
 - To keep the object checked out, select the **Keep Checked Out** check box before closing.

Getting Help

Tooltips are available in the Object Editor. Point to any editor option and a tooltip appears, showing the attribute name. This name is used when referring to the attribute in scripts, for example.

Each object also includes documentation about usage, configuration, run-time behavior, and attributes. For help with configuring the object, click the question mark on the toolbar to open the help for that object.

Help File Structure

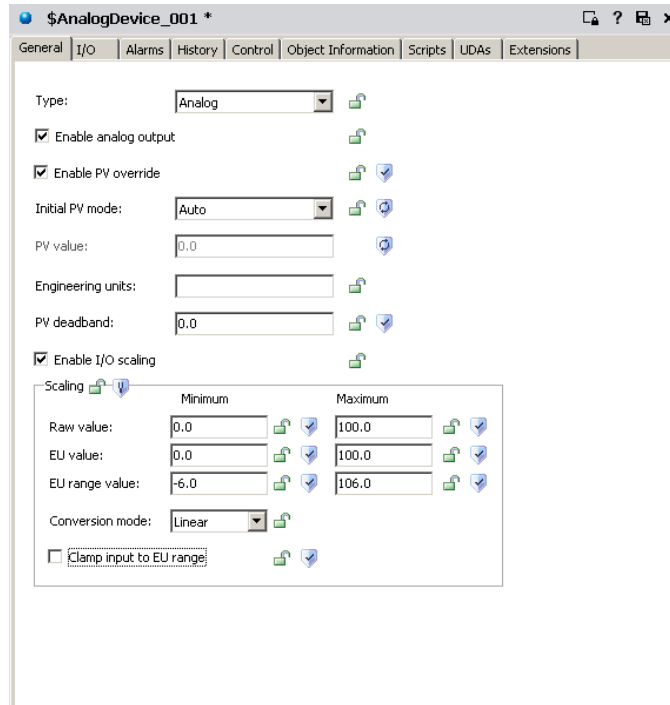
The header part of the Help file contains the following information:

- Tag name The object's name.
- Contained Name The object's contained name. For more information, see [Creating Contained Templates](#) on page 52.
- Description A short summary of what the object is for.
- Code Base The code version of the object.
- Derived From The immediate parent template for the object.
- Object Version The configuration version of the object.
- Process Order The run-time execution order within the host engine's scan (none, before, after) relative to the **Relative Object** element.
- Relative Object The object that runs before or after in the **Process Order**.



The rest of the help file shows general information about the object.

About the General Editor Layout

When you open the attributes for an instance or a template, you see the Object Editor. The Object Editor is where you configure the object's attributes and add scripts or associated graphics to the object. The Object Editor has several pages related to the type of object you select. If you are working with an instance, you see different pages than if you are working with a template. For example, the screen below shows you an analog device template.



When you open the Object Editor, the object is automatically checked out so no other user can work on it. When you close the Object Editor, the object is checked in to the Galaxy database, if it was automatically checked out when the editor was opened.

-  • To keep the object checked out, click **Keep Checked Out** before closing.
-  • To save configuration changes you made, close the editor, and check the object back in, click the **Close** icon.

After the object is checked in, other users can edit it.

Locking and Unlocking Template Attributes

When you create derived templates, you can lock or unlock some or all of the attributes. Locking an attribute prevents the attribute from being changed in derived templates or instances. You can only lock attributes in templates.


Locking an attribute in a template specifies that its value or setting is inherited by all derived objects, both templates and instances. Locking an attribute also makes the attribute act as a constant during run time.





You can reference the attributes:

- Attributes that are locked in a parent template are referred to as “locked in parent.” This parent can be at any parent level above the selected object.
- Attributes that are locked in a template are referred to as “locked in me.”

If an attribute is locked in the template, you can change the value in that template, but not in the derived children. If you change the value in the parent template, the change propagates to all child objects.





Lock controls and status are shown with an icon. If the option is enabled, click the lock control to switch it between locked and unlocked. These icons mean:

Icon	Name	Description
	Locked (in me)	<p>The associated attribute is locked (in me) and enabled. Only templates can have this kind of lock. The attribute value is read/write.</p> <p>Derived templates and instances do not have a unique copy of this attribute. Child objects share the locked attribute of the parent.</p> <p>Changing the value of a locked attribute in the parent template updates the value of that attribute in all derived templates and instances.</p>




Icon	Name	Description
	Locked (in parent)	<p>The associated attribute is locked in the parent object and cannot be unlocked or modified by the child object. Both templates and instances can have these. The attribute is read-only.</p> <p>The templates and objects do not have a unique copy of this attribute, but instead use the attribute value in the parent where the attribute is locked.</p>
	Unlocked	<p>The associated attribute is unlocked and enabled. Both templates and instances can have this kind of lock. The attribute is read/write.</p> <p>The object has its own copy of the attribute value and the value is not shared by derived objects.</p>
	Indeterminate	Refers to a specified group of options. An indeterminate state indicates different lock states for individual options in the group.
	Undefined	The associated attribute doesn't exist. This indicates that another attribute be enabled before the associated attribute is created and before its lock status can be determined.

Note Locking a UDA during configuration makes its value a constant. You cannot write to locked UDAs during run time.

To lock an attribute example

- 1 Create a derived template from the \$Discrete Device base template. Name the derived template \$Valve.
 -  2 Edit the \$Valve template and set an attribute value. Lock the attribute by clicking the **Lock** icon for the attribute.
 - 3 Save \$Valve.
 - 4 Create a derived template from \$Valve. Name it \$BigValve.
 -  5 Create an instance from \$Valve named Valve1. In the editor of \$Valve, the attribute lock icon shows the attribute is locked in me.
-  You cannot change the attribute value in \$BigValve and Valve1. The editor options for the attribute are disabled and the lock icon, if shown, indicates a lock in the parent.
-  Also, the attribute lock icon in children derived from \$Valve is now locked and disabled.
- If you change the attribute value in \$Valve, the change propagates to \$BigValve and Valve1 after you save the changes.






To unlock an attribute example



- 1 Using the objects from the previous example, in the \$Valve template's editor, unlock the locked attribute.
 -  2 Save \$Valve. In the editor for \$Valve, the attribute lock icon shows it is unlocked.
-  The lock type for this attribute of \$BigValve now indicates locked in me. The lock type for this attribute of the Valve1 instance shows unlocked but the locking icon is unavailable.
- 

Setting Object Security

Operators interact with objects through the individual attributes of those objects. Each attribute on the Object Editor that can be modified by operator's at run time and can have an associated security control, which is used to modify its run-time security classification.

If an attribute's security classification is configurable, click the security control to select one of seven possible states:

Security Icon	Description
 Free Access	Lets you change this value without restriction even if you have no defined permissions on the object. Anyone can write to these attributes to perform safety or time critical tasks that can be hampered by an untimely logon request. For example, halting a failing process.
 Operate	Lets you work with Operate permissions to do certain normal day-to-day tasks. These include writing to attributes like Setpoint or Command for a Discrete Device object. This level of security requires you to have Operate permission for the security group for the object.
 Secured Write	Requires you to authenticate using your user name and password each time you want to write to the attribute. You also need to have Operate permissions for the object.
 Verified Write	Requires you to have Operate permissions to log on again and a second, different user to also log on before writing to the attribute. You also need to have Operate permissions for the object.
 Tune	Allows end users with Tune Operational permissions to tune the attribute in the run-time environment. Examples of tuning are attributes that adjust alarm setpoints and PID sensitivity.

Security Icon	Description
 Configure	Allows end users with Configure Operational permissions to configure the attribute's value. Requires that the user first put the object off scan. Writing to these attributes is considered a significant configuration change. For example, a PLC register that defines a Discrete Device input.
 Read Only	Only allows users to read this attribute's value in the run-time environment. This attribute is never written to at run time, regardless of the user's permissions.

If an attribute's security is shown in gray, its security classification is locked in its parent object and cannot be changed or it requires the enabling of a group attribute.

Group Locking/Security

The lock and security controls associated with option groups quickly set those conditions for all options in the group.

The group control typically reflects the setting for all options in the group. But, if at least one option in the group has a lock or security control that is different from the other options, the group control shows an indeterminate icon.



In addition to the undefined controls, the group controls for locking and security are the same as those for individual options.

About the Object Information Page

The **Object Information** page is common to all object configuration editors.

The screenshot shows a web-based configuration interface for an object named '\$AnalogDevice_001'. The interface has a tabbed menu at the top with options: General, I/O, Alarms, History, Control, Object Information (selected), Scripts, UDAs, and Extensions. The main content area is divided into several sections:

- Description:** A text area containing the text 'The AnalogDevice provides supervisory control capabilities for instruments or'.
- Hierarchical name:** A text input field containing '\$AnalogDevice_001'.
- Container:** A text input field containing 'N/A'.
- Codebase:** A text input field containing 'ArchestrA.AnalogDevice.3'.
- Derived from:** A text input field containing '\$AnalogDevice'.
- Host:** A text input field containing 'N/A'.
- Area:** A text input field containing 'N/A'.
- Security group:** A text input field containing 'Default'.
- Execution order:** A section with a lock icon and a dropdown menu for 'Process order' set to 'None', and a text input field for 'Relative object'.

At the bottom of the page, there is a text prompt 'Click the button to add help for this object.' and a button labeled 'Add Object Help'.

This page includes the following fields:

- **Description:** A short summary of the object's purpose.
- **Hierarchical Name:** The fully qualified name of a contained object, including the container object's TagName.
- **Container:** The name of the other object that contains this object, if applicable.
- **Code Base:** The code version of the object.
- **Derived From:** The immediate parent template of the object, either a base or derived template.
- **Host:** Another object to which the object is assigned (for example, a WinPlatform hosts an AppEngine). An object's host determines where an object will run when it is deployed.
- **Area:** An object that represents a logical grouping to which this object belongs. An object's area mostly affects the way in which its alarms are reported to alarm clients.

- **Security Group:** The security group the object is associated with. For more information, see Working with Security on page 231.
- **Execution Order:** If you want this object to run before or after another object within its engine's scan, select from the **Process order** list. Click the **Browse** button to specify the **Relative object** in the **Attribute Browser**. For more information about the **Attribute Browser**, see Referencing Objects Using the Galaxy Browser on page 79.
- **Add Object Help:** Opens a copy of the HTML help page for the template this object is derived from. You can edit this information. This allows you to create Help about the object you are currently configuring for downstream users. This Help appears when you select an object in a view and then click **Object Help** on the **Help** menu.

Customizing Help

Do not use Microsoft Word as an editor to create downstream object HTML help pages. Use an HTML editor like Microsoft FrontPage.

If clicking **Add Object Help** opens Word on your computer, change the program associated with editing HTML files. Open the Windows Explorer's **Folder Options** dialog box and go to the **File Types** page to make this change. For more information about associating programs with files, see your Windows help.

Finding the Help Folders

The path to each object's Help folder is unique. It depends on the path you selected when you installed the Galaxy Repository. The path to an object's Help is:

```
<Installation Path>\Framework\FileRepository\  
<YourGalaxyName>\Objects\<TheObjectID>\Help\1033.
```

The default is:

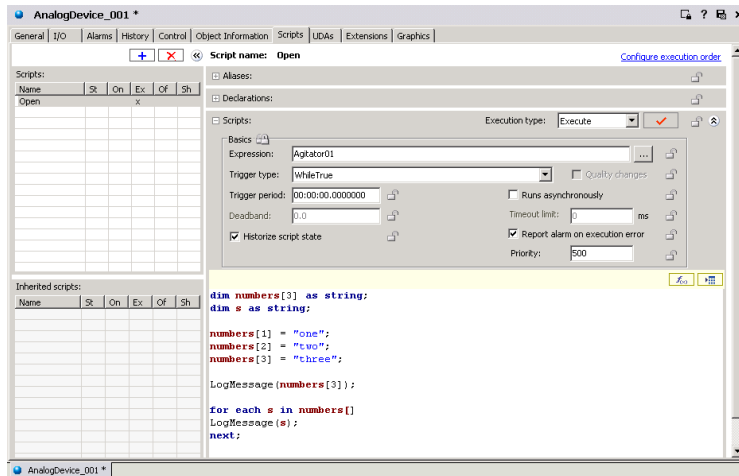
```
<Installation Path> is \<Program Files\Archestra\.
```

To add images to the Help file, place the images in the proper folder on the Galaxy Repository computer and use a relative path to those images in the HTML file.

For the example above, place images in the \1033 folder or create an images folder under it.

About the Scripts Page

The **Scripts** page has five areas. To learn more about using scripts, see [Writing and Editing Scripts](#) on page 106.



The main areas of the **Scripts** page include:



- **Scripts** list: Shows all scripts currently associated with the object. The columns indicate which kind of trigger the script uses: Startup, On Scan, Execute, Off Scan and Shutdown. Click the **Add** button to add a new script.
- **Inherited scripts** name list: Shows all scripts associated with the object's parent. The columns indicate which kind of trigger the script uses: Startup, On Scan, Execute, Off Scan and Shutdown.
- **Aliases** area: Lets you create and modify aliases that apply to the script you are working on. Aliases are logically descriptive names for typically long ArchestraA reference strings that you can use in the script to make the script more readable.
- **Declarations** area: Provides a place to add variable declaration statements, such as `DIM MyArray[1] as FLOAT;`. These declared variables live from the start to the shutdown of the object and can be used to hold values that persist from one execution of the script to the next. They apply only to the script in which they are declared.

- **Basics area:** Provides a location in which you set the expression, triggering conditions, and other settings that run the script in the run-time environment. See *Writing and Editing Scripts* on page 106 for descriptions of triggers and when they are executed. This area includes:

Configure Execution Order: Sets the execution order of multiple scripts (inherited and local) associated with this object.

Historize Script State: Select to send the state of the script to the Wonderware historian.

- **Script Creation box:** Shows the script you are writing.

About the UDAs Page

The **UDAs** page has four areas. To learn more about creating UDAs, see *Creating and Working with UDAs* on page 103.

The screenshot shows the 'UDAs' configuration page for an object named '\$AnalogDevice_001'. The 'UDA name' is 'InputArrays'. The 'Data type' is set to 'Boolean' and the 'Category' is 'User writeable'. The 'Value' section is configured as an array with 1 element, where the value at index 1 is 'False'. The 'UDAs' list on the left contains 'InputArrays'. The 'Inherited UDAs' list is empty.

The main areas of the **UDAs** page include:

- **UDAs list:** Lists all UDAs currently associated with the object. Click the **Add** button to add a new UDA.
- **Inherited UDAs list:** Lists all UDAs associated with the object's parent. The object automatically includes these UDAs. They can only be edited by modifying the parent template.

- **Data type** list: Shows the data type options for configuring the selected UDA.

Select from the data types **Boolean**, **Integer**, **Float**, **Double**, **String**, **Time**, **ElapsedTime** or **InternationalizedString**. For more information about each data type and category, see the help file.

- **Category** list: Shows the category options for configuring the selected UDA.

Allowed categories are:

Calculated: Permits only scripts within the same object to write to the attribute. Calculated attributes are not saved across restarts.

Calculated retentive: Permits only scripts within the same object to write to the attribute. Calculated retentive attributes are saved across restarts.

Object writable: Permits other objects to write to this attribute in addition to being set by scripts within this object. Object Writeable attributes are saved across restarts, and they are `Writeable_S`. This category is not user writeable.

User writeable: Permits other users to write to this attribute in addition to being set by scripts and objects throughout the system. User writeable attributes are saved across restarts, and they are `Writeable_USC_Lockable` and they can be locked at configuration time. This category is not user writeable.

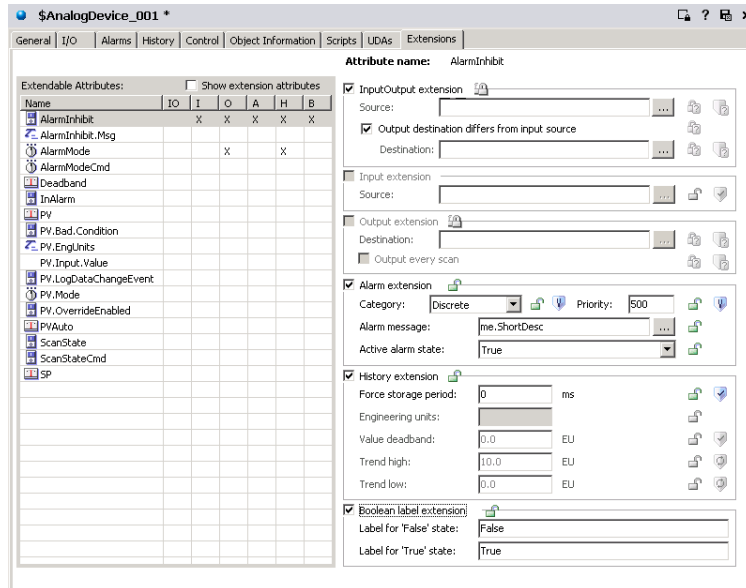
Note You can lock writable attributes. If you select **Calculated** for an attribute, only scripts running on the same object can write to the attribute.

Select **This is an array** and specify the array's length in the **Number of elements** box. You can create an array for each data type except **InternationalizedString**.

The **Value** parameter specifies the initial setting for the attribute when the object is deployed. Enter value data for each data type. In the case of a non-arrayed **Boolean**, select the **True/False** check box to use a True value. Clear the check box to use a False value. For an arrayed Boolean, select the desired element and provide a default value by typing either `true` or `false`.

About the Extensions Page

The **Extensions** page consists of seven areas. To learn more about creating extensions, see *Creating and Working with Extensions* on page 114.



The areas include:

- **Extendable Attributes** list: Lists all attributes currently associated with the object that can be extended. The list can include those added through the UDA tab. Select the **Show Extension Attributes** check box to include attributes added on the **UDAs** page.
- **InputOutput extension** group: Configure an attribute so that its value is both read from an external-reference source and written to an external-reference destination. The source and destination might not be the same. The extension reads the **Source** attribute's value and quality and updates the extended attribute's value and quality every scan. Changes read from the source are not written back to the **Destination** attribute.
- **Input extension** group: Configure the attribute to be readable for an external object. The extended attribute gets the update value of the **Source** attribute.

If you have a large I/O count, use InputOutput Extension instead of using Input Extension and Output Extension separately. Boolean attributes/UDAs that are extended as an InputOutput can handle momentary changes such as false-true-false transitions within a scan.

- **Output extension** group: Configure an attribute to be writeable to an external object. When the value or quality (from Bad or Initializing to Good or Uncertain) of the extended attribute is modified, the value of the **Destination** attribute is updated. The behavior of Boolean attributes/UDAs that are an extended as InputOutput can handle momentary changes such as false-true-false transitions within a scan.

- **Alarm extension** group: An alarm is triggered depending on the state defined in the **Active alarm state**.

When the alarm name contains more than 294 characters (<object name>, <attribute name>), InTouch will not raise an alarm even though the pv.limit and description are minimum length. For more information, see Working with References on page 213.

In the **Alarm message** box, you can browse and select an existing attribute or you can type a text string as an alarm message. This text string can be seen in the InTouch HMI.

If you specify custom labels for the **False** and **True** Boolean states in the **Boolean label extension** area, these custom strings appear in the **Alarm active state** list. These strings can also be used in the InTouch HMI.

Select the **Category** for this alarm. Specify a **Priority** for this alarm. Valid values are 0 to 999.

- **History extension** group: Historize the value of an attribute that does not already have history capabilities. These values are stored in the Wonderware historian.
- **Boolean label extension**: Specify custom text strings for the False state and the True state. These custom text strings appear in the **Active alarm state** list in the **Alarm extension** area for you to select. If you are using the InTouch HMI, you can see these custom text strings in InTouch.

Referencing Objects Using the Galaxy Browser

Use the Galaxy Browser to browse for:

- Attributes of objects. You can quickly find an object attribute or attribute property and add a reference to it when you are configuring an object.
- ArcestrA graphics.
- Graphic element properties.

The Galaxy Browser shows attributes, graphics, or attributes and elements, depending on what you are doing at the time you access the browser.

Browsing for Attributes

You use the Galaxy Browser to browse for:

- Attributes of objects, either instances or own relative references.
- Attributes of templates.

You can open the Galaxy Browser to browse for attributes from:

- Within an AutomationObject Editor. For example, from a script, from an attribute of type MxReference, or from a custom alarm message attribute field).
- Within an ArcestrA Graphic Editor. For example, to use in scripts, animation links and references, properties and custom properties.
- Within InTouch WindowMaker. For example, to use in a reference expression from an animation link or a script.

The Galaxy Browser shows objects in the left pane and the attributes associated with the current selection on the right pane. Only attributes that can be referenced at run time are shown. You can browse "Me." references for alarm messages only.

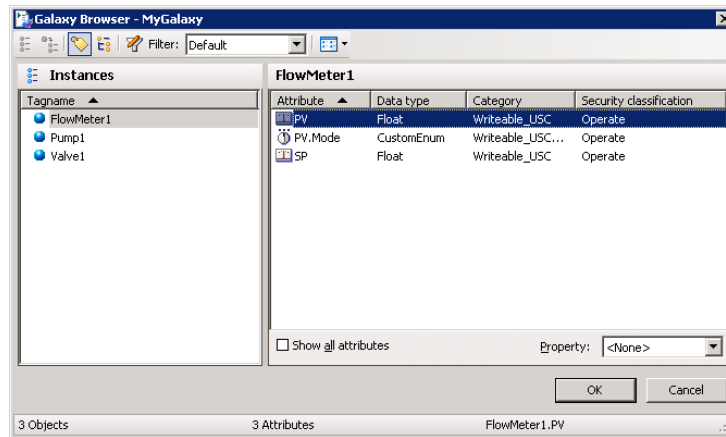
When you open the Galaxy Browser, the last browsed location for attributes is shown. The Galaxy Browser shows the object list based on the last used state (tag name or Hierarchical name). If the last used state of the browser was Tagname and the selected editor reference is a Hierarchical name, the browser opens in Tagname mode.

The status bar displays the attribute property name, and the it displays the graphic element attribute name and description.

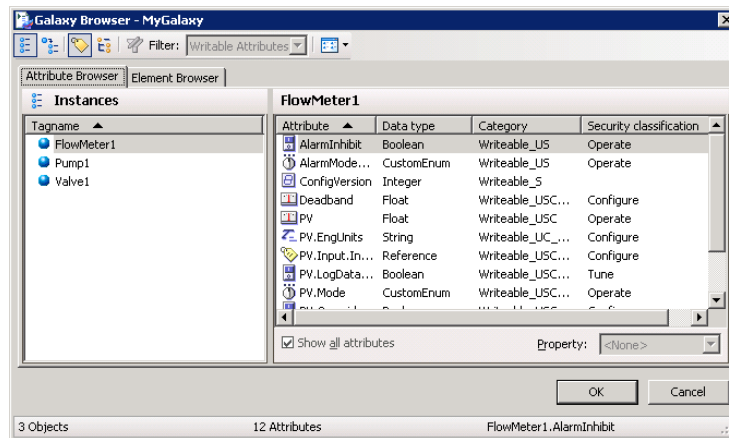
To browse for attributes



- 1 In any area on a page, click the **Browse** button, if available. The Galaxy Browser opens.



If you are browsing for attributes to use with an ArchestrA symbol, such as for an animation or script, the Galaxy Browser shows the attributes in an **Attribute Browser** tab.



- 2 By default, the browser shows only those attributes that are frequently accessed. If you are viewing the attributes of an object for the first time, the right pane can be blank. Select the **Show all attributes** check box to show all of the object's attributes.



- 3 To filter the list of tagnames, click the **Filter** button. For information about configuring a filter, see [Creating a Filter for the Galaxy Browser](#) on page 86. To switch the content of the left pane between a **Tagname** and **Hierarchical Name** list of objects, click the **Show Tag name** or **Show Hierarchical name** buttons.

- 4 You do not have to explicitly make a selection in the **Property** list. If you only select an attribute (leaving **Property** set to <none>), the property of the attribute defaults to **Value**.

If the option in the Object Editor is already configured with an object reference, the **Attribute Browser** shows it or expands to the nearest matching object/attribute/property currently configured in the Galaxy.

If you selected text in the script editor, that text is used as the initial reference string and the browser finds the nearest attribute reference to the selected text.

- 5 When you are done selecting the attribute/property, click **OK** to place the reference into the Object Editor and close the **Galaxy Browser**.
- The fully-qualified reference string appears in the editor option.
 - If you are working in the script editor, the selected reference appears in the script at the current cursor position and replaces text that was selected.

Viewing Attribute Details in the Galaxy Browser

When you view attributes in the Galaxy Browser, you see two areas. The objects shown in the left area include all of the logged in user's checked-out objects plus the checked-in versions of all other objects.

Important The Galaxy Browser shows only the Primary AppEngine and its attributes of a redundant pair. Any Backup AppEngine is not shown. For information about using redundancy, see [About Redundancy](#) on page 304.

The right area shows the attributes of the object selected in the left pane. Depending on the attribute selected, you can see these properties:

<none>	Automatically defaults to the Value property of the selected attribute.
Category	Determines when and where the attribute's data exists (for example, configuration or run time), which users can write to it, and whether the attribute is lockable or unlockable.
Dimension1 (only for arrays)	Returns the dimension of the attribute if it is an array.

Locked	Determines whether the attribute is currently locked. Valid values are: <ul style="list-style-type: none">• Unlocked• LockedInMe• LockedInParent.
Quality	The quality of the attribute as defined in the OPC Draft 3.0 quality definition. ArcestrA stores and transports OPC quality as a 16-bit value. OPC quality is stored for an attribute as a current quality, and it can be historized and sent to clients.
SecurityClassification	Determines which permissions a user has with respect to the attribute when using an ArcestrA application in the run-time environment. Relevant only for attributes that can be written to by users in the run-time environment. If an attribute has no security, this column is blank. For more on security classifications, see Working with Security on page 231.

Data Type	<p>The data type of the attribute:</p> <ul style="list-style-type: none"> • Integer • Boolean • Float • Double • String • Internationalized String • Time • ElapsedTime • ReferenceType • CategorizedStatusType • DataTypeEnum • SecurityClassificationEnum • DataQualityType • CustomEnum • CustomStruct
Value	<p>For information about each of these, see the help for the object.</p> <p>The primary value of the attribute.</p> <p>Sometimes, a list of numbers is included in the Property list. Those numbers map to single bits in an integer attribute's Value property. Valid bit field specifiers are:</p> <p>.00 (least significant bit)</p> <p>.01 .02 .03 .04 .05 .06 .07 .08 .09 .10 .11 .12 .13 .14 .15 .16 .17 .18 .19 .20 .21 .22 .23 .24 .25 .26 .27 .28 .29 .30</p> <p>.31 (most significant bit)</p>

Important Bit field specifiers are not allowed for integer arrays. Although bit field access is only supported in integers, they appear to be allowed for data types besides integer because they do not cause a warning during configuration. They cause errors in the run-time environment.

Browsing for Graphics

You use the Galaxy Browser to browse for graphics from:

- The ArcestrA Symbol Editor, when editing a graphic from the Galaxy, being either a graphic from an object (Template or Instance) or a graphic from the Graphic Toolbox.
- InTouch WindowMaker when editing a managed InTouch application hosted by the Galaxy.

The graphic currently being edited (or browsed from) does not appear in the list of graphics.

Within the same user session, the Galaxy Browser remembers the last browsed location for graphics and presents it whenever called as the starting location so that context is kept. Initial default location for graphic browsing is the Graphic Toolbox with the root selected (Galaxy node).

You can use the Galaxy Browser to browse graphics from:



- The Graphic Toolbox. The browser shows the Graphics Toolbox toolset organization on the tree in the left pane and a list of the graphics contained in the currently selected node (Galaxy node or toolset node) in the right pane.



- AutomationObject templates. The browser shows the Template Toolbox in the left pane and the right pane shows the graphics associated with the currently selected template in the right pane.



- AutomationObject instances. The browser shows a flat list of existing instances of objects in the left pane. The right pane shows the graphics associated with the currently selected instance. You create or apply filters to reduce the scope of the instances shown in the left pane.

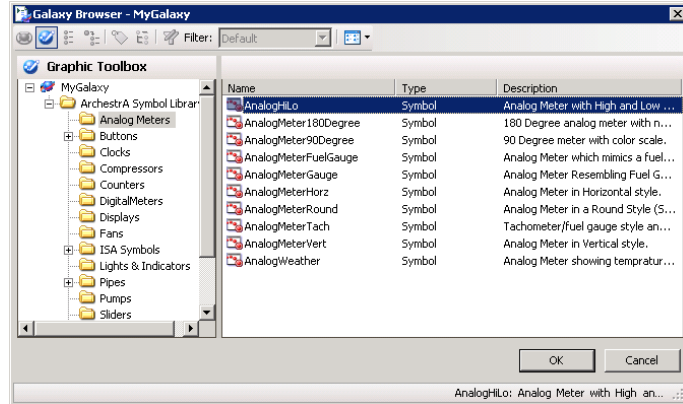


- Relative references. This is possible only when you edit a graphic belonging to an object and browse for graphics from that specific object.

To browse for graphics from the Symbol Editor



- 1 Click the **Embed Graphic** button in the Symbol Editor. The Galaxy Browser opens, showing the location of the graphics on the left pane and the graphics associated with the current selection on the right pane.



- 2 Select a graphic from the list and then click **OK**.
- 3 Click in the canvas to place the graphic.

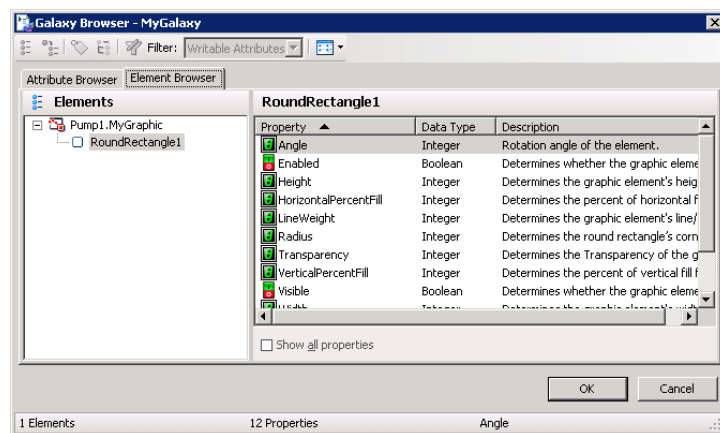
Browsing for Element Properties

If you are working on a graphic, you can create references to properties of other graphics. For example, you can reference another graphic's properties from an animation link or script. You can browse the properties of all elements on the canvas or custom properties.

To browse for element properties



- 1 In any area on a page, click the **Browse** button, if available. The Galaxy Browser opens.
- 2 Click the **Element Browser** tab.



- 3 By default, the browser shows only those properties that are frequently accessed. If you are viewing the properties of an element for the first time, the right pane can be blank. Select the **Show all properties** check box to show all of the object's attributes.
- 4 Select the property and then click **OK**.
 - The fully-qualified reference string appears in the option.
 - If you are working in the script editor, the selected reference appears in the script at the current cursor position and replaces text that was selected.


For more information, see the *Creating and Using Archestra Graphics User's Guide*.

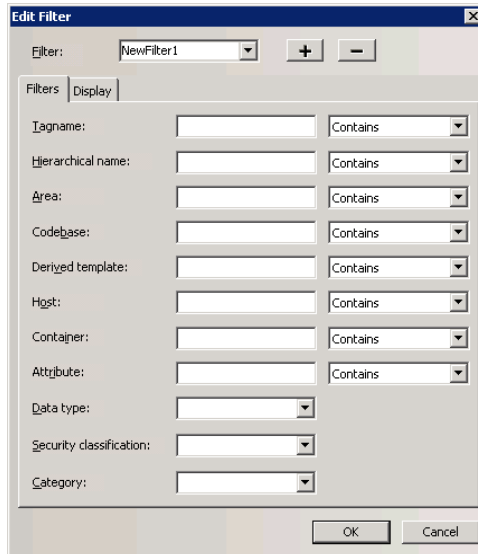
Creating a Filter for the Galaxy Browser

You can create one or more filters that limit the list based on the object name or common attributes. You can also configure the columns you want to show for the list.

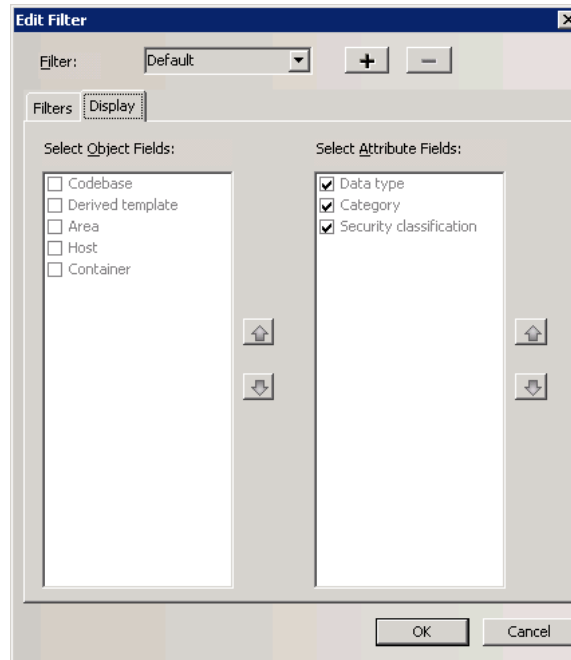
The Default filter provides an unfiltered list of objects and attributes. It cannot be edited.

To create a filter

- 1  Click the **Filter** icon. The **Edit Filter** dialog box appears.
- 2 Click the **Plus** button and type a name for your new filter.
- 3 Click the **Filter** tab.



- 4 Configure the filter details.
- 5 Click the **Display** tab.



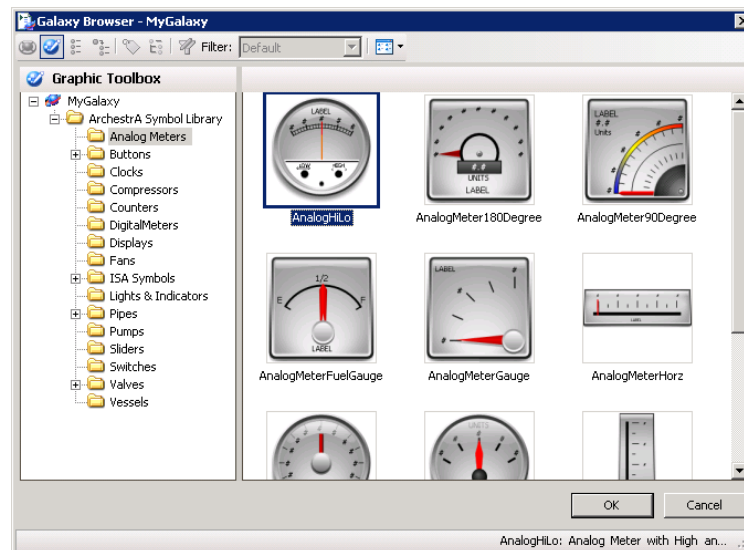
- 6 Configure the columns to show in the right pane of the **Galaxy Browser**. Use the up and down arrows to set the order for the columns.
- 7 Click **OK**. The new filter appears in the **Filter** list.

Changing How Information is Shown in the Galaxy Browser

You can change how information shown in right pane of the Galaxy Browser. You can view:

- A list of only the attribute or graphic names.
- A list of details for attributes or graphics.
- Named graphic icons.
- Thumbnails for graphics.

The following figure shows graphic thumbnails.



Chapter 4

Managing Objects

After you create several objects, like templates, you need to manage them. For example, you need to check objects in and out, you need to validate objects, and you may need to rename objects. You can also export and import objects, allowing you to reuse objects created in one Galaxy in another Galaxy.

Checking Objects Out

To make changes to an object, you must check out the object. Then, you can modify the object and save private versions of it before checking the object in for other users to use. You can select more than one object for checking out at the same time.

The Galaxy marks the objects as checked out to you and it updates the object's **Change Log** that you can view in the **Properties** dialog box. A check mark is shown next to an object's icon in the IDE. No one else can check out the object until you check it back in or until you perform an **Undo Checkout operation**. However, others can open the object for read-only viewing.

To check objects out

- 1 In the **Template Toolbox** or **Application views**, select the objects you want to work with.
- 2 On the **Object** menu, click **Check Out**. Or, open the Object Editor. An object is automatically checked out to you when you open its editor.

The Object Editor opens. You are ready to make your changes.

Checking In Objects

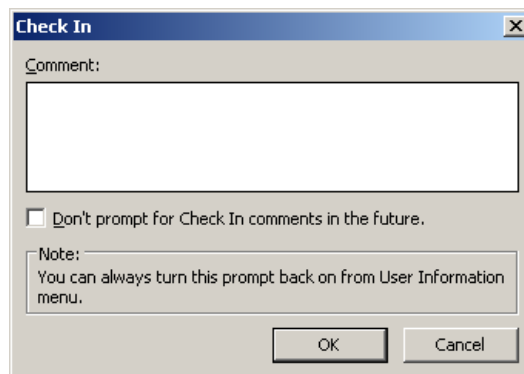
After you finish making your changes, you check the object back into the Galaxy. When you check the object back in, a dialog box prompts you to enter comments about changes you made.

You can turn this dialog box off if you do not want to enter information about your changes. For more information, see *Customizing Your Workspace* on page 31.

Note If the object was automatically checked out when the editor was started and you close the editor without making any changes to the object's configuration, an undo-checkout is automatically performed.

To check in an object to the Galaxy database

- 1 In an **Application view** or the **Template Toolbox**, select the object after you are finished making your changes.
- 2 On the **Object** menu, click **Check In**. The **Check In** dialog box appears.



- 3 Type any comments you want and click **OK**.

Validating Objects

Objects need to be validated before they can be deployed. An object validates its configuration either when you are configuring it, typically when you save that configuration to the Galaxy database.

Validating an object's configuration includes:

- Checking allowable attribute value ranges.
- Compiling its scripts.
- Verifying its attribute references.
- Validating its extensions.
- Validating other configuration parameters that are unique to the object.

Important Script validation on a template does not resolve references used in the script. For example, references to non-existing UDAs are not discovered.

Typically, each option on the Object Editor that requires a string or numeric input has an allowable range of inputs. If you type an input outside the allowable range and then try to change the Object Editor page, close the Object Editor or save the object's configuration, a message appears about the input error, showing the allowable range.

To open the Validation area

- ◆ On the **View** menu, click **Operations**. The **Validation** area opens.

Validating Scripts and Other External Components

Some objects depend on external components to run, such as script function libraries and references to other objects' attributes. The status of these external components can change, perhaps disabling some capability of the object.

For example, an object refers to a value of an attribute of another object, which is subsequently deleted. This will result in the remaining object going to a Warning status.

Normally, the system will update the validation status of an object when the missing script function or object/attribute is later added to the system. But there are a few cases where the status of an object needs to be manually validated by the user.

For example:

- When importing scripts and script libraries, there are cases when the script will import before the associated library and validate incorrectly, and
- When graphics associated with an object are imported along with a graphic they embed, the containing graphic may be imported first and validated incorrectly.

In each of these situations, the object may incorrectly have a status of either Bad or Warning. In this case, you may want to manually validate the object to update its status, especially if the status is preventing the object from being deployed. For more information, see [Validating Manually](#) on page 93.

Two kinds of indicators are shown in the object icons:

- Deployment status for instances only
- Configuration status for templates and instances.

Validating Manually

After you check an object in, you can verify that an object's configuration is valid and update its status by manually validating it. You can use the **Template Toolbox**, the **Application** views or the **Find** dialog box to find objects that need to be validated.

To validate all objects in the Galaxy, validate the Galaxy object.

Note For a large Galaxy this is potentially a time consuming operation, and should be used only when necessary.

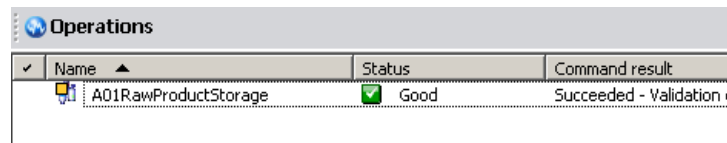
You can select more than one object for validation.

If an object is being edited, validation may not be performed. Also, if validation is in process on an object, other operations you start on the object will fail.

Note You cannot cancel validation operations.

To manually validate one or more objects

- 1 In the **Template Toolbox**, the **Application views** or the **Find** dialog box, select the objects you want to manually validate.
- 2 On the **Object** menu, click **Validate**. The **Operations** view in the IDE opens.



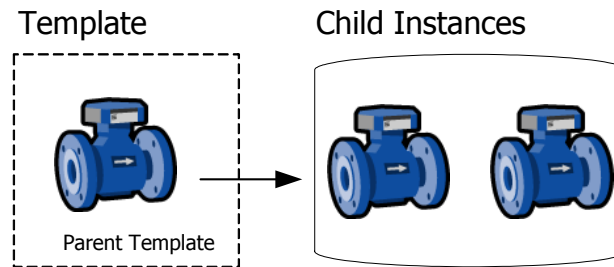
- 3 Continue using the IDE to perform other operations, if needed, while validation is going on, including work on other objects in the Galaxy. If you are validating a Galaxy, then you must leave the Galaxy alone until the validation process is complete.
- 4 When the validation process is complete, you see the results of the validation in the **Operations** View. If the validation failed on an object, you see a message. Correct the problem and validate again.

Note You can also see the errors or warnings that led to an object having a status that is not Good by looking at the Error/Warnings tab within the object's Properties.

Creating Instances

After you create templates, you can create instances. Creating instances makes a specific object from a template, with all the characteristics and attributes of the template.

After you have created an instance of an object, it can be deployed.



You can also customize an instance, if needed. For example, you can have a valve template. When you create an instance of that valve, you can specify the inputs and outputs for that specific valve on your factory floor.

To create an instance

- 1 Select the template you want to use for the instance. For example, to create a valve instance, select a valve template.
- 2 On the **Galaxy** menu, click **New** and then click **Instance**. An instance is created.
- 3 Rename the instance. Select the instance. On the **Edit** menu, click **Rename**. Type the new name. Instance names can be up to 32 alphanumeric characters. You cannot use \$ as the first character. The name must include at least one letter. Instance names cannot include spaces.
- 4 To move the new instance in the **Deployment** view or **Model** view, drag the instance to the new location. You are ready to configure the instance, if needed. For more information, see [Editing Objects](#) on page 63.

Renaming Objects

You can rename an object. Although you can change an object's containment relationship with another object, you cannot directly rename an object's hierarchical name. You can rename its tagname and contained name and its hierarchical name changes automatically. See [Renaming Contained Objects](#) on page 62 for more information about renaming contained objects.

Object names must be unique within each namespace, not within the Galaxy.

- Template names can be up to 32 alphanumeric characters, including the required \$ as the first character. The second character cannot be \$ and the name must include at least one letter. You cannot use spaces in an object name.
- Instances names can be up to 32 alphanumeric characters. You cannot use \$ as the first character. The name must include at least one letter. You cannot use spaces.

Note You cannot use the following reserved names for objects: `Me`, `MyContainer`, `MyArea`, `MyHost`, `MyPlatform`, `MyEngine` and `System`.

An object can have three kinds of names, depending if it is contained by another object. The three names include:

Name	Description
Tagname	The unique name of the individual object. For example, <code>Valve1</code> .
Contained name	The name of the object within the context of its container object. For example, the object whose Tagname is <code>Valve1</code> may also be referred to as <code>Tank1.Outlet</code> , if <code>Tank1</code> contains it and it has the contained name "Outlet".

Name	Description
Hierarchical Name	<p data-bbox="729 264 1289 365">Hierarchical names that are fully-qualified names of a contained object include the name of the objects that contain it.</p> <p data-bbox="729 384 1289 516">Because the object that contains it may also be contained, there are potentially multiple hierarchical names that refer to the same object.</p> <p data-bbox="729 535 935 564">For example, if:</p> <p data-bbox="729 583 1235 684">"Reactor1" contains Tank1 (also known within Reactor1 by its contained name "SurgeTank").</p> <p data-bbox="729 703 1211 804">"Tank1" contains Valve1 (also known within Tank1 by its contained name "Outlet").</p> <p data-bbox="729 823 1122 852">Valve1 could be referred to as:</p> <p data-bbox="729 871 842 900">"Valve1"</p> <p data-bbox="729 919 927 949">"Tank1.Outlet"</p> <p data-bbox="729 968 1117 997">"Reactor1.SurgeTank.Outlet".</p>

When you rename an object, references from other objects to the object being renamed can be broken. Objects deployed with broken references receive bad quality data during run time.

Note Some objects may refer to themselves or to parent/host objects up in the parent/child hierarchy. References that go up or down the hierarchy to refer to other objects are called relative references. Objects with relative referencing are updated automatically if you rename them.

After renaming, all IDEs connected to the Galaxy show the new object name.

To rename an object's tagname

- 1 Select the object you want to rename.
- 2 On the **Edit** menu, click **Rename**.
- 3 Type the new name for the object.
- 4 When you are done, press **Enter**.

Deleting Objects

You can delete both templates and instances with the following exceptions. You cannot delete:

- Deployed instances
- Containers for other objects
- Objects checked out by other users
- Templates that have children (derived templates or instances)

Note Make sure you correctly select the objects you want to delete. After you delete an object, you cannot undelete it. You must recreate it.

To delete an object from the Galaxy

- 1 In the **Template Toolbox** or **Application views** area, select the object to delete. Select multiple objects by using **Shift+click** or **Ctrl+click**.
- 2 On the **Edit** menu, click **Delete**. When the message appears, confirm you want the object deleted and click **Yes**.

Exporting Objects

You can export some or all of your Galaxy objects. When you export, you are exporting the objects' associated templates, configuration state, and containment state of those objects. The information is saved in an .aaPKG file.

After the Galaxy objects are exported, you can import into the same or another Galaxy.

If your objects have scripts associated with them, you need to export the script library separately. For more information about exporting script libraries, see [Exporting Script Function Libraries](#) on page 98. For more information about scripts and script libraries, see the *Application Server Scripting Guide*.

Before you start, make sure all objects you want to export are checked in. If an object selected for export is checked out, the checked in version of that object is exported instead. This can lead to old versions of objects being exported.

Exporting an entire Galaxy is different than backing up the database. Unlike with backups, change logs for the objects are not exported. When you export objects, only the related security information for the specific object is exported.

To export an object

- 1 In the **Template Toolbox** or **Application Views**, select one or more objects to export.
- 2 On the **Galaxy** menu, click **Export** and then click **Automation Object(s)**. The **Export Automation Object(s)** dialog box appears.
To export all of the objects in the Galaxy, on the **Galaxy** menu, click **Export** and then click **All Automation Objects**.
- 3 In the **Export** dialog box, browse to a path and type a name for the exported file.
- 4 Click **Save**. The file is saved with the specified name and a .aaPKG extension.
- 5 When the export is complete, click **Close**. Now you can import the .aaPKG file into another existing Galaxy.

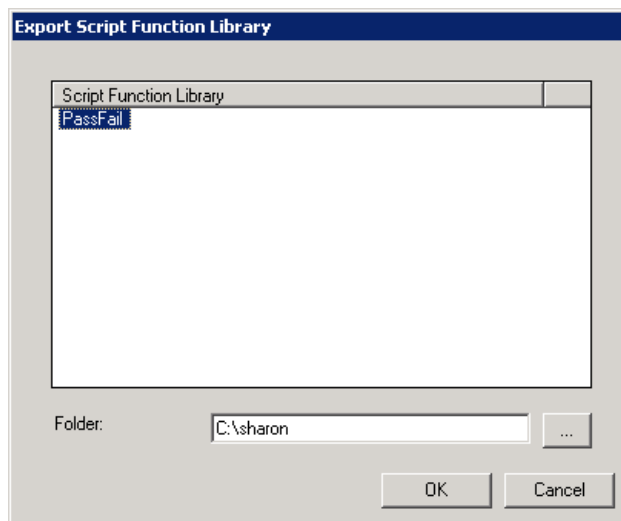
Exporting Script Function Libraries

If you want to export objects that use scripts, the scripts are exported with the object.

Some scripts include functions that depend on external files called script function libraries. In this case, you must export the script function libraries separately.

To export a script function library

- 1 On the **Galaxy** menu, click **Export** and click **Script Function Library**. The **Export Script Function Library** dialog box appears.



- 2 In the **Script Function Library** list, select the library or libraries you want to export. If needed, browse to folder where you keep your script libraries.

- 3 Click **OK**. The selected script library is exported. Each script is named with the name of the script and a .aaSLIB file name extension.
- 4 When the export is complete, click **Close**. Now you can import the .aaSLIB file into another existing Galaxy.

Importing Objects

You can reuse objects from another Galaxy in your Galaxy. This saves you a lot of time if the objects are already set up in another Galaxy.

Importing instances previously exported from a Galaxy retains previous associations, when possible, such as assignment, containment, derivation, and area.

You can import objects from exported .aaPKG files or from an .aaPDF file. An .aaPDF file contains the configuration data and implementation code for one or more base templates. It's created by a developer using the ArcestrA Object Toolkit.

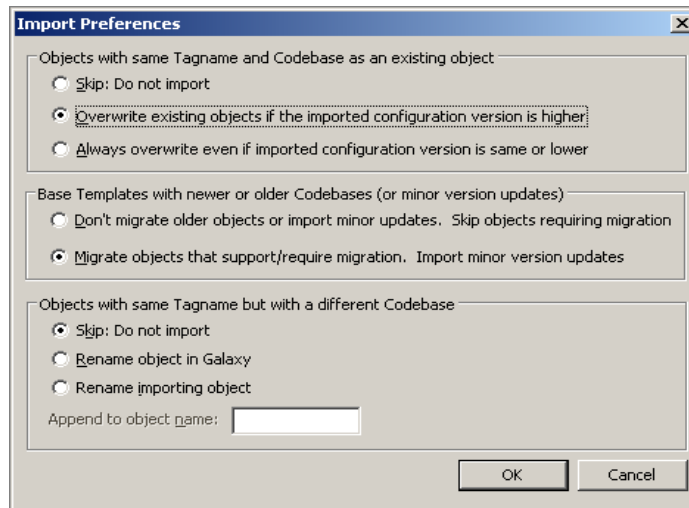
You cannot have two objects with the same name or more than one copy of the same version of an object in the same Galaxy. When you import an object, you can choose how you want naming and version conflicts handled.

You should perform an "Upload Runtime Changes" before importing a new version base template if instances of the template are deployed. This saves changes made at Runtime to the Galaxy database. For more information, see [Uploading Run-time Configuration](#) on page 145

To import objects

- 1 On the **Galaxy** menu, click **Import** and click **Automation Object(s)**. The **Import AutomationObject(s)** dialog box appears.

- 2 Browse for the file with either a .aaPKG or a .aaPDF extension. You can select more than one file. Click **Open**. The **Import Preferences** dialog box appears.



- 3 In the **Objects with same Tagname and Codebase as an existing object** area, select one of the following:
 - Skip: Do not import** leaves the existing object unchanged.
 - Overwrite existing objects if the imported configuration version is higher** (default) replaces the existing object with the object being imported if the imported object has a newer configuration.
 - Always overwrite even if imported configuration version is same or lower** replaces the existing object regardless of whether the existing object has an older configuration or the same configuration.
- 4 In the **Base Templates with newer or older Codebases (or minor version updates)** area, select one of the following:
 - Don't migrate older objects or import minor updates.** **Skip objects requiring migration** does not migrate objects with an older codebase when a newer codebase exists in the Galaxy.
 - Migrate objects that support/require migration. Import minor version updates** migrates an older codebase when the replacement object is available.

For more information about migrating, see *After You Import* on page 101.
- 5 In the **Objects with same Tagname but with a different Codebase** area, select one of the following:
 - Skip: Do not import** leaves the existing object unchanged.

Rename object in Galaxy imports an object with a matching tagname but a different codebase from the existing one. The existing object is not overwritten but is renamed.

Rename importing object imports an object with a matching tagname but a different codebase from the existing one. The existing object is not overwritten. The imported object is renamed.

- 6 Click **OK**. The import process starts.
- 7 When the import process is complete, you can start using the objects you imported.

Importing Script Function Libraries

You can enhance an object's functionality by attaching a script to it. Some scripts include functions that depend on external files called script function libraries. Scripts are included in the object import operation, but you must import the script function libraries separately.

If you import an object whose script references a script function library that is not resident in the Galaxy, the imported object is set to Bad state and cannot be deployed. To correct this, import the script function library and validate the object. For more information about scripts, see the *Application Server Scripting Guide*. For more information about validating scripts, see Validating Objects on page 91.

Script function libraries that are COM libraries developed using Visual Studio 6 or earlier are not automatically deployed. To place this COM library on the target platform, you can either:

- Install it directly on the target platform and register it.
- Import it to the Galaxy, and then export it as an aaSLIB. Modify the aaSLIB xml to designate that the library is to be registered as a COM object. Reimport the aaSLIB so that it is automatically deployed and registered.

After You Import

Imported templates are listed in the proper toolset in the Template Toolset as defined in the object. Imported instances are shown in the Application views. The following post-import rules apply:

- If a toolset does not exist, it is created.
- If the object belongs to a security group that does not exist, it is associated with the Default security group.

- If the object belongs to an area that does not exist, it is associated with the Unassigned Area.
- If the host to which the object is assigned does not exist, it is assigned to the Unassigned Host.
- If you selected from the **Import Preferences** dialog box, the migrated objects are marked with “software upgrade required” if they are deployed. These objects will be upgraded when the objects are redeployed.

Note If you import a new version of an existing instance, the new version is marked as requiring deployment if the existing object is already deployed.

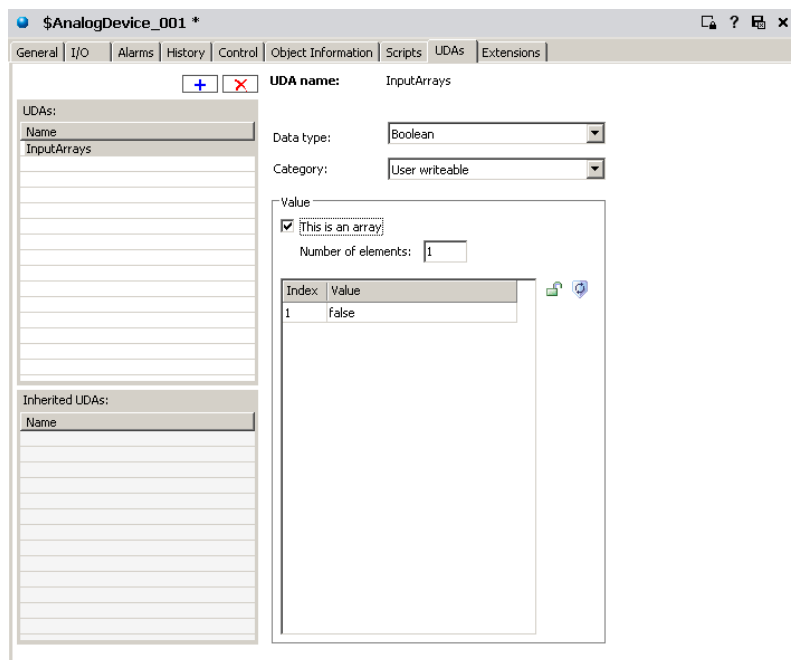
Chapter 5

Enhancing Objects

After you create an object, you can enhance and extend the object by using User Defined Attributes (UDAs), scripts, and graphics extensions.

Creating and Working with UDAs

You can add UDAs to a template or an instance. When you add a UDA to a template, the UDA, its data type, and category are automatically locked in the child instances. For an overview of the UDAs page, see About the UDAs Page on page 75.



If UDA parameters such as initial values and security classifications are locked in the template, they cannot be changed in child instances. If these parameters are unlocked in the template, the initial value and security are editable and lockable in derived templates. When unlocked in either the base or derived template, the value is editable in instances.

After you add an attribute to an instance, it appears in the **Attribute Browser** list for use with the scripting and attribute extension functions. For more information about using the Attribute Browser, see Referencing Objects Using the Galaxy Browser on page 79.

In the **UDAs** page, you can:

- Add a new attribute to an object.
- Configure its data type.
- Specify the attribute category.
- Set initial values and locks on the new attribute.
- Set whether the new attribute is an array and how many elements are in the array.

UDAs and Scripting

When using UDAs in scripting, keep the following list in mind.

- If you use **Calculated** and **Calculated retentive** UDAs as counters, they must be manually initialized. For example, if you use `me.UDA=me.UDA+1` as a counter in a script, you must also initialize the UDA with something like `me.UDA=1` or `me.UDA=<some attribute value>`.
- **Calculated** UDAs can be initialized in scripts with **Execution type** triggers of On Scan and Execute, but not initialized in Startup scripts.
- You must initialize **Calculated retentive** UDAs in Startup scripts and you can initialize these UDAs in On Scan and Execute scripts. A **Calculated retentive** UDA retains the attribute's current value after a computer restart, redundancy-related failover, or similar situation in which valid checkpoint data is present. Your Startup script should contain a statement testing the Boolean value of the `StartingFromCheckpoint` attribute on the object's AppEngine. If the value is `TRUE`, do not initialize the UDA. If the value is `FALSE`, initialize the UDA. For more information about `StartingFromCheckpoint`, see the help for the AppEngine object.


UDA Naming Conventions

UDA names can have up to 32 alphanumeric characters, including periods. UDA names must include at least one letter.

A UDA name that starts with an underscore (_) as the first character of the name is a hidden attribute. Hidden attributes do not appear in the **Attribute Browser**, the **Properties>Attributes** dialog box, or the Object Viewer unless you select to view **Include Hidden**.

Important After creating a UDA, it is available, like all attributes, when extending the object further on the **Extensions** page. If you extend a user defined attribute and then delete or rename the user defined attribute, all object extensions added to the object on the **Extensions** page are lost.

To create and associate a UDA with an object

-  **1** On the **UDAs** page of the Object Editor, click the **Add** button. A UDA is added to the **UDAs** list.
- 2** Type the new UDA name.
- 3** In the **Data type** list, select the **Data type** for the new attribute. The available options in the **Data type** list change depending on your selection in the **Data type** list.
- 4** Set the remaining parameters as needed.

Note For detailed information about each item on the **UDAs** page, see **About the UDAs Page** on page 75.

- 5** Lock the values, if needed. The lock is available only when you are working with a template. If you are working with an instance, it shows the lock status for the value in the parent object.
- 6** Set any security you need. For more information about setting security, see **Setting Object Security** on page 70.
- 7** Save and close the Object Editor when you are done.

Writing and Editing Scripts

You can write scripts to extend and customize your objects. You can write scripts that run commands and logical operations based on specified criteria being met. For example, a script starts when a key is pressed, a window is opened, or the value of an attribute changes.

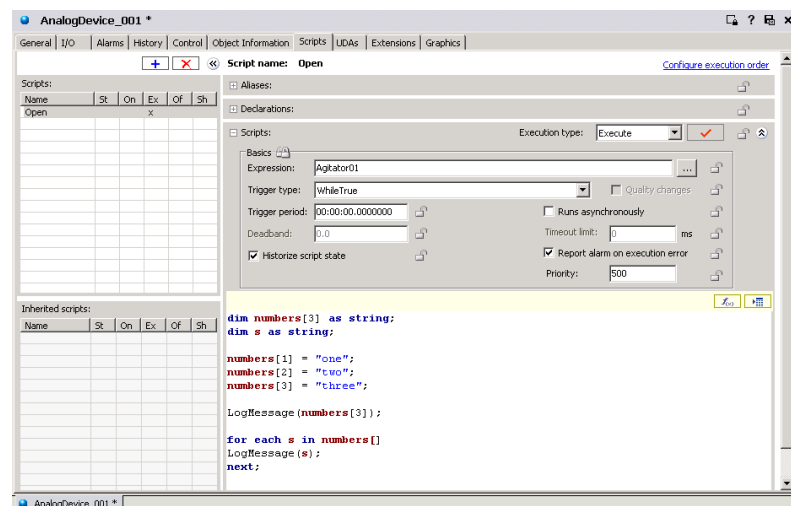
Using scripts, you can create a variety of customized and automated system functions. A script adds behavior that runs when the object that contains the script is deployed and the object is either:

- On scan in the run-time environment or
- Changes scan or start/shutdown state

A script typically runs based on attributes of the object that contains it, but can be started by another script based on changing values of attributes of more than one object.

When a script condition is true, the script runs at least once immediately. The maximum length of a script trigger period is 49-days. A script never runs if the trigger period exceeds 49-days.

For specific information about writing scripts, including the scripting language, syntax, commands, and using .NET, see the *Wonderware® Application Server Scripting Guide*.



For more information about the scripts page, see About the Scripts Page on page 74.

Important You cannot pass UDAs as parameters for system objects. Instead, use a local variable as an intermediary or explicitly convert the UDA to a string using an appropriate function call when calling the system object.

About Scripts

The following characteristics apply to the scripting environment:

- Script text has no length limitations.
- Selecting a script function from the **Script Function Library** dialog box adds it and its syntax to the script text where you can edit it.
- You can save a script with syntax errors, but the object cannot be deployed until you correct the script syntax errors.
- You can validate your scripts before using them. This helps you avoid syntactically correct but semantically incorrect combinations such as two statements declaring the same variable. Variables can be declared only one time in a single block.
- You can change the name of a script at any time by renaming it in the Object Editor.
- In the run-time environment, a script execution error stops the script's current execution. Script execution is retried on the next AppEngine scan.

Script Execution

The existence and execution order of scripts associated with an object are inherently locked at each stage of development in the template, derived template, and instance.

For example, a set of scripts associated with a template are treated as an ordered block in the **Configure Execution Order** dialog box when configuring execution order in a next-generation derived template.

New scripts in the derived template can be ran in any order before and after the template's block of scripts. The derived template's execution order is treated as a block in any downstream derived templates or instances.

Scripts cannot trigger any faster than the scan period of the AppEngine the script is associated with or faster than the scan period of the AppEngine that hosts the object that the script is associated with.

Scripts run in one of two modes:

- Synchronous scripting mode is the default for running scripts in the run-time environment. This mode runs scripts in order while an object is running on scan.
- Asynchronous scripting mode is a group of scripts running on the same, lower priority execution thread. These scripts only support Execute triggering and run independently from each other. Set the maximum number of independent threads in the AppEngine configuration editor.

To use either scripting mode, you must select **Execute** as the **Execution Type** in the **Scripts** area on the **Scripts** page.

To create and associate a script with an object

- 1 Add a script. On the **Scripts** page of the Object Editor, click the **Add** button. A script is added to the **Scripts Name** list.
- 2 Type a name for the script and press **Enter**. Script names can be up to 32 alphanumeric characters, including periods. At least one character must be a letter.

Note For detailed information about each item on the **Scripts** page, see [About the Scripts Page](#) on page 74.

- 3 Select a trigger that starts the script in the run-time environment.
Execution Type triggers include: Startup, On Scan, Execute, Off Scan and Shutdown.
 - If you select **Startup**, **On Scan**, **Off Scan**, or **Shutdown**, the **Basics** group is unavailable. The script is triggered when the object starts, goes on scan, goes off scan, or shuts down.
If you select **Execute**, the **Basics** group is available.
 - If you selected **Execute** as the script trigger, select a **Trigger Type**. Depending on the type selected, you are required to enter an **Expression** and/or **Trigger Period** and **Deadband** values. When the combination of **Expression**, **Trigger Type**, **Trigger Period** and/or **Deadband** is satisfied in the run-time environment, the script starts running. See the following table for more information.

The **Trigger Period** format is as follows:

Hours:Minutes:Seconds:Milliseconds

For example, 3 hours, 5 minutes, and 10.5 seconds is:
03:05:10.5000000

Expressions are limited to one language statement in length and calling only synchronous mode script functions. Avoid using script functions with side effects in expressions because subtle behaviors can occur.

Trigger Type	Description
Periodic	Script runs at an interval specified in the Trigger Period box. A time interval of zero (0) starts the script during every scan. This trigger does not require an expression.
While True	<p>When the object containing the script is going On Scan, a While True script evaluates its expression at the next scheduled scan period of the AppEngine. The script runs if true and then periodically thereafter at the trigger interval.</p> <p>The script continues running as long as the Expression value evaluates to true. A Trigger Period is required. Zero (0) evaluates the expression at the AppEngine scan period and non-zero means the expression is evaluated at the specified time interval.</p>
On True	When the object containing the script is going On Scan, an On True script evaluates its expression at the next scheduled scan period. The script starts at the transition between the expression going from false to true.
On False	When the object containing the script is going On Scan, an On False script evaluates its expression at the next scheduled scan period. The script starts at the transition between the expression going from true to false.

Trigger Type	Description
Data Change	<p>Scripts run when the value or quality of the expression changes. The expression must evaluate to a single, non-arrayed value of the following types: integer, real, time, elapsedtime, string, double, Boolean, custom enumeration and quality. To allow execution based on quality, select the Quality changes check box.</p> <p>A deadband can be specified for all data types. Deadband units for time and elapsedtime types are milliseconds. Deadband is always ignored for strings because any change (even from “ABC” to “abc”) is considered a change. Only major changes in quality (from Good/Uncertain to Bad/Initializing or vice versa) are considered changes.</p> <p>After the object is put on scan, Data Change-triggered scripts start running at the AppEngine’s next scan period and then at each subsequent scan period in which the value or quality changes.</p>
While False	<p>When the object containing the script is going On Scan, a While False script evaluates its expression at the next scheduled scan period, runs if false, and then periodically thereafter at the trigger interval.</p> <p>The script runs as long as the Expression value evaluates to be false. A Trigger Period is required. Zero (0) evaluates the expression at the AppEngine scan period and non-zero means the expression is evaluated at the specified time interval.</p>

- 4 Select one or more of the following:
 - Set the **Runs Asynchronously** and associated **Timeout Limit** parameters, as needed.
 - Select **Report Alarm on Execution Error** and set a **Priority** for the alarm if you want the alarming function to alert you if a script execution failure occurs.
 - Select **Historize Script State** to store the state of the script in your application’s historian.

- 5 In the **Declarations** area, type variable declarations about the script you are writing.
- 6 Set aliases for the reference strings in the **Aliases** area. This can simplify the script code and allows script code to be created and locked at a template level using alias names. When an individual instance of that template is created, you can link external attributes to the alias names.



In the **Aliases** area, click the **Add** button to add a new alias. An alias is added to the list. The name is shown in edit mode. Double-click the **Reference** entry, and enter a reference string for the alias. You can also click the **Browse** button at the end of the **Reference** block to open the Attribute Browser for easy selection of an object's attributes.



- 7 Write the script in the **Script Creation** box. Use the **Display Script Function Browser** and **Display Attribute Browser** buttons to help you insert script functions and object attribute references in your script. For help with the specific commands and syntax, see the *Application Server Scripting Guide*.



Click the **Validate Script** button to validate if your script contains any syntax errors.

- 8 Order the scripts. If you have more than one script associated with a single object, click **Configure Execution Order**. Ordering does not apply to asynchronous scripts. If a script is added to an instance derived from a template that contains scripts, the new script automatically defaults to running after the derived scripts.
- 9 When you are done creating your script and setting its execution triggering parameters, save and close the Object Editor.

Locking Scripts

When you lock a script in a template, the following rules apply:

- The name of a script and its existence is implicitly always locked. This means:
 - You cannot delete the script in derived objects.
 - You cannot change the name of the script in derived objects.
 - If you rename the script in the template, the name changes in all derived objects.
 - You can delete a script in the template after you create derived objects. The script disappears from the derived objects.
 - You can add a script to the template after you create derived objects. The script appears in the derived objects.
 - You can add scripts to derived objects. Adding scripts to derived objects does not affect parent object scripts.
- You can lock or unlock the script text in a template. There is script text for **Declarations, Execute, Startup, Shutdown, On Scan** and **Off Scan**. You cannot separately lock each script in the script editor. You use a single group lock to lock or unlock all at once.

After you lock a script, derived templates and instances cannot modify any of the script text.

- When the script text is locked in a template, the alias names are automatically locked. The alias references are never locked. Locking of aliases is not specified separately.

Locking aliases means the entire list of alias names is locked, including the number of items in the list. You cannot add new alias names in derived templates or instances when the alias list is locked. The alias references are always editable in derived templates and instances even when the entire list of alias names is locked. This is the primary objective of aliases.

- The script description, runs asynchronous flag, expression, trigger type, trigger period, deadband and execution error alarm are individually lockable and can be locked separately from the script text. A group lock is provided for this group of attributes.

- When you add a script to a template, all properties of the script are editable.
- When you add a script to an instance, all properties of the script are editable except for the lock properties. A lock is never editable in an instance.

Important An expression typically uses attribute references. To lock the expression and the associated script in a template, use aliases in both the expression and the script. This allows you to specify the attributes that the aliases point to on a per instance basis while the script code is locked.

The following rules apply to the derivation behavior of locked script attributes:

- If an attribute is locked in a template, then all templates and instances derived from the template share the copy of the value of the locked attribute. A change to the value is only allowed in the template that locked it. The change propagates to all derived templates and instances.

For scripts, locking an attribute of the script, such as its script text or execution type, in a template means all derived templates and instances point to that locked attribute. Future changes to that locked attribute's value, such as modifying the script text, propagate and appear in all derived templates and instances.

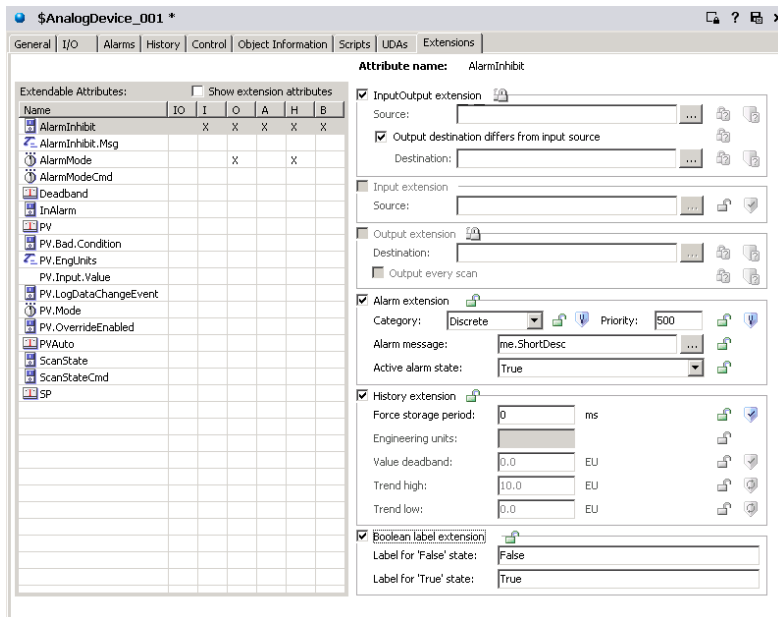
If instances are deployed, they are marked pending update status. After they are redeployed, the change to the locked attribute in the template exists in the deployed instance.

- If an attribute is not locked in a template, then all templates and instances derived from that template receive their own copy of the value of that locked attribute. A change to that unlocked value is allowed in derived templates and instances because they own their own copy. Any change to the unlocked attribute value in the template does not propagate to any derived template or instance.

An unlocked attribute in a script (such as expression or script order) in a template means that all derived templates and instances have their own copy and the value of that unlocked attribute can change. Future changes to that locked attribute's value (for example, modified expression) in the template does not propagate to any derived template or instance. If instances are deployed, their status does not change to pending update. Redeploying them does not cause the value to change in the deployed instance.

Creating and Working with Extensions

The **Extensions** page allows you to configure an existing attribute for input, output, alarm, and history functionality not embedded in the original object.



About Extension Inheritance

You can add object extensions to either derived templates or instances. Base templates cannot be extended. The following parent-child object characteristics also apply to object extensions:

- If you add an extension to a derived template that has objects derived from it, all child objects inherit the extension.
- You cannot add an extension to derived objects that duplicate parent object extensions in name and type.
- You cannot add an extension with the same name as an existing attribute extension.
- Renaming an extension in the template to which it was originally added renames all other objects derived from the template. This change happens when the template is checked in.

- You can check in a template with a new extension with the same name as an existing attribute in a derived object. The template definition of the extension overrides the extension in the derived object.
- If you remove an extension from a template, that extension is removed from any child object. You see the change when you check in the template.

To create and associate an extension with an object

- 1 On the **Extensions** page, select an attribute from the **Extendable Attributes List**. The extension groups dynamically change to allowed extension rules for the selected attribute type.
- 2 Select the check box for the kind of extension you want to apply to the selected attribute. The associated parameters for each kind of extension become available. For detailed information about each item on the **Extensions** page, see *About the Extensions Page* on page 77.
- 3 Select the parameters you want. Do the following:
 - For **InputOutput extension**, enter a Source attribute by either typing in the reference string or by using the **Attribute Browser** to search for the reference string in an object. For specific information about using this extension, see *Using the InputOutput Extension* on page 117.

If **Destination** is different from **Source**, click **Output destination differs from input source**. Enter a **Destination** attribute by either typing in the reference string or clicking the **Attribute Browser** button. An **X** appears in the **IO** column of the selected attribute.

Important If you clear the **Output destination differs from input source** check box, the **Destination** box automatically shows “---”. In the run-time environment, “---” is the same reference as the **Source** value entered during configuration time. During run time, you can change the **Source** reference. During configuration, do not lock the **Destination** parameter if you clear the **Output destination differs from input source** check box.

- For **Input extension**, enter a **Source** attribute by either typing in the reference string or clicking the attribute browser button at the right. Use the **Attribute Browser** to select an attribute and automatically insert the correct reference string for that attribute. An **X** appears in the **I** column of the selected attribute. For more information about using the Attribute Browser, see Referencing Objects Using the Galaxy Browser on page 79.
- For **Output extension**, enter a **Destination** attribute by either typing in the reference string or clicking the attribute browser button at the right. Use the **Attribute Browser** to search for a reference string in an object. For specific information about using this extension, see Using the Output Extension on page 120.

Select the **Output Every Scan** check box if you want the extended attribute to write to the **Destination** attribute every scan period of the object. Otherwise, the write executes only when the value is modified or when quality changes from Bad or Initializing to Good or Uncertain. An **X** appears in the **O** column of the selected attribute.

- For **Alarm extension**, select a **Category** from the list: **Discrete, Value LoLo, Value Lo, Value Hi, Value HiHi, DeviationMinor, DeviationMajor, ROC Lo, ROC Hi, SPC, Process, System, Batch** or **Software**. For specific information about using this extension, see Using the Alarm Extension on page 124.

Type a **Priority** level for the alarm (default is 500). An **X** appears in the **A** column of the selected attribute.

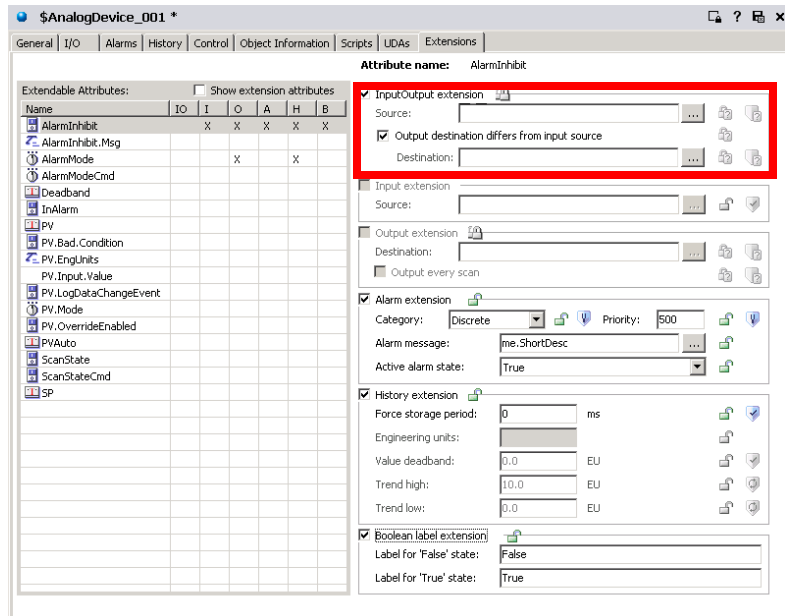
From the **Active alarms state** list, select the value that triggers that alarm. The items in this list can be customized in the **Boolean label extension** area.

- For **History extension**, enter values for the remaining parameters: **Force Storage Period, Engineering Units, Value Deadband, Trend High and Trend Low**, if available (depends on the data type of the selected attribute). An **X** appears in the **H** column of the selected attribute. For specific information about using this extension, see Using the History Extension on page 125.
- For **Boolean label extension**, specify different text strings for the **Label for 'False' state** and the **Label for 'True' state**, if needed. These text strings appear in the **Active Alarm State** list for you to select.

- 4 Lock the values, if needed. The lock symbol is available only when you are working with a template. If you are working with an instance, it shows the lock condition of the value in the parent object.
- 5 Set any security for the attribute. For more information about setting security, see [Setting Object Security](#) on page 70.
- 6 Save and close the Object Editor to include the new attribute extensions in the configured object.

Using the InputOutput Extension

InputOutput extensions allow an attribute in a template or an instance to be configured so that its value is both read from and written to an external reference. The InputOutput extension monitors the value/quality of an input and sends outputs on state change.



The output destination can be the same or different from the input source. The references are always to another acceptable attribute type in the Galaxy.

You can add multiple InputOutput extensions to an object. However, you cannot add an InputOutput extension to an attribute that already has an input or output extension.

Note You can extend lockable attributes with an InputOutput extension, but they only function correctly during run time if the extended attribute is unlocked.

When Objects Are On Scan

When an object is On Scan, the value and quality of the InputOutput-extended attribute mirrors the quality of the externally referenced attribute during a successful read. The data quality of the extended attribute is set to Bad when reads fail. Reads can fail because of communication errors or datatype conversion failures.

While the extended object is On Scan, the data can change quality. If an external set (for example, from a user) to the extended attribute changes either the value or quality, then a write of the extended attribute's value to the destination occurs during the next execute phase. The quality must be Good or Uncertain for a write to occur. For writes to occur because of a quality change, the quality change must be a transition from Bad or Initializing to Good or Uncertain.

The attribute called WriteValue is publicly exposed and plays an important role in driving outputs. When the extended object is Off Scan, quality is always Bad and user sets are accepted.

Using InputOutput Extensions in Scripts

Two common types of scripts can be written on InputOutput-extended attributes: One can look at the input side and one can look at the output side.

The input side script uses the current value coming from the input source location and performs logic or calculations on it. This script refers directly to the extended attribute in its expressions. For example, if the extended attribute is "me.udal", the script refers directly to "me.udal" for data change conditions and for expressions within the script.

The output side script can modify output or validate a new requested output value. This script refers to the "WriteValue" attribute that extends the extended attribute: "me.udal.WriteValue". So, to validate a new requested value to the udal, for example, a data change condition expression is written on "me.udal.WriteValue". In addition, if the script wants to do clamping or validation, it can manipulate the "me.udal.WriteValue" directly to clamp the output value. For example:

```
If (me.udal.WriteValue > 100.0 ) then
    Me.udal.WriteValue = 100.0;
Endif;
```

The data change expression for this script is "me.udal.WriteValue" because this value changes when a new value is about to be written to the field.

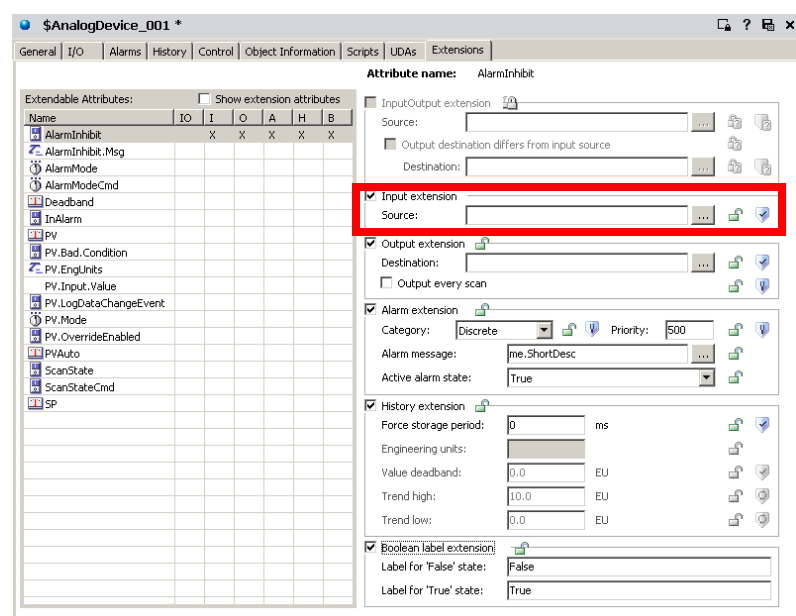
The script can intercept this value just before output and manipulate it. To prevent WriteValue from being written out, its data quality can be set to Bad with the SetBad() function.

For more information, see Working with Outputs on page 121.

Using the Input Extension

You can add multiple input extensions to an object. However, you cannot add an InputOutput extension to an attribute that already has either an input or output extension. Arrays are not supported.

Note Lockable attributes can be extended with an Input extension, but they only function correctly during run time if the extended attribute is unlocked.



If the data types of the extended and **Source** attributes are the same, they are set to equal values according to the extended object's execution rate. If the two attributes are different data types, coercion rules are applied.

If coercion fails or the input value is out of the extended attributes range, quality for the extended attribute is set to Bad. Otherwise, the extended attribute's quality matches the **Source** attribute. When the extended object is Off Scan, quality is always Bad and user sets are accepted.

Attributes extended by an input extension are not protected by their security classification. The only enforced security specifies if an IDE user can edit, or extend, the object. An input extension can be added to a template or instance. If added to a template, the existence of the input extension is automatically locked in derived objects.

Using the Output Extension

Writeable and Calculated attributes can be extended with an output extension. Arrays are not supported.

The screenshot shows the configuration window for the \$AnalogDevice_001 * object, specifically the Extensions tab for the AlarmInhibit attribute. The window is divided into two main sections: a table of extendable attributes and a list of extension types.

Extendable Attributes:

Name	IO	I	O	A	H	B
AlarmInhibit		X	X	X	X	X
AlarmInhibit.Msg						
AlarmMode						
AlarmModeCmd						
Deadband						
InAlarm						
PV						
PV.Bad.Condition						
PV.EngUnits						
PV.Input.Value						
PV.LogDataChangeEvent						
PV.Mode						
PV.OverrideEnabled						
PVAuto						
ScanState						
ScanStateCmd						
SP						

Attribute name: AlarmInhibit

Extensions:

- Input/Output extension
 - Source: []
 - Output destination differs from input source
 - Destination: []
- Input extension
 - Source: []
- Output extension (highlighted with a red box)
 - Destination: []
 - Output every scan
- Alarm extension
 - Category: Discrete
 - Priority: 500
 - Alarm message: me.ShortDesc
 - Active alarm state: True
- History extension
 - Force storage period: 0 ms
 - Engineering units: []
 - Value deadband: 0.0 EU
 - Trend high: 10.0 EU
 - Trend low: 0.0 EU
- Boolean label extension
 - Label for 'False' state: False
 - Label for 'True' state: True

An output extension can be added to a template or an instance. If added to a template, the existence of the output extension is automatically locked in derived objects. The output **Destination** attribute in the extension is separately lockable in templates.

If the data types of the extended and Destination attributes are the same and only when the quality of the extended attribute is good, the two attributes are set to equal values according to the extended object's execution rate. If the two attributes are different data types, coercion rules are applied. If coercion fails, the extended attribute is placed into a configuration error and type mismatch state.

An attribute that is enhanced with an Output Extension has the following characteristics:

- A value can be output only when quality is Good or Uncertain. The quality is not output, only the value is output, because quality is not output on sets.
- When the quality changes from Bad or Initializing to Good or Uncertain, the value is output, even if the value is not modified.
- When the quality changes from Good to Uncertain, with no value modification, the value is not output.
- When the object goes Off Scan, no output is done.
- When the extended object is Off Scan, quality is always Good and user sets are accepted.

Working with Outputs

The following information applies to the functionality of InputOutput and Output extensions as well as the output function of the Field-Reference, Switch, and Analog-Device objects.

If a single set request is made to a destination attribute during a single scan cycle, that value is sent to the destination. During a single scan cycle, though, more than one set request to the same destination is possible. In that case, folding occurs and the last value is sent to the destination.

During a single scan cycle, only the last value requested during a scan cycle is sent to its destination when the object executes. Its status is marked as Pending as it waits for write confirmation from the destination object. All other set requests during that scan cycle are marked as successfully completed.

If one or more new sets are requested during the next scan cycle, then the second scan cycle's value is determined as described above. It is then sent to the destination when the object executes again and the value sent to the destination during the previous scan cycle is marked with successful completion status even if write confirmation is not received.

Within a single scan cycle, data is folded and only the last set requested is sent to the destination. For example, an {11,24,35,35,22,36,40} sequence of set requests results in a value of 40 being sent to the destination object. All other values result in successful completion status.

Boolean data types are the exception to this rule. Boolean data types are used in User sets from InTouch or FactorySuite A² Gateway. This allows an unknown user input rate (for example, repeated button pushes) with a consistent object scan rate for outputs, and creates reproducible results.

In this case, a combination of folding as described above plus maintenance of a queue of one element deep better meets the expectation of users. To begin with, the first value set after the object is deployed (the default True or False) is always written to its destination.

Subsequently, the following occurs during a single scan cycle: A two-tiered caching scheme of a Value to be Sent and a Next Value to be Sent is implemented. The Value to be Sent is based on data change as compared to the last value sent to the destination object. The Next Value to be Sent is based on data change as compared to the Value to be Sent value.

When the first data change occurs, the new value is cached in the Value to be Sent queue. Folding occurs if the same value is requested again. If another value change occurs, this second value is cached in the Next Value to be Sent queue. Again, folding occurs if the same value is requested again.

The Value to be Sent value is sent during the next scan cycle, and the Next Value to be Sent value is sent during the following scan cycle.

Note In the case of Boolean data types used in Supervisory sets (sets between ApplicationObjects and Arcestra) or a mixture of Supervisory and User sets during a single scan cycle, the behavior is the same as the other data types.

For Boolean data types and User sets, the following examples apply:

Previous Scan Cycle Value Sent	Scan Cycle Set Requests	Value to be Sent	Next Value to be Sent
0	1,0,0,1,1	1	none
1	1,0,0,1,1	0	1
0	1,1,0,0	1	0
1	1,1,0,0	0	none

When the same attribute is extended with an Input extension and an Output extension, writes to the Output extension's **Destination** occur every scan regardless of whether the extended attribute has changed.

This behavior occurs even when the **Output Every Scan** check box is cleared, which may add more network traffic. The behavior does not apply to an Input extension.

Quality of Input, InputOutput and Output Extensions

When the object is On Scan, the value and quality of the Input-extended attribute mirrors the quality of the externally referenced attribute in the case of successful reads. The data quality of the extended attribute is set to Bad when reads fail because of communication errors or datatype conversion failures.

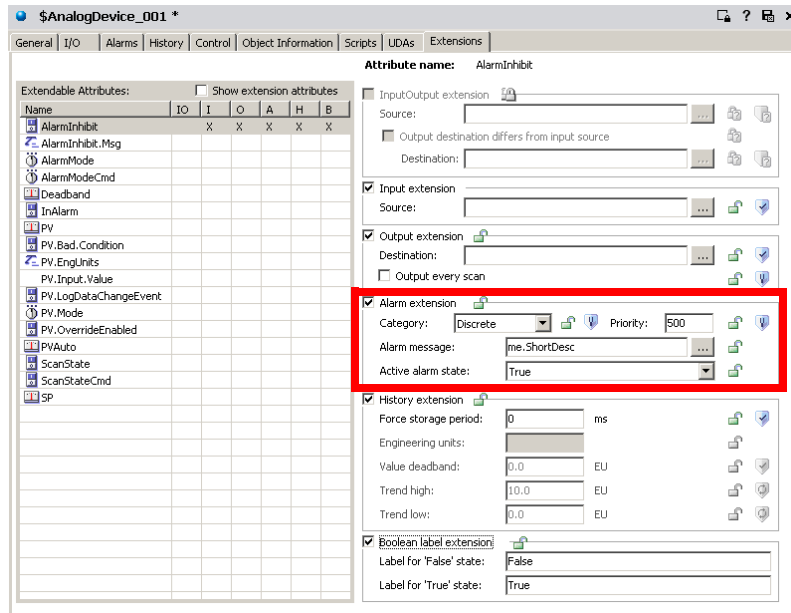
While the extended object is On Scan, it behaves as follows: If an external set (for example, from a user) to the extended attribute causes either the value or quality to change, then a write of the extended attribute's value to the destination occurs during the next execute phase.

The quality must be Good or Uncertain for a write to occur. For writes to occur because of a quality change, the quality change must be a transition from Bad or Initializing to Good or Uncertain. The attribute called WriteValue is publicly exposed.

When the extended object is Off Scan, quality is always Bad and user sets are accepted.

Using the Alarm Extension

An alarm extension can be added to a template or instance Boolean attribute. If added to a template attribute, the alarm extension is automatically locked in derived objects. Attribute arrays cannot be extended.



Select the **Category** and specify an **Priority** for this alarm. Valid values are 0 to 999.

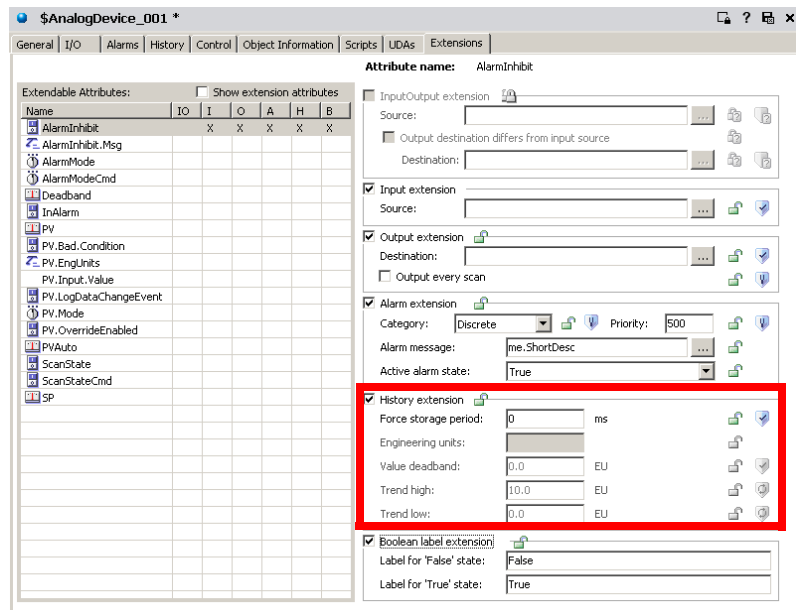
In the **Alarm message** box, you can browse and select an existing attribute or you can type a text string as an alarm message. This text string appears in the InTouch alarm view.

The **Active alarms state** list can show the customized items you specified in the **Boolean label extension** area. It lets you select the state that triggers the alarm. If you do not specify a value in the **Boolean label extension** area, you see **True** and **False**.

For more information about using Alarms, see Working with Alarms and Events on page 175.

Using the History Extension

Any attribute that exists at run time and is not already historized can be configured with the history extension.



A history extension can be added to a template or an instance attribute. If added to a template attribute, the existence of the history extension is automatically locked in derived objects.

You can extend Writeable and Calculated attributes of the following data types with a history extension:

- Float, Double (stored as a Float)
- Integer
- Boolean
- String stored as Unicode, 512 character limit
- Custom Enumeration stored as an Integer
- ElapsedTime stored as seconds

History Extension group parameters include:

- Force Storage Period: Period after which the value must be historized even if the value has not changed.
- Engineering Units: Engineering units of the attribute to be historized.

- **Value Deadband:** The threshold value, measured in engineering units, that the absolute value of the difference between the new and last-stored values must differ before storing the new value to history. A value of zero (0) is valid and means that any level of change results in the new value being stored. A change in Quality always causes a new record to be stored, regardless of whether the Value has changed.
- **Trend High:** The default top of a trend scale.
- **Trend Low:** The default bottom of a trend scale.

Using the Boolean Label Extension

In the **Boolean label extension** area, you can specify an explicit name for the True and False states of Boolean attributes. For example, if a Boolean attribute is associated with the status of a motor, you can specify the states as **Stopped** and **Running**.

The labels defined in this area are available in the **Active alarm state** box in the **Alarm** extension area.

These labels are also shown in the **Value** and **Limit** columns of the Alarm and Event database and InTouch AlarmView control.

The screenshot shows the configuration window for an AnalogDevice_001. The 'Boolean label extension' section is highlighted with a red box. The configuration is as follows:

Extendable Attributes:	IO	I	O	A	H	B
AlarmInhibit		X	X	X	X	X
AlarmInhibit.Msg						
AlarmMode						
AlarmModeCmd						
Deadband						
InAlarm						
PV						
PV.Bad.Condition						
PV.EngUnits						
PV.Input.Value						
PV.LogDataChangeEvent						
PV.Mode						
PV.OverrideEnabled						
PWAuto						
ScanState						
ScanStateCmd						
SP						

Boolean label extension configuration:

- Label for 'False' state: False
- Label for 'True' state: True

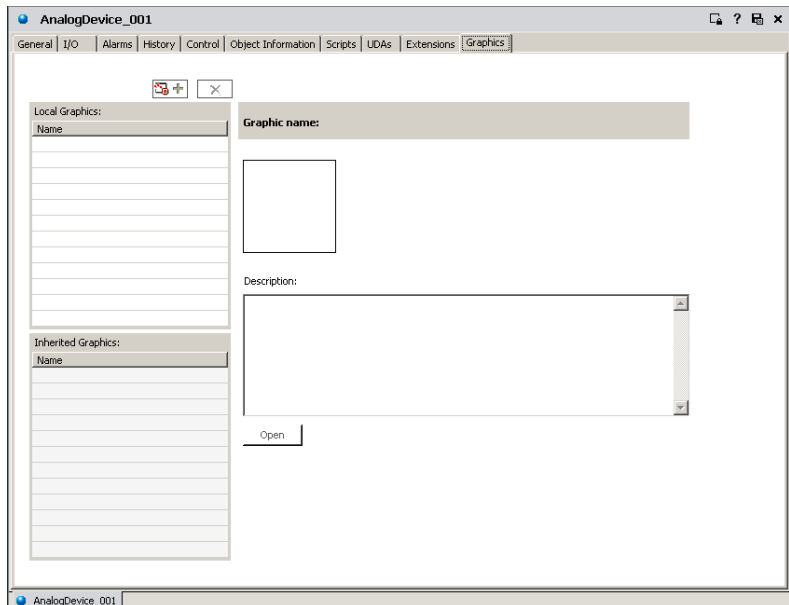
Creating and Working with Graphics

Use the **Graphics** page to add, modify, rename or delete local graphics; or to view inherited graphics. You must have the derived template or object instance checked out in order to add, modify, or delete local graphics; otherwise, you can only view local graphics.

You can view graphics inherited from parent templates and object instances. You cannot add, modify, or delete inherited graphics. This means that in order to edit an inherited graphic, you must check out the derived template or object instance where the graphic is local.

You can only add graphics to derived templates and object instances. You cannot add graphics to a base template.

On the **Graphics** page, use the **Open** button to start the Symbol Editor for the selected graphic.



Adding Graphics

You can add graphics to an object.

To add a graphic symbol to the object



- 1 On the **Graphics** page of the Object Editor, click the **Add** button. A graphic name is added to the **Name** list.
- 2 Type the new local graphic symbol name.
- 3 In the **Description** box, type a description for the graphic symbol being added.
- 4 Click **Open**. The graphics tool box opens. For instructions on using the Symbol Editor, see *Creating and Managing ArcestrA Graphics User's Guide*.

Modifying Graphics

All modifications made to graphic symbols referenced by other objects are visible in the referenced objects.

To modify a graphic symbol

- 1 On the **Graphics** page of the Object Editor, click the **Name** of the graphic symbol to be modified.
- 2 Click **Open**. The graphics tool box opens, showing the selected graphic symbol.
- 3 Make changes. For instructions on using the Symbol Editor, see *Creating and Managing ArcestrA Graphics User's Guide*.

Renaming Graphics

You can rename a graphic symbol.

To rename a graphic symbol

- 1 On the **Graphics** page of the Object Editor, click the **Name** of the graphic symbol to be renamed.
- 2 Type the new name, and press **Enter**. The new name is saved.

Deleting Graphics

Caution Deleting a graphic symbol with embedded references breaks the links to their related objects. “Symbol Not Found” appears when you open objects whose embedded graphic symbols have been deleted.

To delete a graphic symbol

1 On the **Graphics** page of the Object Editor, click the **Name** of the graphic symbol to be deleted.



2 Click the **Delete** button. The **Delete** dialog box appears. The left pane lists the graphic symbol selected for deletion. The right pane shows all embedded references to the selected graphic.

3 Click **Yes**.

Chapter 6

Deploying and Running an Application

You can deploy and test your objects at any time during development. When you are ready to test or run the application in production, you deploy the Galaxy.

You can see what your application looks like in the Deployment view or the Model view. Both views show you the structure of your application. For more information, see *Using IDE Application Views* on page 23.

Planning for Deployment

Deploying your Galaxy copies the objects from the development environment to the run-time environment. This makes your objects “live” and functional.

Until you deploy your IDE configuration environment to the run-time environment, changes you make in the IDE do not appear in the run-time environment. To see run-time data associated with your objects, use Object Viewer or InTouch. For more information about using Object Viewer, see the *Object Viewer User’s Guide*.

Objects deploy from the configuration environment to the run-time environment as follows:

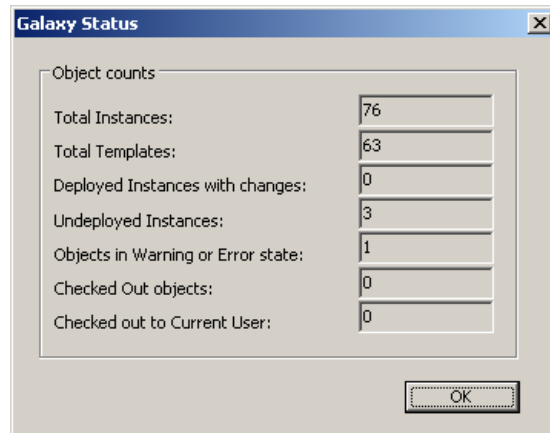
IDE Configuration Environment	deploys to	Object Viewer Run-time environment
Galaxy database	‡	[Does not exist in run-time environment]
Templates	‡	[Does not exist in run-time environment]
Instance objects	‡	Instance objects <i>[Run-time configuration and behavior]</i>
Security: General permissions	‡	[Does not exist in run-time environment]
Security: Operational permissions	‡	Run-time permissions to acknowledge alarms and modify attributes
Scripts configuration	‡	Scripts execution
Alarms configuration	‡	Alarms generate and acknowledge
History configuration	‡	History Logs <i>[Wonderware Historian]</i>

Determining Galaxy Status

You can see an overview of the condition of your Galaxy before you deploy. This lets you know if you have objects that are in warning or error status.

To determine the status of a Galaxy

- 1 Connect to the Galaxy.
- 2 On the **Galaxy** menu, click **Galaxy Status**. The **Galaxy Status** dialog box appears.

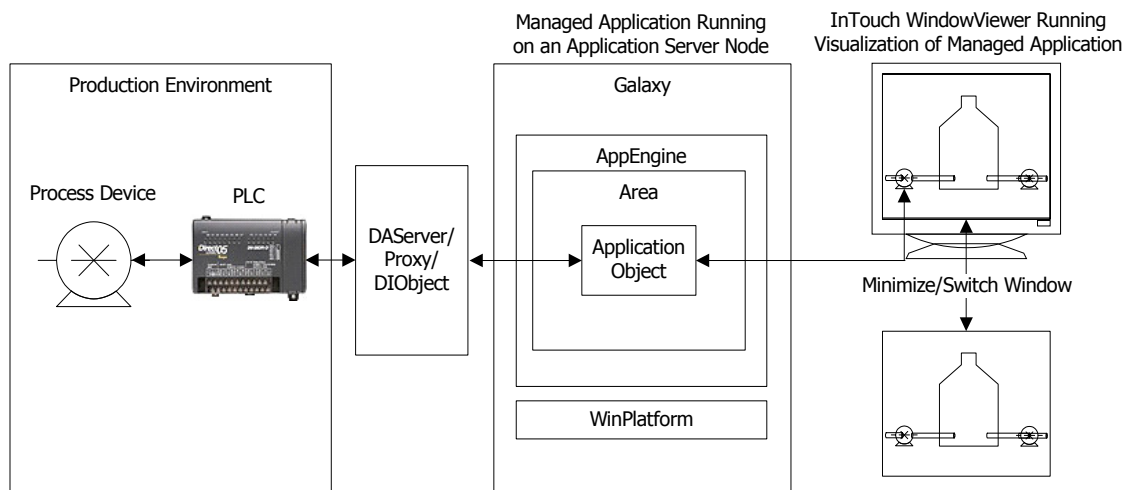


The **Galaxy Status** dialog box shows the counts of total instances, total templates, deployed instances with changes, undeployed instances with changes, objects that have an error or warning state, objects that are checked out, and object you have checked out.

- 3 Click **OK**.

Configuring Advanced Communication Management

Advanced Communications Management minimizes network traffic and CPU usage of a DI Object and DA Server when a particular DI attribute is currently subscribed to, but its value is not currently shown. For example, scanning for updated values from a DI attribute representing pump RPM is suspended when an operator minimizes the application window containing a pump graphic containing attribute references that subscribe to the DI Object's pump RPM attributes.



When the items on the current window are no longer active because the window is minimized, a Suspend operation is performed for each item. Suspending scanning causes data updates to stop flowing to the application without deleting the subscriptions for each item. This reduces the overhead of unsubscribing and subscribing again to items when windows are minimized and restored.

In Advanced Communication Management, applications monitor the number of outstanding references to attributes that are updated with values from an external source object. When an attribute's external reference count reaches zero, the application sends a Suspend message to the source object requesting it to suspend updates. For example, a UserDefined object named PumpRPM has an integer UDA named RPM, which is input extended from plc1.n7:11 where plc1 is an ABPLC DI object.

When there are no active subscriptions to ud1.RPM, the subscription between input extension and PumpRPM is suspended. When a client like ObjectViewer or a managed application running in WindowViewer subscribes to ud1.RPM, an Activate message is sent to the DI object to start sending data between the plc1 object and the PumpRPM object. This enables ObjectViewer or a managed client application running in WindowViewer to receive current plc1 data.

Unlike defined static attributes, some scan groups contain *dynamic* attributes. Dynamic attributes within a scan group provide the means for a running application's objects to dynamically subscribe to data from an external device. Dynamic attributes are created, activated and added to the scan group using ArchestrA object UDA's or InTouch indirect tags and scripts that include the IOSetRemoteReferences function.

When a dynamic attribute is poked, the time stamp is updated to the timestamp passed in with the value if available, or the current time provided by the hosting AppEngine is used. If the data provider passes in a Value, Time, Quality (VTQ) triplet of data in the callback, the time stamp is associated with the value and quality used to update the attribute.

To enable Advanced Communication Management, you must:

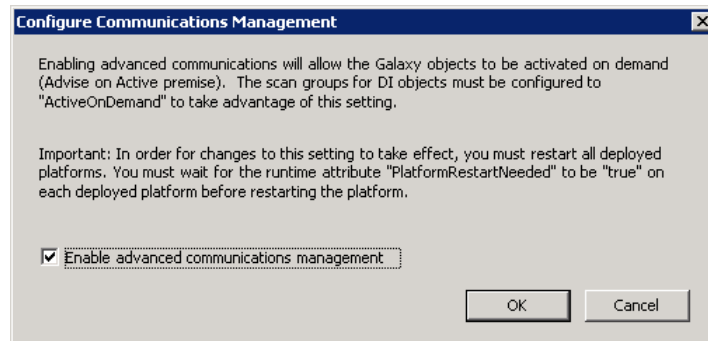
- Select Advanced Communication Management for your Galaxy.
- Set the scan mode for each scan group that belongs to your device integration objects within the Galaxy.

Selecting Advanced Communication Management

Advanced Communication Management is a Galaxy-wide configuration setting that you set from the ArchedrA IDE. Advanced Communication Management is active by default.

To configure Advanced Communication Management

- 1 Connect to the Galaxy.
- 2 On the **Galaxy** menu, click **Configure** and then click **Communications Management**. The **Configure Communications Management** dialog box appears.



- 3 If necessary, select **Enable advanced communication management** and click **OK**.

Note Any deployed platform and application engines must be restarted when the Advanced Communication Management state changes. Restarting should be done by means of the SMC.

Configuring Scan Modes

A scan group is a collection of object attributes with associated data items. Scan group attributes reflect I/O points in the address space of Data Access Server whose values are polled from devices at a common update interval. The update interval for each scan group is inherited from the scan groups configured in the source DIOjects.

You can configure scan groups for OPCClient, InTouchProxy, DDESuiteLinkClient, and RedundantDIOject device integration objects. Within each scan group, you can specify data items to use as the object attributes. At run time, scan group items are updated with the latest values from the OPC DA Server according to the update interval.

The OPCClient, RedundantDI, DDESuiteLinkClient, and InTouchProxy objects contain a Scan Mode attribute that can be set for each scan group. The assigned scan mode value determines if objects are continuously updated with current data when an operator minimizes or closes the application window.

You can assign three different scan modes to a scan group:

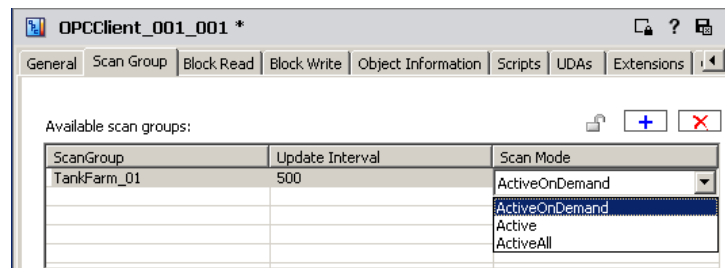
- **Active**
 - Items are deleted and added to the scan group as requested (referenced) by the clients.
 - Items that exist when a scan mode change occurs are not deleted unless the previous mode was ActiveAll and the items are no longer referenced.
 - The Active scan mode polls all points that are referenced, whether active or inactive.
 - In Active scan mode an attribute is always in the active state. When the last reference to the attribute is unregistered/unadvised, the attribute is deleted.
- **ActiveAll**
 - Scanning is continuous and dynamic attributes are never removed when unsubscribed.
 - Items are activated by client requests, but continue to exist even after the client are not interested in them anymore.
 - Items are not removed when the scan mode changes.
 - The scan group polls the field device for all points irrespective of whether they are currently active, inactive, or even subscribed to by items.
- **ActiveOnDemand**
 - When the scan mode is configured as ActiveOnDemand, attributes that are not actively being referenced by any client or object are made Inactive and are not scanned.
 - New Items are deleted and added to the scan group as requested (referenced) by clients.
 - Items that exist when a scan mode change occurs are not deleted unless the previous mode was ActiveAll and the items are no longer referenced.
 - ActiveOnDemand scan mode polls the field device for currently active items only. Inactive items are not scanned.

The following table shows scan mode scan states for dynamic attributes based on whether items are referenced or not.

Dynamic Attribute State	Scan Mode Active	Scan Mode ActiveAll	Scan Mode ActiveOnDemand
Reference/ Some active	Scan	Scan	Scan
Reference/All Inactive	Scan	Scan	No Scan
Not Referenced	Delete	Scan	Delete

To set a scan mode for a scan group

- 1 Edit the device integration object.
- 2 Show the editor page containing scan groups by completing one of the following:
 - Click the **Scan Group** tab if you are editing a RedundantDIObject or OPCClient object.
 - Click the **Topic** tab if your are editing a DDESuiteLinkClient object.
 - Click the **Items Configuration** tab if your are editing an InTouchProxy object.



- 3 If necessary, add a scan group by clicking the plus sign box above the list of available scan groups.
- 4 If necessary, set the length of the scan group update interval in milliseconds. The default update interval is 500.
- 5 Double-click in the **Scan Mode** box of the scan group to show a drop-down list of available scan modes. ActiveOnDemand is the default scan mode.
- 6 Select a scan mode from the list.
- 7 Save your changes to the device integration object and exit from the editor.

Deploying Objects

You deploy object instances for three reasons:

- Testing.
- Placing the application into production to process field data.
- Updating an existing application with changes you made.

When you are ready to deploy, make sure the following conditions are met:

- Bootstrap software is installed on the target computers.
- The objects being deployed are not in an error state in the Galaxy database.
- You created, configured, and checked in objects to the Galaxy.
- Objects are assigned to a host.
- The object's host is already deployed. A cascade deploy operation, which deploys a hierarchy of objects, deploys all objects in the correct order. This deploys an object's host before the object is deployed.
- Any associated script libraries are ready for use on the target computer. For more information, see [Importing Script Function Libraries](#) on page 101.

Note DINetwork objects have specific configuration limits. For example, whether more than one object can be deployed to a single WinPlatform. The IDE does not check for these limits. For more information, see the help file for the DINetwork object for specifics on configuration limits.

You can tell if you have objects that need to be deployed by looking at the icons next to the objects. Deployment status icons include:



Not deployed

[No icon]

Deployed



Deployed, but pending configuration changes exist that have not been deployed.



Deployed, but software modifications exist that have not been deployed.



Applies only to redundant AppEngines. An AppEngine is undeployed, but its redundant pair is deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, but its redundant pair is not deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, its redundant pair is not deployed, and pending configuration changes exist that have not been deployed.



Applies only to redundant AppEngines. An AppEngine is deployed, its redundant pair is not deployed, and software modifications exist that have not been deployed.

[No icon]

Good



Warning



Error. The object is in an Error state and cannot be deployed.

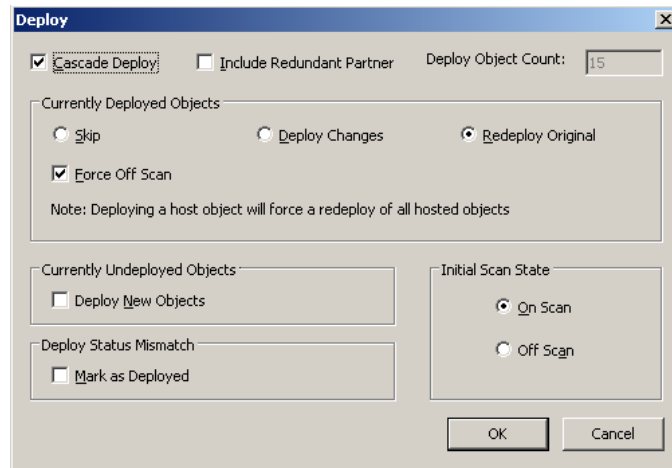


InTouchViewApp application files are being asynchronously transferred to the target node. This icon is normally visible for only a few moments at the end of an InTouchViewApp deployment operation, unless the object is deployed on a slow network.

Note This icon is larger than the other icons and completely replaces the original while it is being shown.

To deploy an object

- 1 Select the object in an **Application view**.
- 2 On the **Object** menu, click **Deploy**. The **Deploy** dialog box appears.



- 3 Select one or more of the following
 - **Cascade Deploy:** Select this check box to deploy the object selected for deployment as well as any objects it hosts. This option is selected by default if the object is a host. If you are deploying an individual host object, clear the check box. Objects being deployed across multiple platforms are deployed in parallel.
 - **Include Redundant Partner:** Select this check box to also deploy an AppEngine's redundancy partner object. This option is selected and unavailable when the redundant engine has pending configuration changes or software updates.
- 4 In the **Currently deployed objects** area, select one or more of the following options. These options are not available if the selected object has not been deployed before.
 - **Skip:** If one of the objects you are deploying is currently deployed, selecting **Skip** makes no changes to the already-deployed object.
 - **Deploy Changes:** If one of the objects you are deploying is currently deployed, this option updates the object in question with new configuration data. The run-time state from the run-time file is preserved and the state is modified with any changes.

- **Redeploy Original:** If one of the objects you are deploying is currently deployed, this option deploys the same version as previously deployed. For example, use this option to redeploy an object that is corrupted on the target computer.
 - **Force Off Scan:** If one of the objects you are deploying is currently deployed, this option sets the target object to off scan before deployment occurs.
- 5 In the **Currently undeployed objects** area, select the **Deploy New Objects** check box to start a normal deployment.
 - 6 In the **Deploy Status Mismatch** area, select the **Mark as Deployed** check box to mark the object as deployed in the Galaxy. A mismatch happens when the object is previously deployed to a target node, but the Galaxy shows the object is undeployed. Clear this option to redeploy the object to the target node.
 - 7 In the **Initial Scan State** area, select one of the following:
 - **On Scan:** Sets the initial scan state to on scan for the objects you are deploying. If the host of the object you are deploying is currently off scan, this setting is ignored and the object is automatically deployed off scan. When you deploying multiple objects, the deploy operation deploys all of the selected objects "off-scan." After all of the objects are deployed, the system sets the scan-state to "on-scan."

Objects can run only when both the host/engine is "on scan" and the object is "on scan." If either the host/engine or the object is "off scan," the object cannot run.

Always deploy Areas to their host AppEngines on scan. Because Areas are the primary providers to alarm clients, deploying Areas off scan results in alarms and events not being reported until they are placed on scan.

- **Off Scan:** Sets the initial scan state to off scan for the objects you are deploying. If you deploy objects off scan, you must use the ArcestrA System Management Console Platform Manager utility to put those objects on scan and to function properly in the run-time environment.

Note The System Management Console controls the state of the host/engine. The ObjectViewer controls the state of the objects.

The default scan setting is set in the **User Default** settings in the **Configure User Information** dialog box. For more information, see *Configuring User Information* on page 33.

- 8 Click **OK** to deploy the objects. The **Deploy** progress box appears. If you see error messages, see *Deployment Error Messages* on page 143. When the deploy is complete, click **Close**.

Deployment Error Messages

If the object being deployed has configuration problems that were not known during configuration, the **Deploy** progress box shows messages.

The **Progress** dialog box also shows the affected instances and any error and warning messages. The target object can take any actions necessary to achieve a valid state, including changing attribute values provided during deployment.

WinPlatforms are the only objects whose configuration designates its deployment location. Deployment problems unique to WinPlatforms are the following:

- The target computer could not be found on the network. Ensure the WinPlatform was configured properly and the target computer is properly connected to your network.
- Another WinPlatform is deployed already to the target computer. Resolve this problem by undeploying the existing WinPlatform before deploying the new one.
- The target platform is running on the old version of the product. To resolve this problem, the user must upgrade the remote platform.
- If a WinPlatform object is deployed on a slow network and it does not respond to the IDE before the 30-second message time-out, a communication error occurs and the object is shown as not deployed. You may need to adjust the message time-out for the WinPlatform object to accommodate the slow network speed.

Redeploying Objects

Redeploying is similar to deployment. While you are testing, you frequently redeploy your application to see changes you make. The redeploying process undeploys the object and then deploys it back.

You may have an object with a Pending Update deployment state. The Pending Update state means the object changed since its last deployment. When you deploy those changes, the new object is marked as the last deployed version in the Galaxy.

To redeploy

- 1 On the **Object** menu, click **Deploy**.
- 2 Follow the procedure for Deploying Objects on page 139.

Undeploying Objects

You may need to undeploy one or more objects. Undeploying removes one or more objects from the run-time environment.

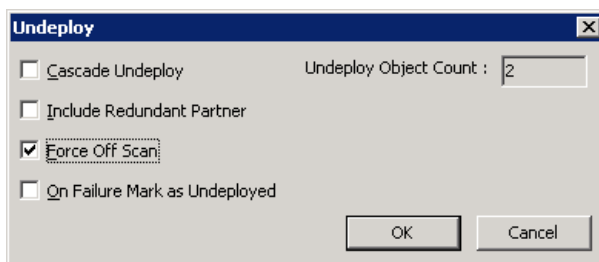
Before you start, you need to select the object or objects you want to undeploy in the IDE.

Before you delete or restore a Galaxy, undeploy all objects in the Galaxy.

Undeploying can fail if the target object has objects assigned to it. Make sure you select **Cascade Undeploy** in the **Undeploy** dialog box.

To undeploy

- 1 On the **Object** menu, click **Undeploy**. The **Undeploy** dialog box appears.



In the upper right of this dialog box, the **Undeploy Object Count** box shows the number of objects being undeployed. You can select a single object in **Application view** and, if you selected **Cascade Undeploy** and other objects are assigned to the selected object, the total number of objects appears in this box.

- 2 Select one or more of the following. Some of these options might not be available, depending on the kinds of object you select.
 - **Cascade Undeploy:** Select to undeploy the selected object as well as any objects it hosts.
 - **Include Redundant Partner:** Select to also undeploy an AppEngine's redundancy partner object.

Note The AppEngine in a redundant pair that was configured as the Primary can be undeployed alone because objects hosted by it run on the deployed Backup AppEngine, which becomes Active.

- **Force Off Scan:** If one of the objects you are undeploying is currently on scan, selecting **Force Off Scan** sets the target object to off scan before undeployment. If you do not select **Force Off Scan** and the target object is on scan, the undeployment operation fails.
- **On Failure Mark as Undeployed:** Marks the object as undeployed in the Galaxy when the object targeted for undeployment is not found.

Uploading Run-time Configuration

You can upload run-time configuration changes to the Galaxy database. This lets you keep any attribute values that changed during run time.

Note Upload run-time changes will not be permitted from old run-time node to the Galaxy

The values of certain attributes can be set in the configuration environment, but they can also be changed by the user at run time. As a result, the values of these attributes can differ between the run-time and configuration environments. These attribute types are:

- Writeable_UC
- Writeable_UC_Lockable
- Writeable_USC
- Writeable_USC_Lockable

For example, you create an object with a UDA `myInteger`. In the Object Editor, you specify an initial value of 30.

Then you deploy the object. At run time, you write a new value to `myInteger` of 31. If you redeploy this object, the original value of 30 overwrites any value assigned during run time. To avoid losing changes made during run time, upload changes before redeploying an object.

If you want to upload run-time changes to the Galaxy, make sure the selected objects are:

- Not edited and checked in since last deployment or upload
- Not in Pending Update state
- Checked in

Objects whose configuration are successfully uploaded have a new version number and a change log entry for the upload operation. The run-time object's version number also has a new version number. That version number matches the version in the configuration database.

If you select an object that is currently checked out to you, a warning appears during run-time upload. If you continue, you lose all configuration changes you made to the checked out object. The Galaxy performs an Undo Check Out operation on it before the run-time attributes are copied to the Galaxy database.

Note You cannot upload run-time changes for objects checked out to other users.

To upload run-time changes to the Galaxy

- 1 Select one or more objects in the **Model** view or **Deployment** view. For example, you could select an entire hierarchy from AppEngine down.
- 2 On the **Object** menu, click **Upload Runtime Changes**. The run-time attributes of the selected objects are copied over those in the Galaxy database.

Undeployment Situations

The following situations occur in these specified undeployment scenarios:

- After undeploying a WinPlatform, only the Bootstrap software remains on the target computer. All other WinPlatforms are notified of the undeployment of the WinPlatform and stop trying to communicate with it over the network.
- Alarm Clients know immediately that an area is undeployed and is no longer available. They remove the area from selection lists. Alarms associated directly with the area (not as a result of containment) are immediately removed from current alarm views.
- Undeploying an object that has Pending Updates status removes that status. It is now marked as undeployed.
- If cascade undeploy fails on one object, then the undeploy continues to the extent possible on other objects. The entire operation is not terminated because the undeploy fails for one object. However, a host might not be undeployable if one of its assigned objects cannot be undeployed.

Assume an ApplicationObject is hosted by the Active AppEngine in a redundant pair and a number of subscriptions is configured in that ApplicationObject that refers to items in a DIObject. If you undeploy the ApplicationObject in question, the items are not removed immediately from the item count of the DIObject. How fast those items are removed depends upon the value of the **Maximum time to maintain good quality after failure** option (Redundancy.StandbyActivateTimeout attribute) on the **Redundancy** page in the AppEngine's editor. This behavior does not apply to the undeployment of ApplicationObjects hosted by non-redundant AppEngines.

Chapter 7

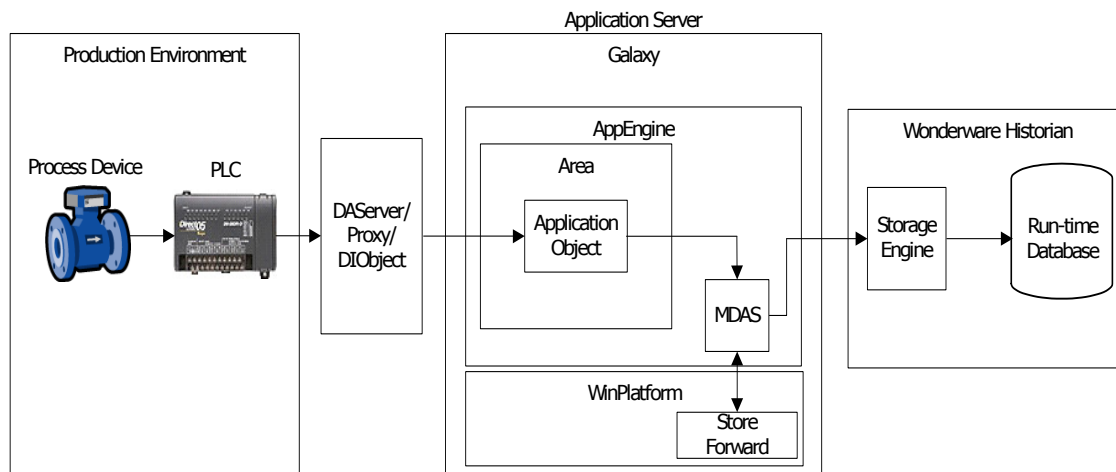
Working with History

You can configure application objects to store process data in the Wonderware® Historian (formerly called IndustrialSQL Server). Historical data from Application Server can be retrieved and viewed using standard Wonderware Historian database utilities. Also, you can use historical data to produce reports shown from your client applications. InTouch includes a set of ActiveX controls designed to show historical data from the Wonderware Historian in trends or graphs that can be embedded in client application windows.

Application Server History Components

To save your process data to a historical database, you must install the Wonderware Historian. A Wonderware Historian database can be installed on any computer outside the Galaxy, but on the same network. In a production environment, the Wonderware Historian should be installed on a dedicated computer without other ArchestrA products running on it.

The following figure shows the major ArchestrA components to save process data from a production device to the Wonderware Historian



A single Wonderware Historian installation can receive historical data from a single Galaxy. A push model is used to send and save new historical updates to Wonderware Historian. Each system object Engine (Platform, AppEngine, ViewEngine) includes a historian primitive that sends all history updates for all hosted objects to the historian. The historian primitive receives the history updates from history primitives, history extension primitives, and bulk history primitives on the same engine only. All Engine objects include an attribute to specify the node name of the computer hosting Wonderware Historian.

The figure shows a single Wonderware Historian. This may be a common configuration, but other Application Server configurations support multiple Wonderware Historian databases for a Galaxy. However, each Engine object only sends its historical data to one Wonderware Historian.

Sending Historical Data Between Application Server and Wonderware Historian

Application Server communicates with Wonderware Historian through an interface called Manual Data Acquisition Service (MDAS). MDAS uses Distributed COM (DCOM) to send data to the historian.

Wonderware Historian requires a tag to be configured in its database for each attribute of an Application Server object configured to save historical data. Thus, there is a one-to-one relationship between a historical object attribute and a tag in Wonderware Historian.

After the Wonderware Historian starts, it configures MDAS by sending its information about the tags (including their data sources) for which the MDAS is to acquire data. When the historian storage subsystem is ready to accept data, MDAS automatically connects to its data sources, starts acquiring data, and sends the data to be saved in the historical database.

Data received by the MDAS is temporarily stored in a series of 64 KB buffers on disk. The contents of the buffers are periodically sent to the Wonderware Historian in packets. MDAS sends data from a buffer after the buffer is full, or at a specified interval. The number of pre-allocated buffers is configurable for each MDAS.

Saving Historical Data During Communication Outages

A variety of problems can break communication between Application Server and Wonderware Historian. MDAS includes store forward capability, which protects against a temporary loss of historical data in the event that the MDAS cannot communicate with the Wonderware Historian.

If MDAS cannot communicate with the historian, all data currently being processed can be stored locally on the computer running MDAS. This hard drive location is called the store forward path and is configurable from the WinPlatform object.

If the MDAS is unable to send data to the historian, data is written to disk. Historical data is stored until the threshold capacity of the cache is reached or communication to the historian is restored. In the event that all store forward disk capacity is used to store historical data, no more data is stored. An error message is logged. Remote store forward paths are not supported.

Collecting Late Data

Application Server data that arrives with a timestamp more than 30 seconds later than the Historian's current time is considered *late data*. The Historian processes late data by several different methods.

- Data that is late, but is sent in steady stream. For example, because of communications delays, a topic defined in an DAServer might send a stream of data values that is consistently several minutes behind the Wonderware Historian time. Although this type of data is late, it is handled by the real-time data storage process if the topic is configured for late data.
- Data that is sent in periodic bursts. For example, a topic defined in an DAServer might represent an RTU that sends a block of data values every few hours. Late data of this type is handled by the "manual" data storage process.

Note Late data is enabled when an older Galaxy is imported or migrated to Application Server 3.1 or later.

The late data option must be enabled for the topic in order for late data to be processed. If the late data option is not enabled, data values are discarded that arrive with timestamps earlier than 30 seconds of the current Wonderware Historian time.

Saving Object Attribute Data to the Historian

Some attribute values can be saved as historical data when values change. Data quality and time are also associated with attribute values. When available, Application Server uses the extended attribute's value, the timestamp when the value changed, and the calculated quality of the data to create a Value, Time, Quality (VTQ) packet sent to the Historian. If there is no timestamp associated with the attribute value, the current scan time from the AppEngine hosting the object is used instead.

Saving Process Values as Historical Data

For attribute data to be stored in the historian, a host AppEngine must be configured to send history data to a Wonderware Historian node. For each object, you can configure attributes of the following data types to be saved to the Wonderware Historian.

- Float (numerical).
- Double (numerical) maps to a Wonderware Historian float data type. If the value of the double exceeds the range of a float, its value is clamped to the maximum value for the float and the quality is set to Uncertain.
- Integer (numerical)
- Boolean (non-numerical)
- String – Unicode (non-numerical). Limited to 512 characters. Characters that exceed the 512 maximum length are truncated. If truncation occurs, the quality value of the packet is set to Uncertain.
- CustomEnum (non-numerical) maps to a Wonderware Historian integer.
- ElapsedTime (numerical) maps to a Wonderware Historian float and is converted to seconds.
- Arrays or parts of arrays are not supported.
- Enum type attributes are saved as Wonderware Historian integer ordinal values.
- IEEE NaN values for float and double data types are converted to null values prior to being sent to the historian. NaN values are associated with a Bad OPC data quality.
- All numerical attributes sent to the Wonderware Historian are in the engineering units specified for the attribute. The Wonderware Historian does not scale numerical values.

Saving Data Quality as Historical Data

Wonderware Historian supports the OPC Quality definition. A 16-bit value for OPC data quality is sent to Wonderware Historian in the VTQ packet. Within the 16-bits, the low-order byte is the standard OPC portion. Wonderware reserves the high order byte and it is currently unused. The Good, Bad, Initializing (which is a form of Bad) and Uncertain quality states are specified in the low-order byte.

Any change in the mapped Wonderware Historian Quality Detail saves a new record to Wonderware Historian, regardless of whether the attribute value has changed. If an attribute value remains constant over a period of time with varying changes in quality, multiple records are stored in Wonderware Historian. The OPC quality stored in Wonderware Historian is the actual quality of the attribute in Application Server without modification.

Wonderware Historian can create and insert additional (non-ArchestrA generated) VTQ records that modify quality in the database. These additional VTQ records provide additional information about network outages or other events to client applications that use historical data from Wonderware Historian.

Additional Quality Data Saved to the Historian

Wonderware Historian also has two additional data quality fields (quality and quality detail) that are non-OPC compliant. Both fields are stored with each record in addition to the OPC quality. Wonderware Historian maps the value of these fields in a manner consistent with its quality definition for those fields. The values of those fields are driven by the OPC quality sent by Application Server. For example, an OPC Good quality results in Wonderware Historian quality detail of Good.

Wonderware Historian Quality is a 1-byte flag that summarizes the quality of the associated data value in the packet sent by Application Server. Wonderware Historian Quality is an enumerated type with the following values:

Hex	Decimal	Name	Description
0x00	0	Good	Good value.
0x01	1	Bad	Value was marked as invalid.
0x10	16	Doubtful	Value is uncertain. Only fabricated at retrieval time.
0x85	133	Initial Value	(Good) Initial value for a delta request.

OPC Qualities from Application Server result in storage of an Wonderware Historian Quality as follows:

OPC Quality (substatus form)	Wonderware Historian Quality Decimal (1-byte)
Good (any)	0 - Good
Bad (any)	1 - Bad
Bad (Initializing form)	1 - Bad
Uncertain (any)	0 - Good

Wonderware Historian QualityDetail contains the detail of data quality as a 32-bit value. QualityDetail is derived from OPC quality. Each source is allocated a byte position within the QualityDetail 32-bit word, which is formatted as:

0xXXRRDDDD

Only the low-order two bytes (DDDD) are mapped to OPC quality for data sent by Application Server and saved to the historian:

OPC Quality (any substatus form)	Results in any one of these Wonderware Historian Quality Details
Good (any)	192 - Good value 150 - Initial value (good) 151 - Initial store/forward value (good) 252 - First value received (good) 44 - First good value received by storage after the connection to MDAS has been restored. 248 - First value in the second stream during a store-and-forward.
Bad (any)	24 (DAServer disconnect)
Bad (Initializing form)	24 (DAServer disconnect)
Uncertain (any)	192

Saving the Data Timestamp as Historical Data

Application Server includes a timestamp in the VTQ packet sent to the historian for each attribute value/quality update that is saved as a historical record. Application Server propagates the timestamp associated with an attribute value. The timestamp is propagated throughout all Application Server components as UTC in FILETIME format.

If an object attribute has a Value DeadBand specified, a quality change or value change from previous scan cycle greater than Value Deadband causes the attribute's Value, Time and Quality (VTQ) to be saved as historical data in the Historian. When a Value DeadBand is not specified, any change to the attribute's value, timestamp, or quality cause the attribute to be saved as historical data.

When an attribute is configured to periodically save a historical record, the attribute's current time is used as the timestamp. If the attribute does not support a timestamp, the hosting AppEngine's current scan time is used as timestamp included in the packet.

If slow updates for real-time data values are received from an I/O source and an intermittent network disconnect occurs, the timestamp of the first data value of a historized attribute uses the AppEngine's timestamp instead of the source's timestamp. In the following example, in row number 4, the value "30" is sent again to the historian, but the timestamp is the AppEngine timestamp and not the actual timestamp of "2008-11-24 18:50:41.061" when the value of 30 was generated from the I/O source.

DateTime	TagName	Value	Quality	QualityDetail	OPCQuality
2008-11-24 18:48:30.081	Realtime_ Client.ival	20	133	44	192
2008-11-24 18:50:41.061	Realtime_ Client.ival	30	0	192	192
2008-11-24 18:51:38.666	Realtime_ Client.ival	NULL	1	24	0
2008-11-24 18:52:06.670	Realtime_ Client.ival	30	0	192	192 → Row number 4
2008-11-24 18:53:22.988	Realtime_ Client.ival	40	0	192	192

If slow updates for late data values are received from an I/O source and an intermittent network disconnect occurs, the timestamp of the first data value of a historized attribute is modified to maintain the time sequence. In the following example, in row number 4, the value "30" is sent again to the historian, but the timestamp is modified by adding 5 milliseconds to previous timestamp of the NULL data value. Note that a QualityDetail of 704 is stored for the data value in row number 4.

DateTime	TagName	Value	Quality	QualityDetail	OPCQuality
2008-11-24 18:48:30.081	Latedata_ Client.ival	20	133	44	192
2008-11-24 18:50:53.624	Latedata_ Client.ival	30	0	192	192
2008-11-24 18:50:53.625	Latedata_ Client.ival	NULL	1	24	0
2008-11-24 18:50:563.630	Latedata_ Client.ival	30	0	704	192 → Row number 4
2008-11-24 18:53:26.863	Latedata_ Client.ival	40	0	192	192

Saving Alarms and Events as Historical Data

Alarms and events detected by the Application Server at run time are saved as historical data to the Wonderware Historian for the host engine. Alarms are a type of event that can be historized. In addition, alarm-related events such as Acknowledge are also saved as historical alarm data.

Deploying and Undeploying Attributes

When an object is deployed with historical attributes, the history primitives cause dynamic reconfiguration of Wonderware Historian. Each history primitive causes a new tag to be created and configured automatically in Wonderware Historian at deployment time. The Wonderware Historian storage system that will be used is determined by the configuration of the host engine object.

This dynamic configuration causes appropriate row/column configuration in the Wonderware Historian schema.

If the connection to the Wonderware Historian is down at deploy time, the attempt to dynamically reconfigure Wonderware Historian is achieved when the connection is restored. In other words, automatic retry is built in. However, any data generated by the object during the time between deployment and the recovery is lost and cannot be recovered by store forward.

When an object configured for history is redeployed, changes to an object's attributes makes the historian reconfigure storage. For example, if the engineering units string of an object changes from "Deg F" to "Deg C" when the object is redeployed, the Wonderware Historian configuration database shows the change.

When an object configured for history is undeployed, all history remains in the Wonderware Historian. The history data can be viewed in the future even if the object is no longer deployed.

Saving Historical Data During Run Time

While an application is running, data is saved to the historian as follows:

- If no previous historical attribute value exists in the historian, the first value is always saved to the historian.
- Numerical attributes (double, float or integer): If the value for the attribute changes and that change is more than the value deadband, or the value's quality changes (for example, from Good to Bad), the data is saved to the historian.
- If the attribute type is qualitative (enumeration, string, or Boolean) and the attribute's value or quality changes.
- If the current time exceeds the last Forced Store period occurred for the attribute by the time period that was set as the Force Storage Period. If no last Forced Store occurred since starting the object, a new store occurs immediately.

Note If enabled, the Value Deadband mechanism resets itself based on this new stored value.

- All new attributes value, timestamp, and quality are sent to the Wonderware Historian for storage.
- Wonderware Historian stores the historical records to the database.

Advanced Communication Management

To ensure that attribute data saved to the historian is always current and continuous, Application Server registers a reference to the attribute. By registering a reference to the attribute, it prevents Message Exchange from suspending historical updates during the period when the client application has minimized the window containing the object. Historical data continues to be saved to the Wonderware Historian even during the period when the object is in an Advanced Communication Management Suspended state.

Store Forward Mode

If the Wonderware Historian shuts down or the network connection to it is lost while an application is running, historical data continues to be stored locally on the computer hosting the WinPlatform object.


When the Wonderware Historian node recovers, data is forwarded from the local node to the Wonderware Historian node at a low priority.


If an AppEngine loses connectivity to the Wonderware Historian node, the Wonderware Historian reports bad data quality to clients. When you undeploy an object with attributes configured for history, the Wonderware Historian stores the final data points with Bad quality.











Configuring Common Historical Attributes

Each application or system object that you select from the Template Toolset include a set of common attributes you configure to save data to the Historian.


The historical attributes you configure are based on the data type of the object. For example, the following figure shows the common historical attributes for numerical attributes.


Historize PV 


History 





Force storage period:	<input type="text" value="300"/>	ms 	Trend High:	<input type="text" value="6000.0"/>	EU 
Value deadband:	<input type="text" value="5.0"/>	EU 	Trend Low:	<input type="text" value="2500.0"/>	EU 
Interpolation Type:	<input type="text" value="Linear"/>	 	Sample Count:	<input type="text" value="5"/>	
Rollover Value:	<input type="text" value="500.0"/>		<input checked="" type="checkbox"/> Enable Swinging Door		
			Rate DeadBand:	<input type="text" value="25.0"/>	% 


If you are configuring a discrete attribute, you see a smaller set of historical attributes.

Enable inputs 

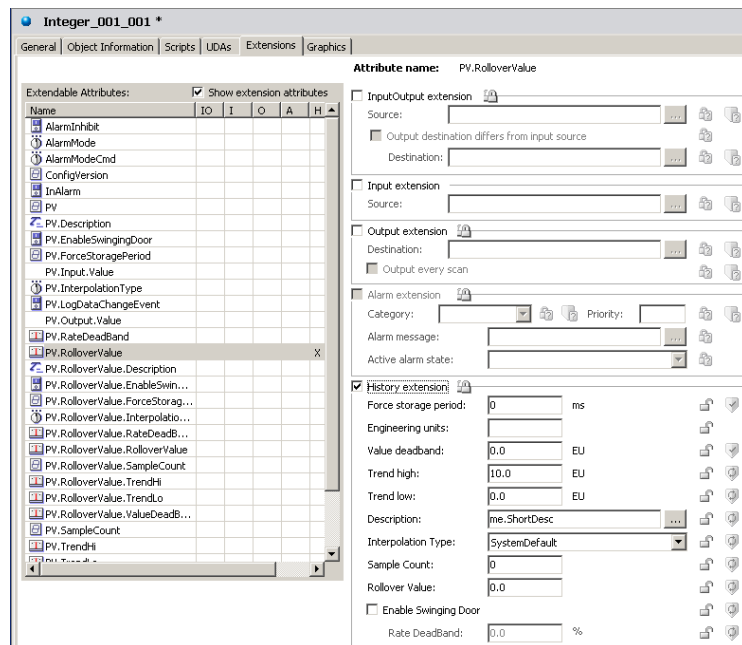
PV 

Historize PV 

Force storage period:	<input type="text" value="0"/>	ms 		Sample Count:	<input type="text" value="65"/>		
-----------------------	--------------------------------	--	---	---------------	---------------------------------	---	---

Generate event upon PV change 

You can also assign historical extensions to extended attributes that you select from the **Extensions** tab of your objects.



You select the **Show extensions attributes** check box to show the extensions of an application object's attributes in the list. Then, you select an extendable attribute from the list whose data you want to save to the historian.

After you select the **History extension** check box to save the data to history associated with the attribute extension you selected. An **X** appears in the **H** column of the **Extendable Attributes** list to indicate its data is saved to the historian.

This section describes the common historical attributes that you can configure for your system and application objects. Refer to this section as you complete the procedures to configure your objects.

- **Force Storage Period**

Interval in milliseconds in which an attribute value is saved to the Historian, regardless of whether the value exceeds its value deadband setting or not. An attribute value is saved to the Historian at every Force Storage interval. The default value of zero (0) disables the Force Storage period.

- **Value Deadband**

Threshold value in engineering units, which is the absolute difference between the current and most recent saved historical values. The current value must exceed the absolute deadband to be saved as historical data.

The Value Deadband filters out small, momentary value changes from being saved to the Historian. A new value that is within the range of the deadband is not saved to the Historian. The default value of 0.0 disables the Value Deadband and any change to a value is saved as historical data.

- **Interpolation Type**

List of methods used by the Historian to interpolate analog historical data. The interpolation type determines which analog value is selected during a Historian data retrieval cycle. Interpolation types include System Default, Stairstep, or Linear. The default interpolation type is System Default.

- **Stairstep**

No interpolation occurs. The last known value is returned with the given cycle time. If no valid value can be found, a NULL is returned to the Historian.

- **Linear**

The Historian calculates a new value at the given cycle time by interpolating between the last known value prior to the cycle time and the first value after the cycle time.

- **System Default**

The Wonderware Historian system-wide interpolation setting. The system-wide setting must be either stairstep or linear interpolated.

- **Rollover Value**

Positive integer value that represents a tag's reset limit when the Historian operates in counter retrieval mode. In counter retrieval mode the Historian uses a tag's rollover value to calculate and return the delta change between consecutive retrieval cycles. The default value is 0.0.

The Rollover value applies only to numeric attributes. The Rollover value is disabled if the historical attribute data type is Boolean or a string.

- **Trend Hi**

Initial maximum trend value in engineering units for clients. The default is 100.

- **Trend Lo**

Initial minimum trend value in engineering units for clients. The default is 0.0.

- **Sample Count**

Integer that indicates the number of samples the Historian should store in the active image for an attribute value. Acceptable range of values is from 0 to 9999. The Historian enforces a sample count of 65 for the active image.

If the current sample count for the active image is 65, and you set the sample count to a value less than or equal to 65, the new sample count setting is not saved to the Historian database, and the active image count remains at 65.

If the current sample count for the active image is greater than 65, and you set the sample count to a value less than or equal to 65, the new sample count is saved to the Historian database and 65 is used by the active image.

Regardless of the current sample count for the active image, if you set the sample count to a value greater than 65, the new sample count is saved to the Historian database and used by the active image.

- **Enable Swinging Door**

A flag that indicates whether the swinging door rate deadband is enabled or disabled. The default is disabled.

Enable Swinging Door is disabled if the historical attribute data type is Boolean or a string.

- **Rate Deadband**

Percentage rate of change deadband based on the change in the slope of incoming data values to the Historian. For example, specifying a swinging door rate deadband of 10 percent means that data is saved to the Historian if the percentage change in slope of consecutive data values exceeds 10 percent.

The default is 0.0, which indicates a swinging door rate deadband is not applied. Any percentage greater than 0.0 can be assigned to the rate deadband.

Configuring System Objects to Store Historical Data

The following table shows the historical attributes for Application Server system objects.

History Attributes	Application Server System Objects				
	WinPlatform	AppEngine	Area	ViewEngine	InTouchViewApp
Buffer count	•	•		•	
Description	•		•	•	
Enable Late Data		•			
Enable Storage to Historian	•	•		•	
Enable Swinging Door	•	•	•	•	
Enable Tag Hierarchy	•	•		•	
Engineering units	•	•	•	•	
Force Storage Period	•	•	•	•	
Forwarding chunk size	•	•		•	
Forwarding delay	•	•		•	
Historian	•	•		•	
History store forward directory	•	•		•	
Idle duration		•			
Interpolation Type	•	•	•	•	
Process Interval		•			
Rate Deadband	•	•	•	•	
Rollover Value	•	•	•	•	
Sample Count	•	•	•	•	
Store forward deletion threshold	•	•		•	

History Attributes	Application Server System Objects				
	WinPlatform	AppEngine	Area	ViewEngine	InTouchViewApp
Store forward minimum duration	●	●		●	
Trend High	●	●	●	●	
Trend Low	●	●	●	●	
Value Deadband	●	●	●	●	

Configuring the WinPlatform Object to Store Historical Data

A WinPlatform object contains attributes to configure how historical store forward data is cached locally and then pushed to the Historian. Also, you can select the WinPlatform object's platform, scheduler, and engine data to be saved to the Historian. Finally, you can select the WinPlatform's extendable attribute values to be saved to the Historian.

The following table shows the WinPlatform attributes you can configure to save data to the Historian. The table shows the attributes by the tabbed pages when you edit WinPlatform attributes.

Attribute Name	Attribute Description
General Tab	
Network Address	Node name of the computer hosting the WinPlatform object.
History store forward directory	Folder location to temporarily save store forward data sent to the Historian. The name of the AppEngine is always appended to the SF directory name. For example, if you specify the store-forward location for AppEngine001 as C:\SFData, the actual folder name is C:\SFData\AppEngine001. Appending the AppEngine name to the store forward folder ensures unique store forward directory names for each AppEngine.
Engine Tab	
Enable storage to historian	Checkbox to select data to be saved to the Historian.
Enable Tag Hierarchy	Check box to select if you want the portion of the model view hierarchy hosted by the AppEngine to get replicated into the public group namespace when the Historian engine starts.
Historian	Node name of the computer that hosts Wonderware Historian.
Store forward deletion threshold	Size in MB to reserve free on the store forward disk and not use during store forward.
Store forward minimum duration	Minimum duration in seconds the MDAS remains in Store Forward mode to send data to the Historian. After the MDAS enters store forward mode, it continues to send store forward data until the condition requiring store forward disappears, or the minimum store forward period has elapsed, whichever comes later.
Forwarding chunk size	Number of bytes of data sent to the MDAS in a single chunk as store forward data.
Forwarding delay	Interval length in milliseconds between sending chunks of store forward data to MDAS.

Attribute Name	Attribute Description
Buffer count	Number of 64K buffers used to store store forward data.

Platform History, Scheduler History, and Engine History Tab

For more information about the historical attributes that can be configured from these tab pages of the WinPlatform object, see *Configuring Common Historical Attributes* on page 160.

To configure a WinPlatform Object to store historical data

- 1 Double-click on an WinPlatform derived template or instance to open the Object Editor.
- 2 Click the **General** tab to show history attributes.

The screenshot shows the Object Editor with the 'Platform History' tab selected. The following fields are visible:

- Network address: TankHistorySvr
- History store forward directory: D:\StoreForward
- Minimum RAM: 1024 MB

- 3 In the **Network address** box, type or select the node name of the computer assigned to the WinPlatform object.
- 4 In the **History store forward directory** box, enter the path to the folder to save store and forward historical data. The folder must exist on the computer specified in the **Network address** box.
- 5 Click the **Engine** tab to show store forward history attributes.

The screenshot shows the Object Editor with the 'Engine' tab selected and the 'History' area expanded. The following fields and checkboxes are visible:

- Enable storage to historian
- Enable Tag Hierarchy
- Historian: TankLine_02
- Store forward deletion threshold: 100 MB
- Store forward minimum duration: 0 s
- Forwarding chunk size: 1024 Bytes
- Forwarding delay: 250 ms
- Buffer count: 128

- 6 Assign values to the Store Forward attributes shown in the **History** area of the **Engine** page.
 - a Select the **Enable storage to historian** checkbox.
 - b In the **Historian** box, type or select the node name of the computer that hosts the Historian.

- c In the **Store forward deletion threshold** box, type the amount of data in megabytes to save in the Store Forward folder before overwriting currently stored data with new data.
 - d In the **Forwarding chunk size** box, type the number of bytes of data sent to MDAS from the Store Forward folder.
 - e In the **Forwarding delay** box, type the delay in milliseconds before sending data from the Store Forward folder to MDAS.
 - f In the **Buffer count** box, type the number of 64K buffers in the Store Forward folder to temporarily store data before sending it to the MDAS.
- 7 Click the **Platform History** tab to show the attributes to save system data from the computer hosting the WinPlatform object to the historian.

The screenshot shows the 'Platform' configuration window with two sections for historical attributes. The first section, 'Historize cpu load of platform', is checked and includes fields for Force storage period (0 ms), Value deadband (0.0 %), Interpolation Type (SystemDefault), Rollover Value (0.0), Trend high (100.0 %), Trend low (0.0 %), Sample Count (65), and Rate DeadBand (0.0 %). The second section, 'Historize average interrupts of platform', is also checked and includes fields for Force storage period (0 ms), Value deadband (0.0 ms), Interpolation Type (SystemDefault), Rollover Value (0.0), Trend high (100.0 ms), Trend low (0.0 ms), Sample Count (65), and Rate DeadBand (0.0 %). Each field has a small icon to its right, likely for locking or unlocking the value.

- 8 Select the checkbox next to each attribute whose data you want saved in the historian. For more information about these historical attributes, see *Configuring Common Historical Attributes* on page 160.
- 9 Click the **Scheduler History** tab to show the attributes to save data to the historian.
- 10 Select the checkbox next to each scheduler attribute whose data you want saved in the historian.
- 11 Click the **Engine History** tab to show the attributes to save data from AppEngines hosted by the WinPlatform object.
- 12 Select the checkbox next to each engine attribute whose data you want saved in the historian.
- 13 Save your changes and check in the WinPlatform object.
- 14 Deploy the object to its target computer in an on scan state.

Configuring an AppEngine Object to Store Historical Data

If an AppEngine is deployed before Wonderware Historian is started, no history data is stored until the objects successfully register with the Wonderware Historian.

Note Except for Late Data, the ViewEngine object contains the same historical attributes as the AppEngine object.

Setting Up an AppEngine for Late Data

If you enable late data for an AppEngine object, you need to configure two parameters:

- **Idle duration**

The idle duration is the delay in seconds after receiving the last data value before the contents of the MDAS buffer are sent to the Historian. For example, an idle duration of 60 seconds means that data from the topic is cached and only processed by the historian after no more data has been received from the MDAS for at least 60 seconds. The default length of the idle duration is 60 seconds.

The idle duration is important if you anticipate bursts of late data being sent to the MDAS. The length of the idle duration depends on your specific historian implementation and application requirements. The longer you set the idle duration, the longer it takes to see the data in your client applications.

- **Process interval**

The process interval is how often Wonderware Historian process buffers from MDAS. The default is 120 seconds.

The process interval ensures that the Historian receives historical data even if the idle duration is never satisfied. The Historian processes data from the MDAS at least once every processing interval.

By default, the process interval is set to twice the length of the idle duration. You cannot set the processing interval to a value less than the idle duration.

To configure an AppEngine to store historical data

- 1 Double-click on the AppEngine instance to open it within the Object Editor.
- 2 Click on the **General** tab to show history attributes.

History

Enable storage to historian

Enable Tag Hierarchy

Historian:

Store forward deletion threshold: MB

Store forward minimum duration: s

Forwarding chunk size: Bytes

Forwarding delay: ms

Buffer count:

Enable Late Data:

Idle duration: s

Process Interval: s

- 3 In the **History** area, select the **Enable storage to historian** check box.
- 4 Select the **Enable Tag Hierarchy** check box if you want the portion of the model view hierarchy hosted by the AppEngine to get replicated into the public group namespace when the Historian engine starts.
- 5 In the **Historian** box, type or select the node name of the computer hosting the Wonderware Historian.
- 6 Specify the characteristics of store forward historical data by setting values for the following attributes:
 - a In the **Store forward detection threshold** box, type the size in megabytes to reserve free on the MDAS store forward disk and not use during store forward.
 - b In the **Store forward minimum duration** box, type the minimum duration in seconds for MDAS to remain in Store Forward mode.
 - c In the **Forwarding chunk size** box, type the number of bytes of data sent by the MDAS in a single chunk as store forward data to the historian.
 - d In the **Forwarding delay** box, type the number of milliseconds between sending chunks of bytes when forwarding store forward data to the historian
 - e In the **Buffer count** box, type the number of 64K MDAS buffers to be pre-allocated for buffering data sent to the historian.

- 7 If you want the MDAS to send late data to the historian, select the **Enable Late Data** check box. The **Idle duration** and **Process Interval** boxes remain inactive until you select the **Enable Late Data** check box.
- 8 In the **Idle duration** box, type the number of seconds the MDAS waits after receiving the last value before sending the contents of the buffer to the historian.
- 9 In the **Process Interval** box, type the number of seconds between each interval when the historian processes the contents of the MDAS buffers.
- 10 Click the **Scheduler History** tab to show the attributes to save data to the historian.
- 11 Select the checkbox next to each scheduler attribute whose data you want saved in the historian.
- 12 Click the **Engine History** tab to show the attributes to save data from AppEngines hosted by the WinPlatform object.
- 13 Select the checkbox next to each engine attribute whose data you want saved in the historian.
- 14 Save your changes to the AppEngine object.

Configuring an Area Object to Save Alarm Counts as Historical Data

An Area object represents a plant area to group objects for modelling and report alarms. You can configure a set of attributes to save the counts of an area's alarm states to the historian. An Area object contains a set of attributes to save the counts of the following alarm states to the historian:

- Active alarms
- Unacknowledged alarms
- Disabled or silenced alarms

To configure an Area object to store historical data

- 1 Double-click on the Area instance to open it within the Object Editor.
- 2 Click on the **General** tab to show attributes for area alarm counts that can be saved as historical data.

The screenshot shows the 'General' tab of the Object Editor. It contains three sections, each with a checkbox to enable historical data storage for a specific alarm state. Each section has several configuration fields:

- Historize active alarm counter:**
 - Force storage period: [] ms
 - Value deadband: [] EU
 - Interpolation Type: []
 - Rollover Value: []
 - Trend High: [] EU
 - Trend Low: [] EU
 - Sample Count: []
 - Enable Swinging Door:
 - Rate DeadBand: [] %
- Historize unacknowledged alarm counter:**
 - Force storage period: [] ms
 - Value deadband: [] EU
 - Interpolation Type: []
 - Rollover Value: []
 - Trend High: [] EU
 - Trend Low: [] EU
 - Sample Count: []
 - Enable Swinging Door:
 - Rate DeadBand: [] %
- Historize disabled (or silenced) alarm counter:**
 - Force storage period: [] ms
 - Value deadband: [] EU
 - Interpolation Type: []
 - Rollover Value: []
 - Trend High: [] EU
 - Trend Low: [] EU
 - Sample Count: []
 - Enable Swinging Door:
 - Rate DeadBand: [] %

- 3 Select the checkbox next to each alarm state counter that you want to save to the historian.
- 4 Set the attributes for each alarm counter that you select. For more information about assigning values to historical attributes, see [Configuring Common Historical Attributes](#) on page 160.
- 5 Save your changes and close the Object Editor.

Configuring Application Objects to Save Historical Data


The following table shows the historical attributes for application objects. These history attributes are described in Configuring Common Historical Attributes on page 160.


History Attributes	Application Server Application Objects										
	AnalogDevice	Boolean	DiscreteDevice	Double	FieldReference	Float	Integer	Sequencer	String	Switch	UserDefined
Enable Swinging Door	●	●		●	●	●	●				
Force Storage Period	●	●	●	●	●	●	●		●	●	
Interpolation Type	●	●		●	●	●	●				
Rate Deadband	●	●		●	●	●	●				
Rollover Value	●	●		●	●	●	●				
Sample Count	●	●	●	●	●	●	●		●	●	
Trend High	●	●		●	●	●	●				
Trend Low	●	●		●	●	●	●				
Value Deadband	●	●		●	●	●	●				










Note The historical attributes available from the UserDefined object are based on the data type associated with the selected field attribute.

To configure an application object to store history

- 1 Open the application object in the Object Editor.
- 2 Click the **General** tab to show the history attributes for the object's PV value.

Historize PV 

History 

Force storage period:	<input type="text" value="0"/>	ms 	Trend High:	<input type="text" value="100.0"/>	EU 
Value deadband:	<input type="text" value="0.0"/>	EU 	Trend Low:	<input type="text" value="0.0"/>	EU 
Interpolation Type:	<input type="text" value="SystemDefault"/>		Sample Count:	<input type="text" value="0"/>	
Rollover Value:	<input type="text" value="0.0"/>		<input type="checkbox"/> Enable Swinging Door		
			Rate DeadBand:	<input type="text" value="0.0"/>	% 

If you selected an AnalogDevice object, click the **History** tab. If you selected a UserDefined object, click the **Field Attributes** tab.

- 3 Select the **Historize PV** check box to enable the common history attributes shown in the **History** area of the page.

If you selected an AnalogDevice object, the History page includes check boxes to save **PV**, **SP**, **PV mode**, **Control mode**, and **Control Track Flag** data to the historian.

If you select a UserDefined object, create a field attribute and then select the **Enable history** check box. If you created a discrete attribute, you see two history attributes after selecting **Enable history**. You see nine history attributes if you created an analog field attribute.

- 4 Assign values to the PV history attributes that appear in the **History** area of the page. For more information about assigning values to the common history attributes, see [Configuring Common Historical Attributes](#) on page 160.
- 5 Set the attributes for each attribute extension that you select. For more information about assigning values to the common historical attributes, see [Configuring Common Historical Attributes](#) on page 160.
- 6 Save your changes and close the Object Editor.

Chapter 8

Working with Alarms and Events

You can create ArcestrA applications that generate alarms and events to provide the status of a running application.

- Alarms warn about process conditions that can potentially cause problems. Typically, you set up an alarm to become active when a process value exceeds a defined limit. For example, you can set an alarm for a pump that warns when no fluid pressure is detected.

An alarm is an abnormal condition that requires immediate attention. An operator usually acknowledges an alarm. ArcestrA handles the real-time reporting of alarms and provides special clients for viewing them.

- Events represent normal system, application, or user occurrences that produce status messages. A typical event occurs when an operator logs on to an application at the beginning of a work shift. Application Server can detect events, store them as historical data, and report them to client applications.

ArcestrA objects include built-in event and alarming reporting capabilities. You must configure alarms for each object in the IDE to use the event and alarm functions.

Understanding Events

An event indicates a significant occurrence that is detected, reported, and saved as historical data. Events can be detected by any Application Server component including automation objects and the SMC.

Events are more general purpose than alarms and provide a means for any software component to log information about a system, application, or operator action. Application Server components use an event API to send event messages to Alarm/Event distributors and the Wonderware Historian. Also, events are sent to client applications like InTouch HMI to be shown in real-time trends or reports.

Types of Events

Application Server creates several types of run-time event messages, which are saved as historical data.

- System events

Event messages are logged when a Platform, Engine, Area, DI Network or DI Device start or stop. Any system action that affects a large number of objects is logged as an event.

A System event message contains the following information:

- Event type, which is System.
- Tagname, which is the name of the object generating the event.
- Tag description, which is a short description of the object generating the event.
- Area, which is the name of the area that contains the object generating the event.
- Event description, which describes the system action and can be either Started or Stopped.
- Timestamp, which is the current system time.
- Security-audit (operator change) events

Event messages are logged to the SMC when an operator logs on to or logs off from an application. Also, security-audit events are logged when an operator changes actions by means of User sets.

A Security-audit event message contains the following information:

- Event type, which is operator change.
- Timestamp, which is the date and time when the operator change event occurred. The timestamp of the event is the current AppEngine scan time.
- Tagname, which is the name of object generating the event.
- Prim.attr, which is the reference string of the attribute being changed.
- Tag description, which is a short description of the object.
- Area, which is the name of the area that contains the object generating the event.
- UserEngine, which is the name of the view engine or other user engine requesting the operator change.
- Operator1, which is the full name of the primary operator requesting a change. The full name is an attribute of the UserProfile.
- Operator2, which is the full name of the secondary operator validating the change, if any.
- Old value, which is the previous value of an attribute.
- New Value, which is the new value of an attribute.
- Application (or process) related events

Application event messages are generated by application objects in response to process actions. For example, application event messages are created when a process pump starts or stops.

An Application event contains the following information:

- Event type, which is data change.
- Timestamp, which is the date and time when the application event occurred. The timestamp assigned to the event is the timestamp of the attribute associated with the event, if available. Otherwise, the event timestamp is the current AppEngine scan time.
- Tagname, which is the name of the object generating the event.

- Prim.attr, which is the reference string of the attribute being changed.
- Tag description, which is a short description of the object.
- Area, which is the name of the area that contains the object generating the event.
- Old value, which is the previous value of an attribute.
- New Value, which is the new value of an attribute.

Understanding Alarms

Application and system objects can detect and generate an alarm. To detect an alarm, a system or application object sets a Boolean Attribute flag to indicate whether the object's alarm condition is currently true or false.

To report an alarm, the object must contain an Alarm Primitive. The Alarm Primitive makes a reference to the object's Boolean flag to determine whether the alarm condition is true. It then combines this information with the current alarm mode to determine whether to report a this as an active or inactive alarm state. An Alarm Primitive is dedicated to reporting a single alarm condition's state. Alarm Primitives send alarm notification messages to ArcestrA alarm and event distributors.

Every alarm notification includes a set of fields containing data that describes the alarm. Some alarm notification data is saved as historical data. The following list describes all fields sent with an alarm notification.

- TagName, which is the name of the object generating the alarm. Saved as historical data.
- Name, which is the name of the alarm. Saved as historical data.
- InAlarm, which is a Boolean value that indicates whether the object's alarm state is currently active or inactive. Saved as historical data.
- Quality, which is the current quality of the data upon which the alarm is based. Saved as historical data.

- **OnTimeStamp**, which is the time when the attribute value transitioned into an alarm state. The attribute's value timestamp is used, if available. Otherwise, the timestamp is the AppEngine scan time. Saved as historical data.
- **OffTimeStamp**, which is the time when the alarmed attribute value returns to normal. The attribute's value timestamp is used, if available. Otherwise, the **OffTimeStamp** is the AppEngine scan time. If an active alarm is disabled, it forces a return to normal and the timestamp is the current time. Saved as historical data.
- **Category**, which is an integer between 1 and 14 that identifies the type of alarm and source of the alarm. These values are associated with Internationalized category labels set by Wonderware.

Alarm category labels can be localized to other languages. Application Server uses the default Galaxy language to retrieve these strings and send them to InTouch. The alarm category labels appear in InTouch and InTouch history as the default Galaxy language strings. Saved as historical data.

- **Priority**, which is an integer value from 1 to 999 indicating the severity of the alarm. An alarm priority of 1 is most urgent and 999 least urgent.
- **TargetValueReference**, which is an optional field that makes a reference to the target value of the alarm. Not saved as historical data.
- **ActualValueReference**, which is an optional field that makes a reference to the actual attribute value for the alarm condition. Not saved as historical data.
- **TargetValue Snapshot**, which is an optional field containing the attribute's target value at the time when the alarm became active. Saved as historical data.
- **ActualValueSnapshot**, which is an optional field containing the attribute's actual value at the time when the alarm became active. Saved as historical data.

- EngUnitsReference, which is the reference to the engineering units string for the condition. Saved as historical data.
- AcknowledgedFlag, which indicates whether the alarm is acknowledged or not. If this flag is FALSE, the alarm is still unacknowledged. Saved as historical event data.
- AcknowledgeTime, which indicates the time when the alarm was acknowledged if the AcknowledgedFlag is TRUE. Saved as historical data at the time of acknowledgement.
- AcknowledgeUserId, which is the string containing the name of the user who acknowledged the alarm. Saved as historical data at the time of acknowledgement.
- AlarmMode, which indicates whether the alarm mode is Enabled, Silenced or Disabled. Saved as historical data at the time when the alarm mode changes.
- Message text describing the alarm, which can be statically or dynamically constructed. The message typically contains the alarm description, the exceeded limit value, and possibly the variable value. For the standard alarm utility primitive, this message is retrieved by means of a reference to a string/international string attribute in the object. The reference is setup at ObjectDesigner time for standard alarm primitives. If none is specified, then the common primitive short description attribute is utilized. This field is also provided in the alarm extension primitive and can be dynamically generated and scripted. Saved as historical data.
- Area, which is the name of the area that contains the object generating the alarm. Saved as historical data.

Types of Alarms

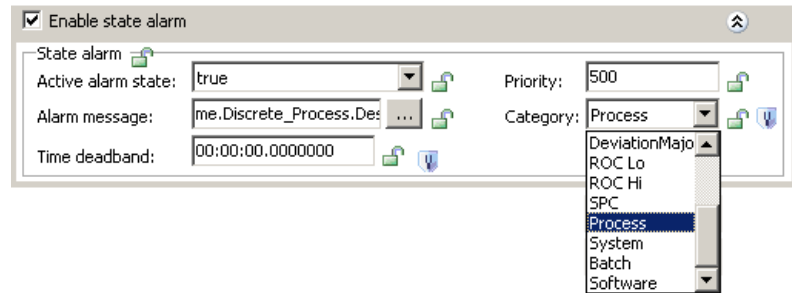
Application Server supports the following types of alarms:

- State alarms, which are also known as Boolean alarms
- Limit alarms
- Target deviation alarms
- Rate of change alarms

The type of alarm that you can configure is based on the data type of the attribute's value.

State Alarms

A state alarm corresponds to a discrete tag with two possible states. When you create a state alarm, you configure whether the active alarm state corresponds to the TRUE or FALSE state of the attribute.

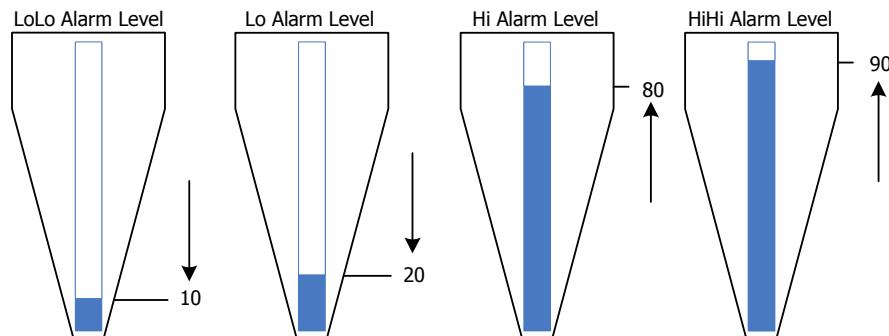


You can set an alarm message and Priority for a state alarm. The time deadband sets the length of time that an attribute value must continuously remain in an alarm or unalarmed state. The time deadband filters out rapid, transitory value spikes.

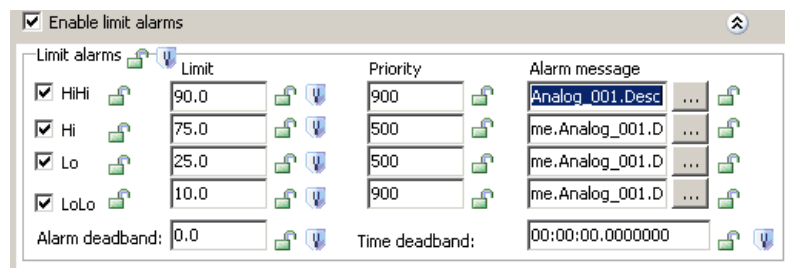
The timestamp when a state alarm becomes active or inactive is the most current timestamp of the corresponding input value. If there is no timestamp associated with the alarmed value, the AppEngine timestamp is used instead.

Limit Alarms

A limit alarm compares the current value to one or more predetermined alarm limits within the attribute's full range of values. If the value exceeds a limit, an alarm occurs.



You can individually select and configure values and priorities for the LoLo, Lo, Hi, and HiHi alarm limits. You can set individual messages for each alarm limit.



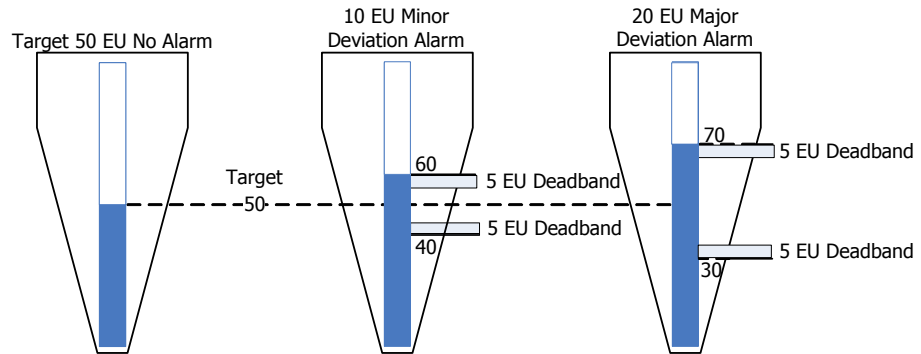
You can also configure alarm and time deadbands for limit alarms. The alarm deadband is expressed as a percentage of the attribute's full value range. The deadband range sets the percentage of the total range that the attribute value must change to reset a limit alarm to the inactive state. For example, if the HiHi alarm limit is 80 and the alarm deadband is 5, then the attribute value must decrease to 74 to reset a HiHi alarm to an inactive state.

The time deadband sets the length of time that an attribute value must continuously remain in an alarm or unalarmed condition. The process variable must remain above or below the indicated limit for at least the indicated deadband time before the application object updates the status of the alarm `CONDITION` Boolean. Then, standard Alarm Primitive logic determines whether to take that updated alarm condition and report changes to the alarm state or not.

The timestamp when a limit alarm becomes active or inactive is the most current timestamp of the corresponding input value. If there is no timestamp associated with the alarmed value, the AppEngine timestamp is used instead.

Target Deviation Alarms

A target deviation alarm compares the current attribute value to a target Engineering Units value. Then, the absolute value of the difference is compared to one or more alarm deviation limits expressed in Engineering Units.



You can individually select and configure values and priorities for the minor deviation limit and the major deviation limit. You can set individual messages for the minor and major deviation alarm limits.

The screenshot shows the configuration window for target deviation alarms. The 'Enable target deviation alarms' checkbox is checked. The configuration includes:

Alarm Type	Tolerance	Priority	Alarm message
Minor	10.0	500	me.Analog_001.D
Major	20.0	900	me.Analog_001.D

Additional settings shown:

- Target: 50.0
- Deviation deadband: 5.0
- Settling period: 00:00:30.0000000

The deviation alarm's settling period is the time allowed for the attribute value to reach an expected target value after a device starts. No alarm can occur during the settling period.

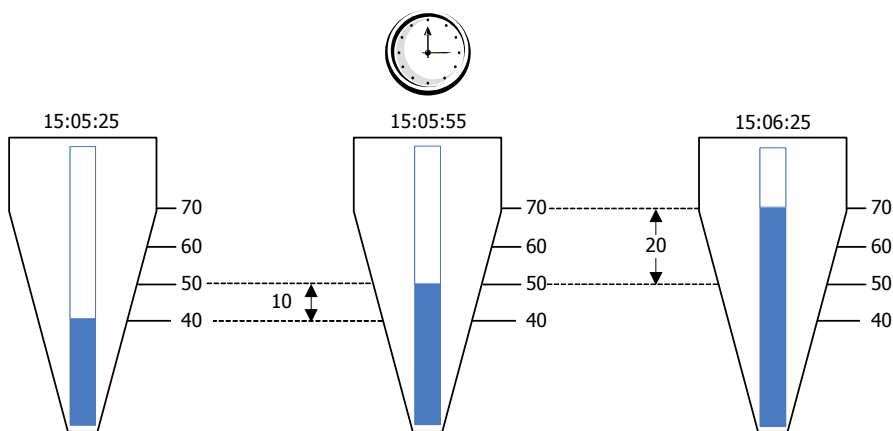
You can also configure a value for a deviation deadband, which is expressed in Engineering Units. The deadband range sets a threshold that an attribute value must change from a deviation limit to reset the alarm to the inactive state.

The timestamp when a deviation alarm becomes active or inactive is the most current timestamp of the corresponding input value. If there is no timestamp associated with the alarmed value, the AppEngine timestamp is used instead.

Rate of Change Alarms

A rate of change alarm identifies when an attribute value is changing too quickly over time. For example, you can set a rate of change alarm for a tank level that indicates when the pump inlet pressure is too high.

Rate of change is the calculated slope, which is the absolute difference between the current and previous attribute values divided by a specified interval. When the slope (positive or negative) exceeds a specified value, a rate of change alarm occurs. For example, if a tank volume increases from 17 to 45 liters over a 5 minute interval, the calculated slope is 5.6 liters per minute. If you set your rate of change alarm limit to 5.0 liters per minute, a rate of change alarm condition exists.



Alarm limits are expressed in the Engineering Units of the attribute's value over an interval, which can be per second, minute, hour, or day.

Direction	Limit	Priority	Alarm message
<input checked="" type="checkbox"/> Up	15.0	500	me.Analog_HIGH.I
<input checked="" type="checkbox"/> Down	15.0	500	me.Analog_LOW.I

Changes per: Sec Evaluate every: 5000 ms

You can select and configure the value and priority for the upward and downward ROC limits. You can set individual messages for ROC alarms that exceed the upward or downward limits.

The timestamp when a rate of change alarm becomes active or inactive is the most current timestamp of the corresponding input value. If there is no timestamp associated with the alarmed value, the AppEngine timestamp is used instead.

Statistical Alarms

A statistical alarm is one in which a statistic is calculated, based upon an attribute. If the statistic exceeds some pre-set limit, the object flags the alarm condition as TRUE.

Setting Alarm State with Object Attributes

The Application Server alarm enable/disable mechanism includes four attributes to set an object alarm mode and report alarm status.

AlarmModeCmd Attribute

AlarmModeCmd is a writable attribute that sets the current commanded alarm mode for the object. You set the AlarmModeCmd to enabled, silenced, or disabled with a script, user input, or from an input extension.

AlarmInhibit Attribute

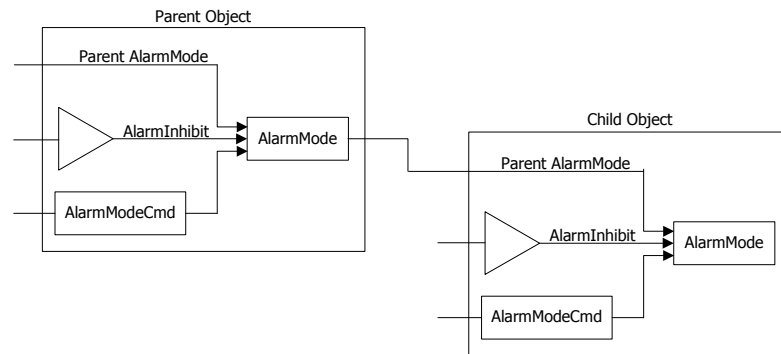
The AlarmInhibit attribute disables one or more alarms when set to TRUE. The value of the AlarmInhibit attribute is typically set by a script, manually by the user, or from an input extension. If the AlarmInhibit attribute is set TRUE, all alarms of the object and of any contained objects are disabled.

When the AlarmInhibit attribute is set to FALSE, alarms are not inhibited and the object AlarmMode and parent object AlarmMode determine whether alarming is enabled, silenced, or disabled.

AlarmMode Attribute

The AlarmMode is a calculated attribute that identifies the object alarm mode and is based upon the current values of an object's:

- AlarmModeCmd attribute
- AlarmInhibit attribute
- Parent object AlarmMode attribute



Application Server checks the AlarmModeCmd and AlarmInhibit attributes of an object and the AlarmMode status of the parent object. Application Server then updates the object's AlarmMode attribute to reflect the most restrictive setting.

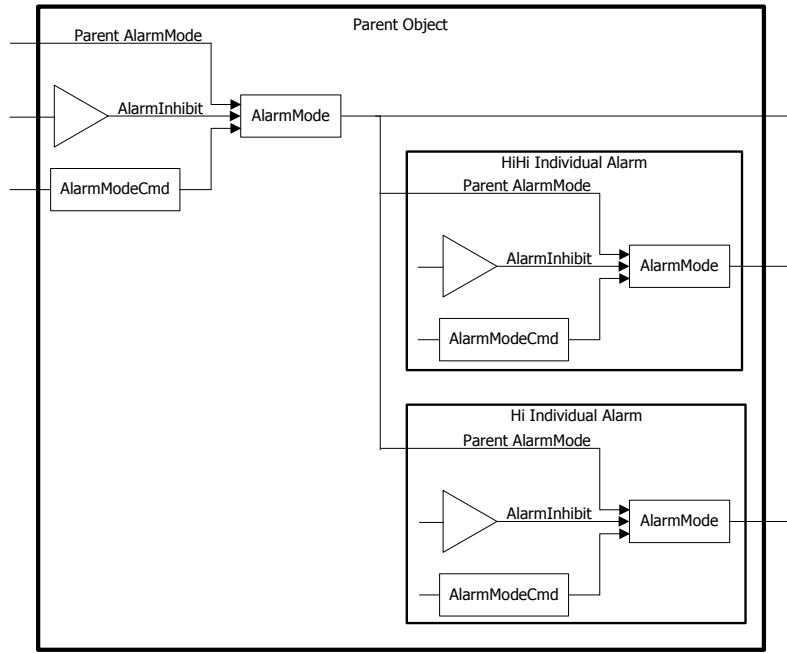
All individual alarms use the object's AlarmMode status to determine whether they are enabled, silenced, or disabled.

AlarmModeEnum Attribute

The AlarmModeEnum attribute can be used by any object to obtain an enumeration of permissible settings for the AlarmModeCmd attribute.

Setting Alarm State for Individual Alarms

You can set individual alarms within an object for each type of alarm. For example, you can set alarms for each of the limits of a level alarm. The following figure shows an object's individual alarms for the HiHi and Hi alarm limits.



The calculated AlarmMode attribute value of an individual alarm uses the same inputs an object alarm. The parent AlarmMode attribute is from the object itself. As with object alarms, the individual alarm mode is set to the most restrictive input state. For example, if the object's AlarmMode state is disabled and the individual alarm's AlarmInhibit is FALSE, the individual alarm is disabled.

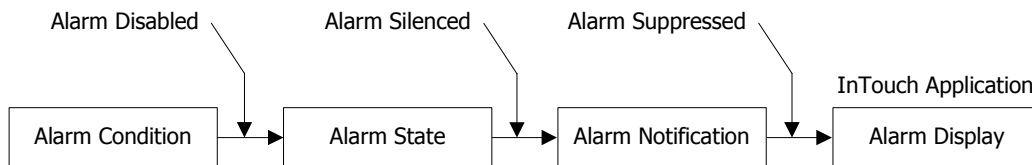
Each individual alarm is autonomous from other individual alarms in an object. The AlarmMode of an individual alarm is not propagated to other alarms. Unlike inhibit for the entire object, inhibit of an individual alarm does not affect the alarms of any contained objects. You can selectively enable, silence, or disable an individual alarm and not set other alarms to the same value within the object hierarchy.

Enabling, Silencing, and Disabling Alarms

Alarms can be enabled, disabled, or silenced while an application is running. Setting an object's alarm state can be set at the Area level, at the container object level, or at the individual object. In addition, individual alarms within a single object can be enabled, silenced, or disabled.

- **Enabled:** All alarms for an object are reported to client applications and saved as historical data. The enabled state is less restrictive than the silenced or disabled alarm states.
- **Silenced:** All alarms for an object are detected, saved to history, and sent to alarm clients. But, alarm clients do not show the alarms. The silenced alarm state is more restricted than the enabled state, but less restrictive than the disabled state.
- **Disabled:** No alarms for the object are detected. The alarm is return-to-normal until the alarm is re-enabled. The disabled state is more restrictive than the silenced and enabled alarm states. A disabled alarm does not require acknowledgement.

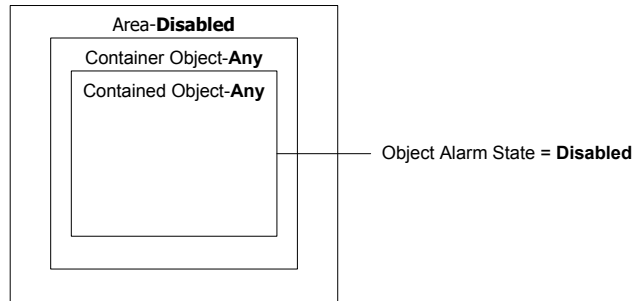
The following figure shows how the different alarm modes affect the different phases of an alarm. In the case of Alarm Suppressed, selected alarms can be filtered out of the InTouch AlarmViewer display by an operator, or programmatically by a script.



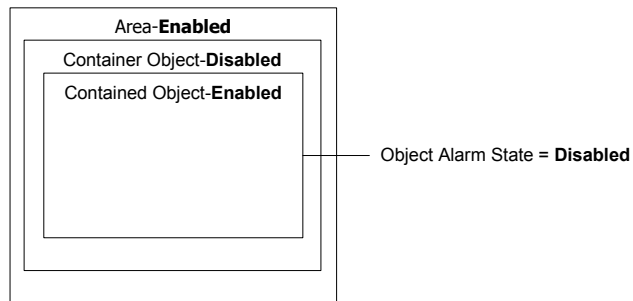
To ensure that alarmed attributes always have current data, the alarm primitive and the alarm extension primitive always register a reference to the alarmed attribute. This guarantees that Message Exchange never suspends updates for this attribute. Even if alarms are disabled for a particular attribute, the alarmed attribute cannot enter an Advanced Communication Management Suspended state.

The object hierarchy and alarm states determine the final alarm condition of an object.

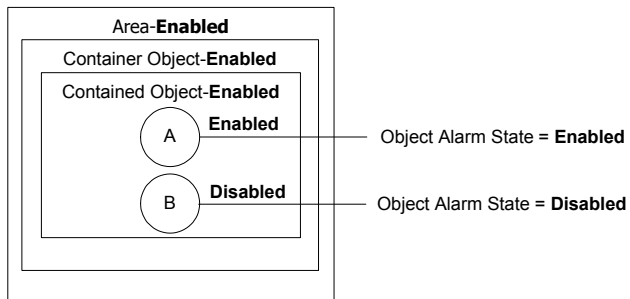
- An Area object's alarm state determines the alarm state for all alarms of objects that belong to the Area.



- The most restrictive setting within an object hierarchy determines the object's alarm state.



- When an individual object alarm is silenced or disabled, it applies only to that alarm and not to other alarms belonging to the object.



The alarms on any contained object are not affected. The disabled or silenced state of an individual alarm does not propagate downward through the object hierarchy to the alarms of any contained or assigned object.

Enabling Alarms

To enable an object's alarms, you must ensure that the `AlarmModeCmd` and `AlarmInhibit` attributes are enabled for the object, its container, and its area. An event, including the user's name, is generated indicating the object's alarms are enabled.

When object alarms are enabled, you can enable, silence, or disable an individual alarm.

Silencing Alarms

When object alarms are silenced, an individual alarm that is enabled or silenced is forced to be silenced. When object alarms are silenced, an individual alarm can be disabled.

Disabling Alarms

When object alarms are disabled, an individual alarm that is enabled or silenced is forced to be disabled.

When object alarms are enabled and an individual alarm is enabled or silenced, the individual alarm can be inhibited. This forces the individual alarm to be disabled.

When object alarms are silenced and an individual alarm is enabled or silenced, the individual alarm can be inhibited. This forces the individual alarm to be disabled.

When object alarms are inhibited, an individual alarm that is enabled or silenced is forced to be disabled.

Throttling Alarms

Alarm throttling prevents network and message queues from flooding during periods of high alarm activity. Throttling sets a maximum number of transitions into and out of an alarm state within a defined period. Alarm transitions that exceed the throttling limit are not reported.

The WinPlatform object includes tuning parameters that specify the maximum alarm rate per second on an engine. The Alarm throttle limit attribute must be used in conjunction with the scan period to determine the maximum number of alarms that can occur in a scan cycle. You can set the alarm throttle limit when you configure the WinPlatform object. For more information about setting the scan period and alarm throttle limit, see *Configuring WinPlatform Object Alarms* on page 196.

Alarm messages can be throttled for alarms going into alarm and out of alarm state. Alarm acknowledgement messages are not throttled. Users can still acknowledge alarms even when alarm throttling is active. Users can still disable alarms for objects or areas when the alarm rate causes throttling to occur. Also, the alarm inhibit and the disable/enable/silence messages are not throttled.

If an active alarm is disabled, the going out of alarm and disabled messages are sent to the notification distributor. If alarms are being throttled, the going out of alarm message can be throttled. The disabled message is accepted regardless of the throttling status. The going out of alarm message is not raised again. This can result in never logging a message for that alarm indicating it went out of alarm. The final alarm list in the Notification Distributor still shows the correct alarm state.

Propagating Timestamps with Alarms and Events

An alarm extension primitive always registers a reference to the alarmed attribute. This registered reference guarantees that Message Exchange never suspends updates for the alarmed attribute. Even if alarms are disabled for a particular attribute, the Advanced Communication Management feature cannot suspend the attribute.

Be aware that the time stamp propagates to all masked bits of an integer attribute, even if only one of the bits changes.

For example, you have an Integer address in a PLC that represents 16 different alarm states. You assign ObjA.UDA_Integer to point to the PLC address. You then split the bits to different alarm attributes, adding a field attribute for each alarm and naming them ObjA.FA_Alarm00 to ObjA.FA_Alarm15. Each field attribute has an input source that refers to a different bit of ObjA.UDA_Integer. For example, ObjA.FA_00.InputSource -> me.UDA_Int.00, and so on. At run time, when bit 00 is changing in the PLC, all of the field attributes (ObjA.FA_Alarm00 to ObjA.FA_Alarm15) get a new time stamp, as all the bits changed. This can result in incorrect time stamps for alarms.

Alarms or Events Become Active

For alarms or events, if an attribute has no timestamp, the current AppEngine scan time is used instead. If an attribute has a timestamp, the timestamp of the value that caused the alarm to occur is used as the value for the object's AlarmOnTime attribute.

When an alarm is silenced or disabled, and an alarm condition becomes TRUE, when the alarm is later enabled, the timestamp for AlarmOnTime is the timestamp of the most recent attribute change, not the time at which the alarm was enabled.

For an event, if an attribute has a timestamp, the timestamp of the value that caused the event to be reported is used for the EventTime. A System Event timestamp is the current system time.

Alarms Become Disabled

If an active alarm becomes disabled, the alarm is forced to return to normal. The timestamp corresponds to the current time when the alarm became disabled, not the timestamp of the attribute nor of the alarm condition. Otherwise, the assigned timestamp is the AppEngine scan time.

Alarms Revert to Normal

If an attribute value has a timestamp, the timestamp of the value that caused the alarm to revert to normal is used for the AlarmOffTime.

If alarm is based upon several attributes or upon several values of a single attribute, the most recent timestamp is used when assigning values to the AlarmOnTime or AlarmOffTime.

Alarm Acknowledgement

Alarms are acknowledged by users who view unacknowledged alarms from their client applications like InTouch HMI. Only alarms of certain priority levels need to be acknowledged.

The basic workflow to acknowledge an alarm consists of the following general steps:

- The alarm is detected and reported to any subscribed alarm clients. The alarm is unacknowledged unless it is of a priority level that does not require acknowledgement.
- An authorized user attempts to acknowledge the alarm from the client. The user can type an optional comment when acknowledging the alarm.
- The user's acknowledge request is sent to the detecting object's alarm primitive. The acknowledgment must pass through standard security checks first. The acknowledge request contains the user's name and any alarm comment.

For the alarm utility and alarm extension primitive, the alarm is acknowledged within the alarm primitive immediately. The user name, comment, and acknowledged time are also saved. Alarm comments can be localized into any supported language. See *Working with Languages* on page 247.

The acknowledge is considered an alarm state change, which is sent to all subscribed clients. When an alarm is acknowledged, the current AppEngine timestamp is used as the acknowledgement time.

After an alarm is acknowledged for the first time, any additional (extra) acknowledgement attempts for the same alarm are rejected and an error is returned.

Configuring Alarms

Alarming capabilities are a part of object templates, but they are not implemented until you configure the object in the IDE. After alarms are configured, you can view Application Server alarms in a client visualization application like InTouch HMI.

Configuring an object to be an alarm provider includes the following general sequence of steps:

- Decide whether alarm notification is needed for each possible alarm condition of an object. For example, a command time-out alarm for a valve if the output command fails to move the valve.
- Edit the object and set an attribute that specifies alarming.
- Edit the object from the IDE and assign values to alarm attributes.
- Configure the alarm properties. Typically, the fields that require configuration are Category, Priority and Description.
- Configuring any limit fields to set an alarm. For example, the feedback time-out time limit.

You can add alarm detection and reporting capabilities to objects that were not originally developed to detect alarms. You do this by setting alarm extensions for the object.

Configuring Alarming for System Objects

To enable alarming for your application, you need to configure your Galaxy's WinPlatform and AppEngine objects as alarm providers. Both system objects report their own alarms.

Client applications subscribe to application object alarms by the area containing the objects. Client applications can also subscribe to WinPlatforms and AppEngines directly. These are called "pseudo-areas." They do not need to be assigned to an area for a client to see the alarms, although the user may want to assign them to an area, such as for simplifying the alarm query in the InTouch Alarm Viewer or Alarm DB Logger.

The following list shows the alarms for each system object.

- WinPlatform
 - Excess CPU load alarm
 - Low disk space alarm
 - Excessive page faults alarm
 - Low memory alarm
 - Engine failure alarm
 - Engine checkpoint failure alarm
 - Object quarantined condition
 - Subscription folding condition
 - Scheduler scan overrun condition

A WinPlatform object includes a general communication alarm when it loses contact with the areas to which it is subscribed.

- AppEngine
 - Checkpoint failure alarm
 - Object quarantined condition
 - Subscription folding condition
 - Scheduler scan overrun condition
 - Redundancy failover alarm
 - Redundancy Standby unavailable
 - Redundancy Standby not ready
- ViewEngine
 - Checkpoint failure alarm
 - Object quarantined condition
 - Subscription folding condition
 - Scheduler scan overrun condition

The Area and InTouchViewApp system objects do not include any alarms.

Configuring WinPlatform Object Alarms

You select the areas of your Galaxy to monitor for alarms from the WinPlatform object. Also, you can select specific alarms for the status of the WinPlatform itself.

You must specify that the WinPlatform object is an InTouch alarm provider to subscribe to alarms from the various areas (and pseudo-areas) of the Galaxy and report them to the InTouch Alarm Manager.

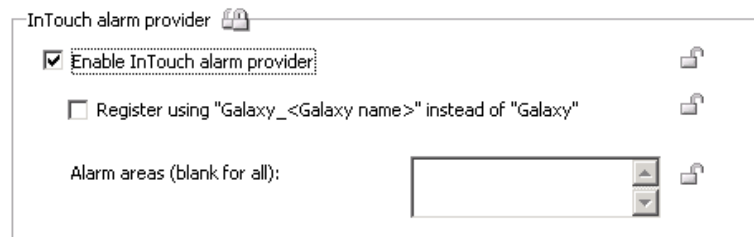
Selecting the Register using “Galaxy_<GalaxyName>” instead of “Galaxy” option:

When you select the Register using “Galaxy_<Galaxy name>” instead of “Galaxy” option, you enable ITAlarmProvider contained in WinPlatform to register the ITAlarmSubsystem using a provider name of Galaxy_GalaxyName.

- This means that all IT applications which query alarms from this platform should be modified to use Galaxy_GalaxyName!AreaName.
- Select this option when the ArcestrA Alarm Control, the embedded alarm client or EAC, or the AlarmViewControl in InTouch queries alarms from different galaxies.

To configure a WinPlatform object to be an alarm provider

- 1 Open the WinPlatform object in the Object Editor.
- 2 Click the **General** tab.



- 3 Select the **InTouch alarm provider** check box.
- 4 Select the **Register using “Galaxy_<GalaxyName>” instead of “Galaxy”** check box to enable Galaxy_<GalaxyName> registration for alarm comment language switching. An information box appears. Click **OK** on the information box to continue. See Working with Languages on page 247.

- In the **Alarm Areas** box, type the names of areas to subscribe to for alarms.

If you leave the **Alarm Areas** box blank, the WinPlatform subscribes to all areas in the Galaxy.

If you want to subscribe to only selected areas within the Galaxy, insert a space between each area name. For example:

Area1 Area2 Area3

- Click the **Engine** tab to show the **Alarm throttle limit** box. Either accept the default throttle limit of 2000 alarms per second, or enter another value. A value of 0 disables alarm throttling. For more information about alarm throttling, see [Throttling Alarms](#) on page 191
- Click the **Alarms** tab to show platform, engine, and scheduler alarms that can be set for the WinPlatform object.

The screenshot shows the configuration window for 'TankFarm8WinPlatform *'. The 'Alarms' tab is selected, displaying the following settings:

- Platform**
 - Report excess CPU load alarm
 - Alarm limit: 90.0
 - Value deadband: 10.0
 - Priority: 800
 - Report low disk space alarm
 - Alarm limit: 750.0 MB
 - Priority: 800
 - Report excessive page faults alarm
 - Alarm limit: 5.0
 - Value deadband: 1.0
 - Priority: 800
 - Report low memory alarm
 - Alarm limit: 250.0 MB
 - Value deadband: 50.0
 - Priority: 900
 - Report engine failure alarm
 - Priority: 900
- Engine**
 - Report checkpoint failure alarm
 - Priority: 500
 - Report object quarantined condition
 - Priority: 500
 - Report subscription folding condition
 - Priority:
- Scheduler**
 - Report scan overrun condition
 - Consecutive scan overrun limit: -1
 - Priority: 500

- 8 Select the checkbox next to each alarm that you want to enable for the WinPlatform object.
- 9 Set the limit, value deadband, and priority for each alarm you selected.
- 10 Save and close the Object Editor.
- 11 Check the object in to the Galaxy.
- 12 Deploy the object in an on scan state.

Configuring Alarms for an AppEngine Object

You can set AppEngine attributes that determine whether alarms are enabled for an engine, scheduler, and redundant failover engine.

To configure AppEngine object alarms

- 1 Open the AppEngine object in the Object Editor.
- 2 Click the **Alarms** tab to show engine, scheduler, and redundancy alarms that can be set for the AppEngine object.

The screenshot shows the 'Alarms' tab for the 'TankFarmAppEngine' object. The interface is organized into three main sections:

- Engine:** Contains three checkboxes for enabling alarms: 'Report checkpoint failure alarm', 'Report object quarantined condition', and 'Report subscription folding condition'. Each checkbox has a 'Priority' input field and a lock icon to its right.
- Scheduler:** Contains one checkbox for 'Report scan overrun condition'. Below it is a 'Consecutive scan overrun limit' input field (set to -1) and a 'Priority' input field, both with lock icons.
- Redundancy:** Contains three checkboxes for enabling alarms: 'Report alarm when a failover occurs', 'Report alarm when Standby becomes unavailable', and 'Report alarm when Standby becomes not ready'. Each checkbox has a 'Priority' input field.

- 3 Select the checkbox next to each alarm that you want to enable for the AppEngine object.
- 4 Set the priority for each alarm you selected.
- 5 Save and close the Object Editor.

- 6 Check the object in to the Galaxy.
- 7 Deploy the object in an on scan state.

Configuring Alarms and Events for Application Objects

The following table shows the different types of alarms that can be specified for application objects. The table shows the application objects containing native alarm attributes.

You can also set alarms for an object's extendable attributes. For more information about setting alarms for extendable attributes, see [Setting Alarms on the Extension Page](#) on page 203.

Application Objects	Alarm Types				
	State	Limit	Target Deviation	Rate of Change	Statistical
AnalogDevice	●	●	●	●	
Boolean					
DiscreteDevice	●				●
Double					
FieldReference					
Float					
Integer					
Sequencer	●				
String					
Switch	●				
UserDefined	●	●	●	●	

The following list shows the types of alarms for each application object in more detail.

- AnalogDevice
 - Level alarms (HiHi, Hi, Lo, LoLo) [limit alarms]
 - Rate of Change alarms (Up, Down)
 - Target Deviation alarms (Minor, Major)
 - PV Bad Value alarm [state alarm]

- DiscreteDevice
 - Uncommanded change alarm [state alarm]
 - Command time-out alarm [state alarm]
 - Active1 state alarm [state alarm]
 - Active2 state alarm [state alarm]
 - Fault state alarm [state alarm]
 - Active1 state duration alarm [statistical alarm]
 - Active2 state duration alarm [statistical alarm]
- Sequencer
 - Execution halted [state alarm]
 - Condition trigger failure [state alarm]
 - OnEntry output failure [state alarm]
 - OnExit output failure [state alarm]
- Switch
 - PV State alarm [state alarm]
- UserDefined (Field Attributes can be alarmed)
 - Discrete field attribute
 - PV State alarm [state alarm]
 - PV Bad Value alarm (that is, bad quality) [state alarm]
 - Analog field attribute
 - Limit alarms (HiHi, Hi, Lo, LoLo) [limit alarms]
 - Rate of Change alarms (Up, Down)
 - Target Deviation alarms (Minor, Major)
 - PV Bad Value alarm (that is, bad quality) [state alarm]

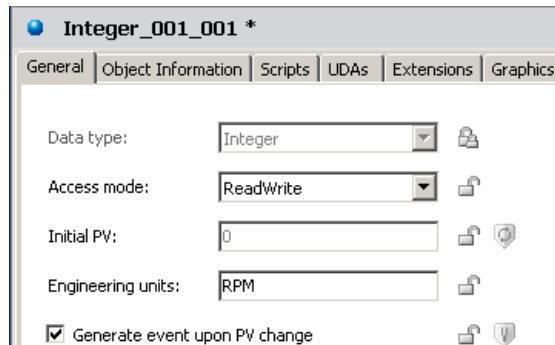
There are no built-in alarms for these application objects:

- FieldReference
- Boolean
- Double
- Float
- Integer
- String

You can also configure your application objects to generate an event each time the object's PV value changes. In addition, you can configure an alarm extension on any object for any Boolean attribute.

To configure alarming and events for application objects

- 1 Open the application object with the Object Editor.
- 2 Click the **General** tab to show the **Generate event upon PV change** checkbox.



An AnalogDevice object does not include this checkbox. The checkbox is located on the **Field Attributes** page for a UserDefined object.

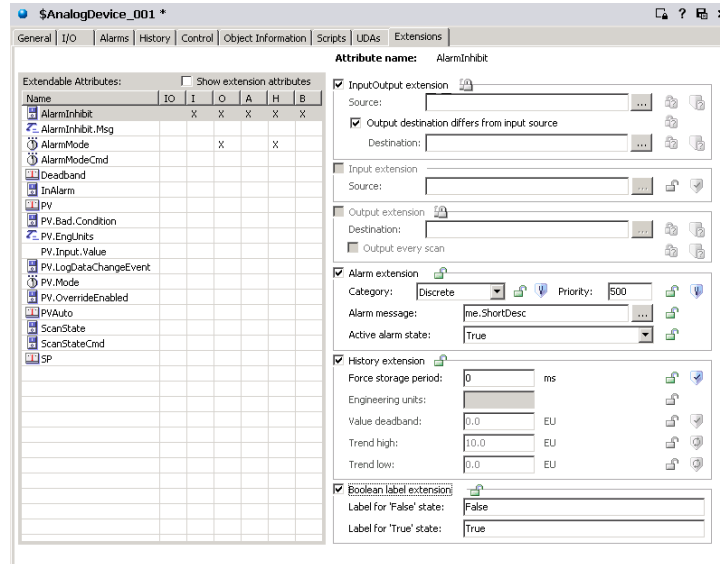
- 3 Select or clear the checkbox based on whether you want to generate an event each time the object's PV value changes.
- 4 Click the tab that lists alarm attributes.
 - For the AnalogDevice object, click **Alarms**.
 - For the DiscreteDevice object, click **Alarms**.
 - For the Sequencer object, click **Settings**.
 - For the Switch object, click **General**.
 - For the UserDefined object, click **Field Attributes**.
- 5 Select the checkbox that enables alarming for the object.
- 6 Assign values to the attributes for the type of alarm you selected by completing the following steps:
 - a Assign values to the alarm limits based on the type of alarm.
 - b Assign an alarm priority (1-999) for each limit you set.
 - c Accept the default alarm message or include another message for each alarm limit.
 - d Assign values to the remaining attributes based on the type of alarm you selected. For more information about other alarm attributes, see Types of Alarms on page 181.
- 7 Save your object changes and close the Object Editor.

Setting Alarms on the Extension Page

You set alarms for all Application Server objects in the Extensions page in the Object Editor.

To specify alarms for an object extension

- 1 On the **Extensions** page of the Object Editor, select an attribute from the **Extendable Attributes List**. The four extension groups dynamically change to allowed extension rules for the selected attribute type.



- 2 Select the **Alarms** check box. For **Alarm Extension**, select a **Category** from the list:

Batch	DeviationMajor	DeviationMinor
Discrete	Process	ROC Hi
ROC Lo	Software	SPC
System	Value Hi	Value HiHi
Value Lo	Value LoLo	

- Type a **Priority** level for the alarm (default is 500).
 - Select to use either the **Object Description for Alarm Message** or type another alarm message in the **Message** box. An **X** appears in the **A** column of the selected attribute.
- 3 For **Boolean Label Extension**, specify text strings for the **False** state and the **True** state, if needed. These text strings appear in the **Active Alarm State** list for you to select.
 - 4 Lock the values, if needed. The lock symbol is available only when you are extending a template. Otherwise, it indicates the lock condition of the value in the parent object.

Distributing Alarms and Events

After you configure object instances for alarm detection, deploy the instances and put them On scan. The instances begin checking for alarm conditions.

When an alarm is detected, or an event occurs, a notification is reported to its alarm and event distributor, which is running on the same AppEngine.

These alarm and event distributors include:

- Area objects Area objects report detected alarms through the Area, which distributes them to alarm and event clients.
- WinPlatform objects Report their own alarms and events.
- AppEngine objects Report their own alarms and events.
- Device IntegrationObjects Report their own alarms and events.

The Area object plays a key role in alarm and event distribution. All objects belong to an Area. Areas can contain sub-Areas. Alarm and event clients are configured to subscribe to a set of Areas.

Areas provide a key organizational role in grouping alarm information and assigning it to users who use alarm and event clients to monitor their Areas of responsibility.

WinPlatforms, AppEngines and Device Integration objects do not report their alarms and events to Area objects even though they belong to Areas. This allows alarm clients to receive alarm notifications without any dependencies on Area objects. For example, a deployed and running WinPlatform can report alarms even though its Area is not deployed and running.

Alarm-event distributor objects maintain a list of all currently active alarms and inactive but unacknowledged alarms. They do not maintain a list of events, which are routed to clients that are currently subscribed at the time of the event.

You can configure a WinPlatform to act as an InTouch Alarm Provider in the run-time environment.

The WinPlatform sends an alarm through the InTouch Distributed Alarm System to InTouch clients when the WinPlatform loses communication with an Area that it subscribes to. This condition typically occurs during a network outage with computers hosting those Areas.

In a network outage, the WinPlatform InTouch Alarm Provider sends an alarm for each disconnected Area that it subscribes to, including all of its alarm distribution hierarchy. Each of these alarms is a high priority alarm that contains the name of the Area to which communication is lost. These communication problem alarms must be acknowledged.

Although they still appear in the historical record, any current alarms from the disconnected Area drop from the InTouch client's summary list. They can no longer be acknowledged.

When communication to the disconnected Areas is restored, any unacknowledged alarms generated in those Areas are sent to the alarm client.

Subscribing to Alarms and Events from a Client

Clients indicate interest in alarms and events by subscribing to an Area. When subscribing to an Area, the subscription is actually to all notification distributors within that Area.

For example, if an Area contains sub-Areas, those sub-Areas are subscribed to. If WinPlatforms, AppEngines or Device Integration objects belong to an Area, those objects are also directly subscribed to.

When a notification distributor receives an alarm and event subscription from a client, the notification distributor provides the client with the following:

- A list of all current alarm conditions, including unacknowledged return-to-normal conditions.
- An alarm condition state change. A state change includes transitions into or out of alarm (return to normal) and change in acknowledged flag.
- An event occurrence.

Alarm and event subscription requests do not include filters, for example, only show alarms greater than a specific priority value. All alarm and event messages received by the notification distributor are sent to all subscribed clients. Filtering is provided as a display option by clients.

Using InTouch HMI as the Alarm and Event Client

InTouch run-time clients subscribe to event reports from a Galaxy. Application Server reports alarms to the InTouch Distributed Alarm System, which subscribe to alarm and event reports from a Galaxy.

An InTouch client application can visualize Application Server components. An InTouch alarm client can show alarm information for new, unacknowledged alarms, including all required fields.

The new alarm is in the unacknowledged state. An operator can view alarms, acknowledge alarms, disable alarms, and enable alarms from the client application running in InTouch WindowViewer.

Understanding the Syntax of Alarm Queries

InTouch alarm queries subscribe to alarm and event information from objects within a Galaxy. The alarm and event queries can be in the form of user input or a script.

Alarm query syntax must be in one of the following forms:

```
\Provider!Area
```

or

```
\\Node\Provider!Area
```

You can have one or more references in a query separated by spaces.

You can also optionally append a tagname filter at the end, separated by another exclamation mark:

```
\Provider!Area!Filter
```

```
\\Node\Provider!Area!Filter
```

The filter can have a wildcard * character at the beginning or at the end, but not both.

The \\Node at the beginning is only important if you want to query for alarms from a provider on another computer.

Otherwise, you can leave it off and the reference is assumed to be a provider on the local computer. The provider name Galaxy refers to alarms and events that get reported by the WinPlatform configured as an InTouch alarm provider on that computer node.

Alarm Query Syntax when Register Using Galaxy_<GalaxyName> is Enabled

In the WinPlatform object, when you enable InTouch alarm provider, you can enable **Register using Galaxy_<GalaxyName> instead of Galaxy**. This option will register the platform to the

alarm subsystem using the Galaxy name preferred by “Galaxy_” instead of just the word “Galaxy”. This allows an InTouch application to monitor alarms from multiple Galaxies and avoid name conflicts.

Syntax changes slightly when Galaxy_GalaxyName is enabled:

- Use \\ for machine name.
- Use \ for Galaxy or Galaxy_<GalaxyName>.
- Use ! for Area.

For example: \\Galaxy\MyGalaxy!Area001.

If Galaxy_GalaxyName is not enabled in WinPlatform, then the default behavior described in the previous section applies.

You can determine if Galaxy_<GalaxyName> has been enabled by monitoring the run-time attribute of the platform `ITAlarmProvider.ProviderNameAsGalaxyNameEnabled`.

Examples of Alarm Queries

- You can submit a query to get all alarms from Area1 and all other alarms within Area1, as reported by the WinPlatform object on the local computer.

```
\Galaxy!Area1
```

The query returns all alarms and events from all objects directly contained in Area1 and any sub-areas contained by Area1. This hierarchy is determined by what is configured in the Model View in the IDE.

- If Area1 and Area2 are two separate mutually exclusive areas, you can submit a query for alarms from both areas.

```
\Galaxy!Area1 \Galaxy!Area2
```

- If you're on NodeA and the WinPlatform is on NodeB, you can submit a query for the alarms from the remote computer.

```
\\NodeB\Galaxy!Area1
```

- You can submit a query for all alarms from objects whose name begins with "Tank" in the TankFarm1 area.

```
\Galaxy!TankFarm1!Tank*
```

The trailing wildcard character matches alarms from all objects with names that begin with “Tank” like Tank001, Tank002, TankUpper, or TankLower.

- You can submit a query for specific alarm types. For example, you can submit a query for all HiHi alarms in the TankFarm1 area.

```
\Galaxy!TankFarm1!*HiHi
```

- You can submit a query for all types of alarms from a specific object within an area.

```
\Galaxy!TankFarm1!Tank752.*
```

The trailing wildcard character matches all alarm types for Tank752.

Alarm Requirements for InTouch Client Applications

For Application Server alarming to function, the following conditions must be met:

In Application Server:

- One or more Area objects are deployed and running.
- The source object is on scan.
- The source object's Area is on scan.
- Alarming must be enabled for the target object.
- An InTouch alarm provider on any WinPlatform in the Galaxy.

In InTouch:

- The InTouch client application is running in WindowViewer.
- An InTouch alarm ActiveX control is placed in a window and configured as an alarm consumer for the Galaxy.
- The user is logged into InTouch using ArcestrA security and is authorized to acknowledge alarms for the object that is in the alarm state. If the user only wants to view alarms, security authorization is not required.

Application Server validates the user has sufficient security privileges to acknowledge the alarm.

If the user does not have privileges to acknowledge alarms, the user can attempt to acknowledge the alarm, but the Galaxy rejects the acknowledgment request. The alarm remains unacknowledged in the InTouch Alarm display.

The rejected alarm acknowledge event is recorded in InTouch Event History if the user attempting the acknowledgement has a valid Galaxy user account. Otherwise, the rejected acknowledgement is not recorded as an event.

Alarms and Events in the InTouch HMI and in Application Server

Both InTouch HMI and Application Server implement alarms and events. The table below shows the similarities and differences between the two products.

Item	InTouch	Application Server
Alarm configured or detected by	Within a tag	Within an object
Alarm Classes (client column)	Only certain classes of alarms are supported or detected: DSC, VALUE, DEV, ROC, SPC.	No system-wide distinction for classes. Alarms are tied to a Boolean that can be triggered from any logic.
Alarm Type (Sub-class) (client column)	Discrete, LoLo, Lo, Hi, HiHi, MinorDev, MajorDev, ROC, SPC. Client column.	No sub-class. The Alarm Primitive name is the closest concept. For example, ".PVHiAlarm". Mapped from Category.
Priority (client column)	1-999 (1 most urgent)	Priority 0-999. 0 most urgent. 0 is mapped to 1 in InTouch.
Name (client column)	Alarm name = Tag name.	Object.attribute
Comment (client comment)	Separate alarm comment, which is different from the tag comment.	Object short description or alarm message where available.
Group	Alarm group allows client-side filtering. Sub-groups must be on same InTouch.	No alarm group. But Area provides mappable concept. Sub-Areas can be on different nodes.

Item	InTouch	Application Server
State	<p>Four 4 states, which are combinations of ACK/UNACK and ALARM/RTN</p> <ul style="list-style-type: none"> • UNACK/ALARM (usually displayed as UNACK) • ACK/ALARM (usually displayed as ACK) • UNACK/RTN (usually displayed as UNACK_RTN) • ACK/RTN (usually displayed as ACK_RTN) <p>These states have a 1:1 correspondence with states of the Alarm Primitive, which keeps track of whether the alarm is InAlarm and IsAked.</p> <p>Alarms in the state ACK/RTN are not shown in the SUMMARY alarm display because they do not need any further action from the operator. But, all four states appear in the HISTORY display, and in the Alarm Database.</p>	<p>Alarm state provides equivalent concept and can be mapped.</p>
Value, CheckValue	<p>Only static values sent with alarm message.</p>	<p>Static values and dynamic references are provided.</p>
Ack	<p>All alarms sent to client and require acknowledgement regardless of priority.</p>	<p>All alarms sent to client and require acknowledgement regardless of priority.</p>
History	<p>Alarm state changes are logged to event history and shown on historical client.</p>	<p>Alarm state changes are logged to event history and shown on historical client.</p>

Chapter 9

Working with References

References allow identification and communication between objects in the ArcestrA environment. Every object, every attribute, and every property can be uniquely referenced. Those references are communicated over the messaging system. The Message Exchange is the object-to-object communications protocol used by ArcestrA and the Wonderware Application Server.

Note ArcestrA is the framework for supervisory control and manufacturing information systems. It is an open and extensible technology based on a distributed, object-based design. For example, if you are using Application Server with InTouch, these products communicate with each other using the ArcestrA framework.

This section describes the concept of references and how to use reference strings in creating your Application Server application.

Using Message Exchange and Attributes

All object attributes have properties, such as Value and Quality. Any data read or written to or from these attributes over ArchestrA Message Exchange is tracked. If an operation cannot be performed, the requesting client is notified.

Message Exchange provides the following features and information:

- Guaranteed response
- Name signatures
- Status and data quality
- Message order preservation within a priority system
- AppEngine-to-AppEngine buffering
- Publish-subscribe heartbeats

Reference Strings

Reference strings refer to an object or to data within an object's attributes. A reference string consists of an object's reference string plus an attribute's reference string.

`object Reference + Attribute Reference`

A reference string is the object name plus the attribute name: `ObjectName.AttributeName`.

In `TIC101.PV`, `TIC101` is the object reference and `PV` is the attribute reference. The `AttributeName` can be omitted in a reference string, `PV` being assumed in such cases.

Note Some objects have a `PV` attribute, while others do not.

Reference strings are concatenated substrings, each no more than 32 characters separated by periods. A substring cannot contain a period. Mathematical operator characters are not allowed. At least one character in each substring must be non-numeric.

Avoid assigning objects and attributes names such that the same reference string can refer to two different things. For example, you have two objects named A1 and B2, and inside A1 you create a UDA Float named B2. A1.B2 refers to the UDA Float named B2. If you then assign object B2 so that A1 is a container of object B2, the reference A1.B2 could refer either to the object B2 or the UDA B2 Float.

Important The Galaxy resolves reference strings. If the GR is not available, resolution is done on a peer-to-peer level. After initial resolution, an object is provided an alias that handles references to its location across your network. If an object is relocated or renamed, the reference string resolution is repeated and a new alias provided.

Relative References

References that go up the hierarchy to parent objects are called relative references. For more information, see [ApplicationObject Containment](#) on page 55.

Relative references, such as `Me`, are valid reference strings. A valid reference string must always contain at least a relative reference or one substring.

The following are valid relative references that refer to the current object:

- `Me`
- `MyContainer`
- `MyArea`
- `MyPlatform`
- `MyEngine`.

Relative references are especially useful in templates because absolute references typically do not apply or make sense.

When you use relative references, like `MyContainer`, you can refer to contained objects within that container. For example, a reference to `MyContainer.InletValve.PV` is equivalent to `Tank1.InletValve.PV` in the following hierarchy:

<code>Tank1</code>	Cannot reference at this level because this is not contained
<code>Inlet Valve (InletValve)</code>	Can reference at this level because this object is contained
<code>Outlet Valve (OutletValve)</code>	Can reference at this level because this object is contained

Property References

Certain property names are reserved for ArcestrA. If a string has a reserved property name in the ArcestrA environment, you can still use it. The `PROPERTY` keyword must be part of the string, for example, `PROPERTY (propertyName) .` In all other cases, the case insensitive `PROPERTY` keyword is not required.

The `Value` property is assumed if no property reference is specified.

The following are property references:

- `.Name`
- `.Value`
- `.Type`
- `.Quality`
- `.Time`

syntax:

```
obj.int.PROPERTY (quality)
```

where:

`obj` = object specifier

`int` = attribute

`PROPERTY` = keyword

`(quality)` = property specifier

For example, you can address the time of an attribute in a scan group for a `DIOBJECT` from within an `InTouch` application, as follows:

```
Galaxy:"<DIOBJECT>.<scangroup>.<attribute>.Property (Time) "
```

This is the same as if you used `.Time`:

```
Galaxy:"<DIOBJECT>.<scangroup>.Attribute (<attribute>) .Time"
```

You can directly address an item without having an attribute in the scan group. For this example, the item is `MB1`:

```
Galaxy:"<DIOBJECT>.<scangroup>.MB1.Property (Time) "
```

and

```
Galaxy:"<DIOBJECT>.<scangroup>.Attribute (MB1) .Time"
```


For objects with a default scan group, you must refer to the `.Time`, `.Value`, and `.Quality` properties using the `.Property(time)`, `.Property(value)`, and `.Property(quality)` notation.

Handling Time Zones with the Time Property

If you need to share time stamp values across different time zones (Platforms), use the `Time` data type in every time zone location. However, if you need to share it as string, remember that when converting the `Time` data type to a string (for example, in a script), it is automatically converted to local time, so you lose the ability to adjust it in a different time zone.

For example, to convert the `Time` property to a string GMT:

```
Dim localDateTime As System.DateTime;
localDateTime = System.DateTime.Parse( obj.attr.Time );
obj.udStringGMTfromLocalTime=
    localDateTime.ToUniversalTime().ToString();
```

To convert the string GMT to a string of local time:

```
Dim univDateTime As System.DateTime;
univDateTime = System.DateTime.Parse(
    obj.udStringGMTfromLocalTime );
Obj.udStringLocalTimeFromGMT =
    univDateTime.ToLocalTime().ToString();
```

Preserving Time Stamps from the Publishing Source

In the following cases, if you want to pass only the time stamp, the subscriber gets the time stamp as converted to the local time zone of the publisher and not the time zone of the data source.

Examples of configurations that do not preserve the original time zone are as follows.

In this configuration, `GalaxyB:Object1.TimeAttr` shows the time adjusted to the local time zone of the `GalaxyA FSGateway` and not the time zone of the `PLC`:

```
PLC.Item <= GalaxyA Object1.IntAttr.Time <=
    FSGateway <= GalaxyB OPCClient <= Object1.TimeAttr
```

In this configuration, `GalaxyB:Object1.TimeAttr` shows the time adjusted to the local time zone of the `InTouch` application and not the time zone of the `PLC`:

```
PLC.Item <= GalaxyA Object1.IntAttr.Time <= InTouch
App I/O Message Tag <= GalaxyB InTouchProxy <=
    Object1.TimeAttr
```

To avoid these problems, subscribe to the `GalaxyA:Object1.IntAttr` value property. This way, both the value and time stamp propagate to `GalaxyB:Object1.IntAttr`. You can then use the `GalaxyB:Object1.IntAttr.Time`. For example:

```
PLC.Item <= GalaxyA Object1.IntAttr <= FSGateway <=
GalaxyB OPCClient <= Object1.IntAttr
PLC.Item <= GalaxyA Object1.IntAttr <= InTouch App
I/O Integer Tag <= GalaxyB InTouchProxy <=
Object1.IntAttr
```

In this configuration, the time property propagates from `InTouch` to `Object.IntAttr.Time`:

```
PLC.Item <= InTouch I/O Integer Tag <= Galaxy
InTouchProxy <= Object.IntAttr
```

Arrays

A reference string can also refer to the `Value` property of an array attribute with an optional `Array Element Reference` that includes up to one dimension:

- `[i]` – individual element
- `[]` – entire array

The letter `i` represents an integer constant.

Formatting Reference Strings

These symbols apply to the reference strings that follow:

This...	means...
::=	can be replaced by
	or
[]	contents optional
{}	contents can be left out, used one time or repeated

Quotation marks are not allowed in tag names, primitive names, or attribute names.

Using Literals

Items outside of angle brackets “<>” are literals. For example:

- `reference_string ::= <Automation_object_reference><attribute_reference> | <tag_name>`
- `Automation_object_reference ::= <absolute_reference>|<relative_reference>`
- `absolute_reference ::= <tag_name>{.<contained_name>}`
- `tag_name ::= <identifier>`
- `contained_name ::= <identifier>`
- `relative_reference ::= <relative_name> | <relative_contained_reference>`
- `relative_contained_reference ::= MyContainer.<contained_name> | MyArea.<contained_name>`
- `relative_name ::= Me | MyContainer | MyArea | MyHost | MyEngine | MyPlatform`
- `attribute_reference ::= <value_ref>|<property_ref>`
- `whole_attribute_ref ::= [.<primitive>][.<attribute>] | [.<primitive>][.ATTRIBUTE(attribute)]`
- `value_ref ::= <whole_attribute_ref>[<array_index>]`

- `array_index ::= <open_bracket> {<index>} <close_bracket>`
`[<open_bracket><index><close_bracket>][<open_bracket><index><close_bracket>]`
- `property_ref ::= <whole_attribute_ref>.<property>`
- `property ::= Value|Type|Quality|BitField|Dimension1|SecurityClassification|Locked|Category | propertyref`
- `propertyref ::= PROPERTY(Value|Type|Quality|BitField|Dimension1|SecurityClassification|Locked|Category)`
- `BitField ::= .00, .01, .02, ..., .31 (valid ONLY for attributes of type MxInteger; otherwise Configuration error occurs at run time)`
- `attribute ::= <static_attribute>|<dynamic_attribute>`
- `static_attribute ::= [<static_attribute>.]<identifier>`
- `<dynamic_attribute> ::= <any_char_but>{<any_char_but>}`
- `primitive ::= [<primitive>.]<identifier>`
- `identifier ::= <valid_char>{<valid_char>}`
- `valid_char ::= <letter>|<digit>|<special_character>`
- `letter ::= any letter in alphabet of any language`
- `digit ::= any numerical character`
- `special_character ::= any graphics char, except the following:`
`. + - * / \ = () ` ~ ! % ^ & @ [] { } | : ; ' , < > ? " whitespace`
- `whitespace ::= CR, LF, Tab, Space, FF, as returned by iswspace()`
- `any_char_but ::= any character except whitespace`
- `open_bracket := [`
- `close_bracket :=]`
- `Galaxy_identifier ::= <letter> | <digit>`

Notes

- `<tag_name>` is an object's unique name.
- `<contained_name>` is an object's optional contained name. It can be specified in a reference when an object is referred to as a contained child of another object.
- `<index>` is `-1` or a positive integer from 1 to 32767.
- `<identifier>` is limited to a maximum of 32 characters.
- An `<attribute>` name or `<primitive>` name can contain several `<identifier>` parts. The length of each `<identifier>` part can be up to 32 characters. Each `<identifier>` part is separated by a period. The maximum total length of the `<attribute>` name is 329. This name length applies to both static and dynamic attribute names. The maximum total length of the `<primitive>` name is 329.
- `<relative_name>` and `<property>` replacements are case insensitive, including `PROPERTY()`.
- If no attribute reference is specified, `.PV` is assumed. If `PV` is an attribute of type array, the resulting reference is invalid. For arrays, the `.PV[]` part must be explicitly supplied.

The exception to this rule is a reference that is preceded with an `@` sign. This reference refers to the object itself and not any particular attribute or property. Currently, this reference string format is used only in the **Execution Order** group on the **Object Information** page of the Object Editor.

- Do not use Property Names or InTouch Pseudo-Property Names for the names of primitives or attributes when enhancing an object's functionality on the **Scripts**, **UDAs** and **Extensions** pages.

ArchestrA Property Names include: Locked, Category, HasruntimeSetHandler, Name, Type, Quality, Dimension1, Value, SecurityClassification, 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 and 31.

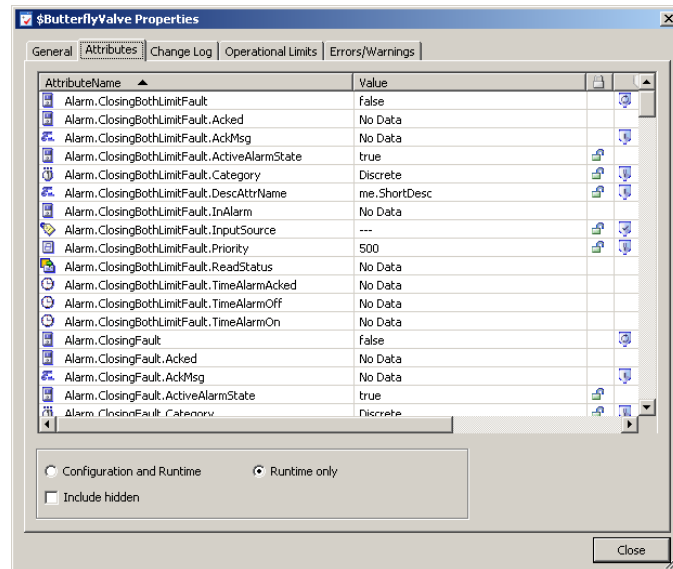
InTouch Pseudo-Property Names include: `#VString`, `#VString1`, `#VString2`, `#VString3`, `#VString4`, `#EnumOrdinal`, `#ReadSts`, `#WriteSts` and `#QString`. For more information on InTouch Pseudo-Property Names, see the *InTouch® HMI Data Management Guide*.

Viewing Attributes in Objects

Within the ArchedrA IDE, you can view the attributes in an object. This lets you see what attributes are available.

To view the attributes in a selected object

- 1 Select an object.
- 2 On the **Galaxy** menu, click **Properties**.
- 3 To see the references for the selected object, click the **Attributes** tab.



This page shows you:

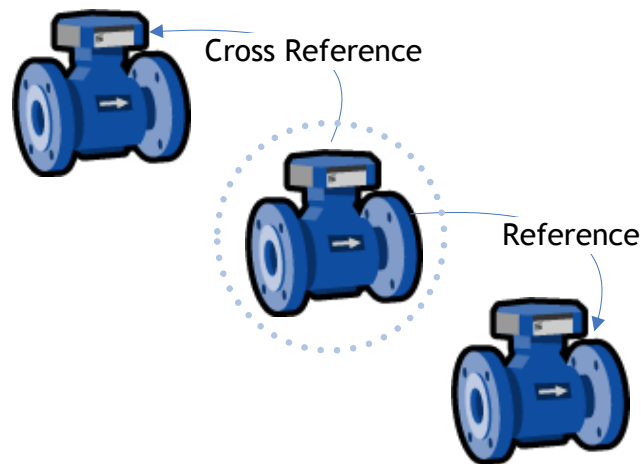
- **AttributeName** list The selected attributes for the object, the current value, and locked and security status. The information shown depends on whether **Configuration and Runtime** or **Runtime Only** is selected.
- **Value** Shows the value of the attribute.
- **Locked/Unlocked** Shows if the attribute is locked or unlocked.
- **Security** Shows the current security setting, if any.

- 4 To filter the view, select one or more of the following:
 - **Configuration and Runtime:** Select to show both configuration and run-time attributes for the selected object.
 - **Runtime only:** Select to show only run-time attributes for the selected object.
 - **Include hidden:** Select to show hidden attributes for the selected object.
- 5 When you are done, click **Close**.

Viewing References and Cross References

Objects have references and cross references.

- *References* are the objects the selected object is looking for.
- *Cross references* are the objects looking for the selected object.



You can view references and cross references for a selected object.

Some attributes are dynamic attributes. These are attributes that get created during run time and exist only in run-time. A device integration (DI) reference to a hardware register is a good example.

The attribute referencing a hardware register does not exist at configuration time by default. DI instances create the attribute dynamically during run time if the hardware register exists in the target device.

Note References and cross-references shown in the **Properties** dialog box only refer to interobject communications. Area associations, containment, or host assignments are not shown.

To view references and cross-references

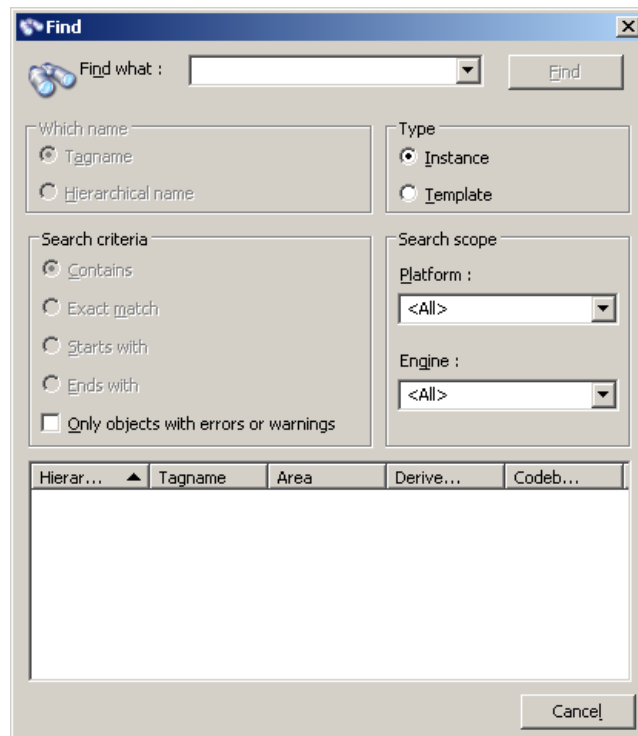
- 1 Select an object.
- 2 On the **Galaxy** menu, click **Properties**.
- 3 To see the references for the selected object, click the **References** tab. You see:
 - **Source Attribute** The attribute in the selected object referencing an attribute in another object in the application Galaxy.
 - **Attribute Reference** The reference string within the **Source Attribute**. This is either an absolute reference or a relative reference.
 - **Target Attribute** The absolute reference of the **Attribute Reference**. If this is a reference to a dynamic attribute, the **Target Attribute** only lists the Tagname name of the instance.
- 4 To see the cross references for the selected object, click the **Cross References** tab. You see the:
 - **Target Attribute** The absolute reference of the **Attribute Reference**. If this is a reference to a dynamic attribute, the **Target Attribute** only lists the Tagname name of the instance.
 - **Attribute Reference** The reference string within the **Source Attribute**. This is an absolute reference or a relative reference.
 - **Source Attribute** The attribute in the selected object that is referencing an attribute in another object in the application Galaxy.
- 5 When you are done, click **Close**.

Finding Objects

Your Galaxy can get very large and can include many objects. It can become difficult to find a specific object. You can search for templates or instances. Also, you can search by part or all of a tag name or hierarchical name.

To search for objects

- 1 On the **Edit** menu, click **Find**. The **Find** dialog box appears.



- 2 Do some or all of the following:
 - In the **Find what** box, type some or all of the name of the object.
 - In the **Which name** area, select either **Tagname** or **Hierarchical name** as the type of name you entered in the **Find what** box.
 - In the **Type** area, specify if you are looking for an **Instance** or a **Template**. If you select **Template**, the **Which name** and **Search scope** groups are unavailable.
 - In the **Search criteria** area, specify how to search for the name in the **Find what** box. The options are: **Contains**, **Exact match**, **Starts with** or **Ends with**.
 - Limit the search scope by selecting from the **Search scope** lists.

- 3 When you are done specifying the search criteria, click **Find**. The search results appear in the bottom pane.
- 4 Double-click an object in the results pane. The object is located and selected for you in the **Application views** area. If you double-click a Backup AppEngine, the IDE opens the **Deployment view** and the object is selected there. See Working with AppEngine Redundancy on page 307 for more information.

Using Galaxy References in InTouch

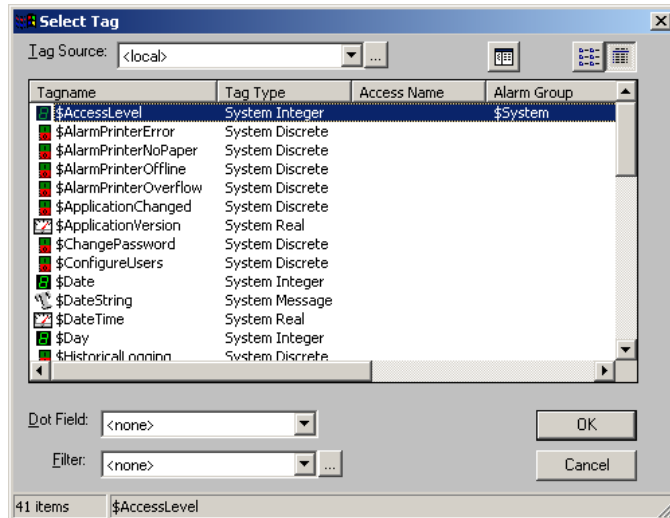
You can use Galaxy references in InTouch. Use the InTouch Tag Browser in unlimited selection mode to browse and include references from an Application Server Galaxy in InTouch applications you are developing.

Note You must install the Bootstrap and IDE on the InTouch node to create the TagSource to browse Galaxy objects.

The following lists the primary ways you can open the Tag Browser in InTouch to see unlimited selection mode:

- Double-click an animation link tagname or expression input box.
- Double-click an ActiveX or wizard tagname or expression input box.
- Double-click an empty area in any InTouch QuickScript window.
- In the InTouch QuickScript editor, select **Tagname** on the **Insert** menu.
- Press the **Alt+N** keys in the InTouch QuickScript editor.

- Double-click a blank **New Name** box in the **Substitute Tagnames** dialog box.
- Double-click the **Tagname** input box in the SQL Access **Bind List Configuration** dialog box.

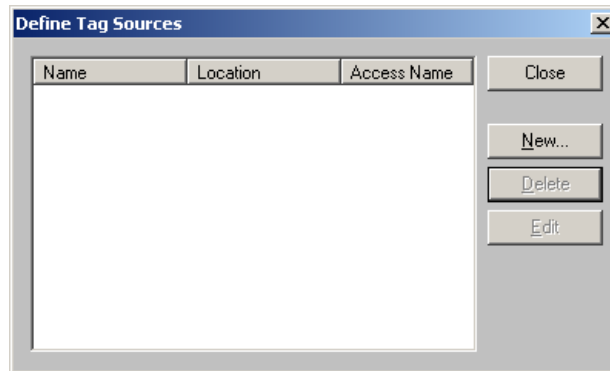


For complete information about using the InTouch Tag Browser, see the InTouch documentation.

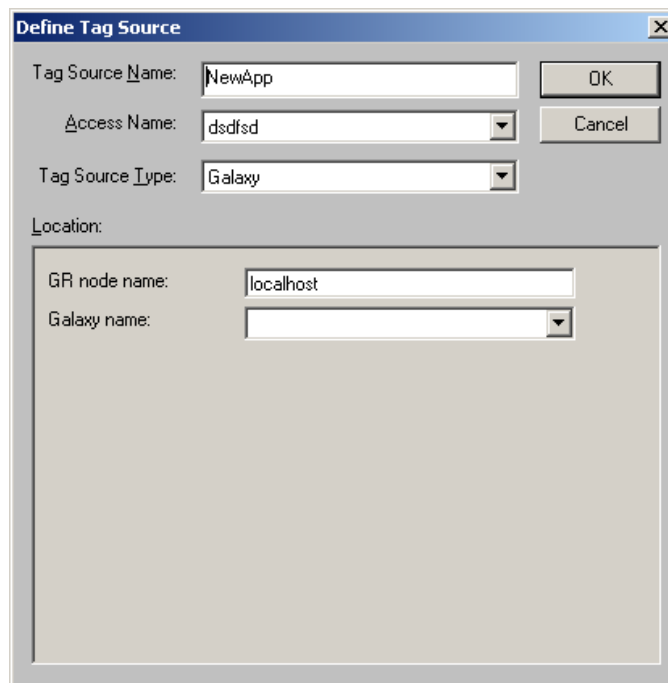
Before you can browse Galaxy references, you must define a new tag source.

To define a new tag source

- 1 In the InTouch HMI **Tag Browser**, click the **Browse** button to the right of the **Tag Source** list. The **Define Tag Sources** dialog box appears.

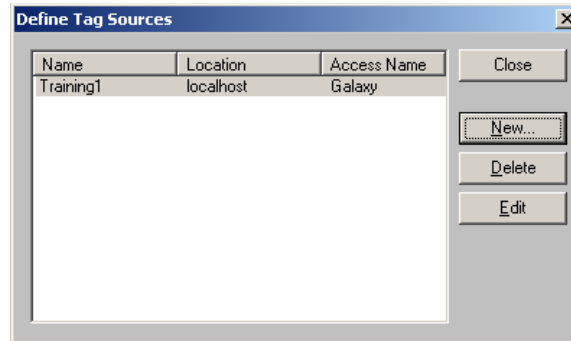


- 2 Click **New** to open the **Define Tag Source** dialog box.



- 3 Type a **Tag Source Name**. This name appears in the **Tag Source** box of the Tag Browser. For example: Training1.
- 4 Select Galaxy from the list for **Access Name** and **Tag Source Type**.
- 5 In the **GR node name** box, type the Host Name of the computer on which the Galaxy is located. If the Galaxy is located on the same computer, use localhost.

- 6 In the **Galaxy Name** list, select the name of your Galaxy. For example: Training. Assuming the examples given in steps 3 and 5, the **Define Tag Sources** dialog box appears as follows.



- 7 Click **Close**. The Tag Browser appears. Now you can browse the TagNames.

To browse attribute references in a Galaxy

- 1 Open the **InTouch Tag Browser** in unlimited selection mode.
- 2 In the **Tag Source** box, select the tag source you created in the previous steps (Training1 in the example). The **Attribute Browser** appears.
- 3 Select the object and attribute you want to reference in your InTouch application and click **OK**.

Note The next time you open the Tag Browser, the **Attribute Browser** automatically opens. To change that, exit the **Attribute Browser** without selecting anything by clicking the blue arrow at top right. The Tag Browser appears and it defaults to the InTouch Tagname Dictionary.

For more information about using the Attribute Browser, see Referencing Objects Using the Galaxy Browser on page 79.

Chapter 10

Working with Security

Galaxies are created without security. After a Galaxy is created, you can assign security to manage access. Using security lets you manage access to:

- IDE for configuring and managing objects.
- ArcestrA System Management Console (SMC) for performing maintenance and system administration functions.
- Any run-time operations.

This section describes the architecture of ArcestrA security and how to use it to manage access to configuration and run-time aspects of your Application Server application.

For more information on ArcestrA security, see the Wonderware Developer Network.

About Security

Security not only controls access to user interfaces in the ArcestrA environment but also controls access to object attributes and the data they represent.

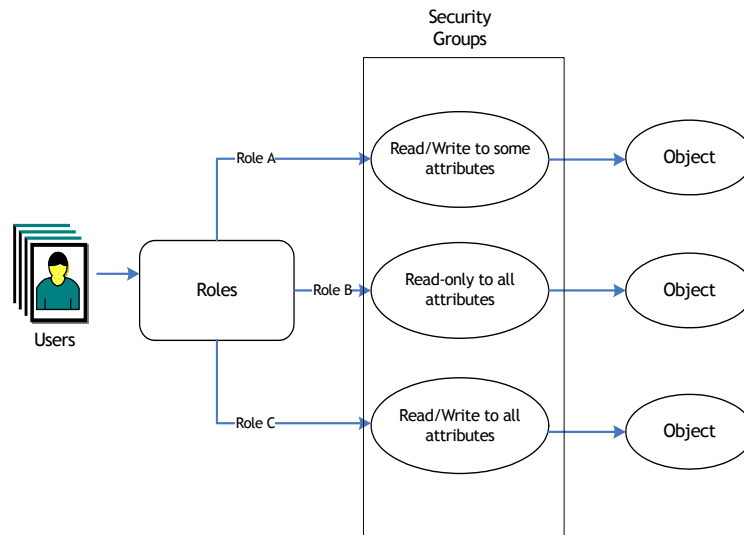
Each Galaxy in the Galaxy Repository manages its own security model. The security schema managed in a Galaxy is a three-level configuration model to create and maintain the following:

- Users associated with specific roles
- User roles associated with specific system administration, configuration and run-time (operational) permissions, which map to security groups
- Security groups associated with specific objects in the Galaxy

The default Galaxy Security model includes:

- Two users: DefaultUser and Administrator, both with full access to everything.
- One security group named Default.
- Two security roles: Default and Administrator, both with full privileges.

The security matrix defines a cascading model of users associated with specific roles that are associated with specific security groups that are associated with specific objects. User run-time permissions can vary from object to object, action to action, and process to process. The security icons associated with object attributes map directly to control points in the ArcestrA security model.



About Authentication Modes

You can select from three authentication modes to assign security:

- **Galaxy:** Uses local Galaxy configuration to authenticate users. All security for the Galaxy is specified and contained at the specific Galaxy level. When the user logs on, security credentials are checked and access to areas and activities is granted at the Galaxy level.
- **OS User Based:** Uses the operating system's user authentication system on an individual user level. All security for the Galaxy is specified and contained in the operating system (OS) on a user level basis. When the user logs on, security credentials are checked and access to areas and activities are decided at the OS user level.
- **OS Group Based:** Uses the operating system's user authentication system on a group basis. All security for the Galaxy is specified and contained in the user-to-roles mapping you created in the OS to assign security. When a user logs on, security credentials are checked and verified at the OS group level. OS groups are mapped to security roles in the Galaxy to allow access to areas and activities in the Galaxy.

Note If you are using OS user-based security or OS group-based security and you have permissions to use the IDE, the **Log In** dialog box does not appear.

For more information about OS security, see [About OS Group-based Security](#) on page 245.

Multiple Accounts Per User

Regardless of the security system, a single user can have multiple accounts. For example, a user can have an account that provides permissions for working with instances but not templates. The same person can have another supervisory account for working with templates and managing users in the ArcestrA environment.

Each account requires a different user name and password. For example:

User Name	Password	Access
bsmith	password	Instances, not templates
bobsmith	super	Instances, templates, not managing users
Robertsmith	admin	Instances, templates, managing users

Changing Security Settings

After you change security for a Galaxy, you see the following behaviors and conditions:

- When you change the authentication mode security, the IDE restarts.
- To switch users, the person must log on as the new user by clicking **Change User** on the **Galaxy** menu.
- If you previously configured security under one authentication mode and then switch authentication modes, only those users created while configuring the new mode are available. Other users are not deleted, just unavailable in the new mode.
- Objects that are reassigned to different security groups are marked as “pending update” and require redeployment for the change in security group to take effect.
- If security was previously configured for an OS-based authentication mode, reconfiguring security synchronizes the user’s full name and OS groups if some data in the OS has changed.

About Security Groups

Every object in the Galaxy belongs to only one security group. You can create and manage security groups that make sense for your organization. These security groups are mapped to roles on the **Roles** page.

Permissions determine what kind of access users have for each attribute. There are four basic operational permissions:

- Acknowledge alarms
- Change the value of attributes with security mode Configure
- Change the value of attributes with security mode Operate; this includes also security modes Secured Write and Verified Write
- Change the value of attributes with security mode Tune

By default, all currently used objects are assigned to a security group called Default.

A user who is a member of a role assigned to Security Role “Default” has permission to:

- Acknowledge alarms
- Change attribute values with “configure” security mode
- Change attribute values with “operate” security mode, including “secured write” and “verified write”
- Change attribute values with “tune” security mode

For example, you want users in certain roles to only have permission to acknowledge alarms that are generated from objects contained in Area1. You have a role named Area1Acknowledgers. You need to:

- 1 Create a new Security Group, for example **SecGrpArea001**.
- 2 Assign all objects that are contained in area Area1 to Security Role **SecRoleArea001**.
- 3 On the **Roles** page, select the **Area1Acknowledgers** role. In the **Operational Permissions the Security Group** for **SecGrpArea001**, select **Can Acknowledge Alarms**.
- 4 Any user that belongs to the **Area1Acknowledgers** role can at least acknowledge alarms of objects contained in the security group **SecGrpArea001**. They do not have any other operational permissions for those objects.

About Roles

You can create and manage user roles that apply to your organization's processes and work-based authorities. Two roles are defined by default: Administrator and Default.

You can specify General and Operational Permissions for each role.

- General permissions relate to application configuration and administration tasks.
- Operational permissions relate to the security groups listed on the **Security Groups** page. By default, the Administrator has all permissions.

Note You cannot modify the General permissions for the role of Administrator.

The **Operational Permissions** that can be associated with a role:

- **Can Modify “Operate” Attributes:** Allows users with operational permissions to do certain normal day-to-day tasks like changing setpoint, output and control mode for a PID object, or commanding a Discrete Device object.
- **Can Modify “Tune” Attributes:** Allows users to tune the attribute in the run-time environment. Examples of tuning are attributes that adjust alarm setpoints and PID sensitivity.
- **Can Modify “Configure” Attributes:** Allows users to configure the attribute's value. Requires that the user first put the object Off scan. Writing to these attributes is considered a significant configuration change, for example, a PLC register that defines a Discrete Device input.
- **Can Acknowledge Alarms:** Allows users to manually acknowledge an alarm in the run-time environment.

About Users

If you select either OS based authentication mode, users with local accounts are added to the **Authorized Users Available** list in the following format: `.\<username>`.

If you select **OS Group Based** authentication mode, the local account must exist on each node in the Galaxy for successful authentication of that user on any computer in the Galaxy.

Two users are defined by default when a new Galaxy is created: Administrator and DefaultUser. These cannot be deleted in an open security setting and they are both associated with the default roles, Administrator and Default.

Configuring Security

Before you open the security editor for a Galaxy, make sure:

- No other user is connected to the Galaxy.
- All objects in the Galaxy are checked in.
- Your user profile has configuration permissions to change Framework Configuration/Modify Security Model, if security is previously configured.

If you try to open the security editor before these conditions are met, a warning message appears and you are denied access.

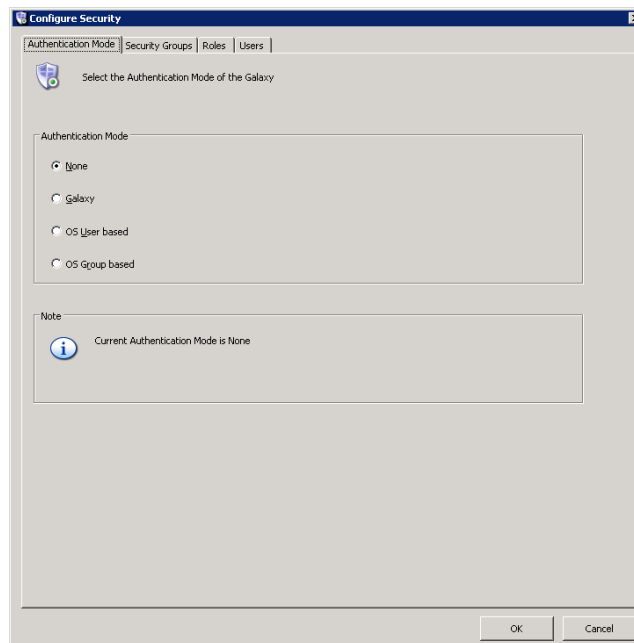
Caution Do not configure security settings of the IDE while an IDE-managed InTouch application is opened for editing in WindowMaker.

Other users who try to open the Galaxy while you are configuring security are denied access to the Galaxy.

Caution You can only change the ArcestrA administrator username or password using the Change Network Account Utility. The administrator account information is cached, and you may need to redeploy engines after you change the account so that the engines run using the current information. For more information about the utility, see About ArcestrA User Accounts on page 296.

To configure Galaxy security

- 1 On the **Galaxy** menu, click **Configure** and then click **Security**. The **Configure Security** dialog box appears.

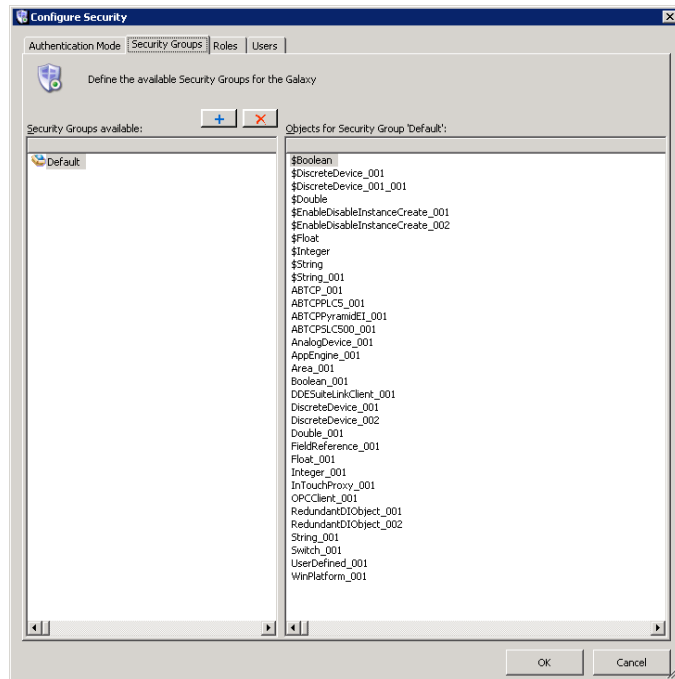


- 2 On the **Authentication Mode** tab, do the following:
 - Select the security type you want. Depending on what you select, more options become available.
 - If you select **OS Group-based** and you are working on a slow or intermittent network, you can specify the intervals in milliseconds:

Login Time: The time-out period (measured in milli-seconds) during which the system validates the user's membership against the OS groups selected as ArcestrA Roles. Minimum value is 0 (zero), maximum is 9,999,999. The default value is 1,000. If the login time is set to 0 (zero), which turns this feature off, the operation does not time out. Specify a value, based on the speed of your network and the number of groups configured in ArcestrA. The slower the network or the larger the number of groups, the greater the value.

Role Update: The time between each validation attempt per OS group for the user's membership when a log on is attempted. The user membership update is done one role per **Role Update** interval to minimize network usage. The minimum allowed value is 0 (zero) and the maximum is 9,999,999. The default value is 0 (zero), which turns off this feature so the operation does not pause between validating user membership and groups. This option operates independently of the **Login Time** option. Even if **Login Time** times out, the role update operation continues in the background and eventually updates user-to-role relationships for this user in the local cache. For more information about OS group-based security, see the Wonderware Development Network.

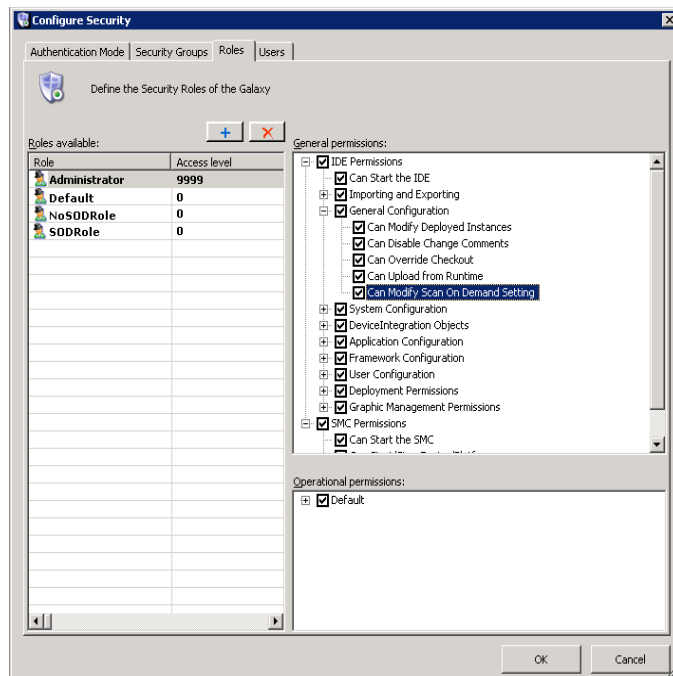
- Click **OK** or click the **Security Groups** tab.



- 3 On the **Security Groups** page, do the following:
 - Create a new security group by clicking the **Add** button. Type a unique name for the new group in the **Security Groups Available** pane. Security group names can be up to 32 alphanumeric characters, including a period. The name must include at least one letter and cannot start with \$.

Note Security group names are not case sensitive. Admin is the same as admin.

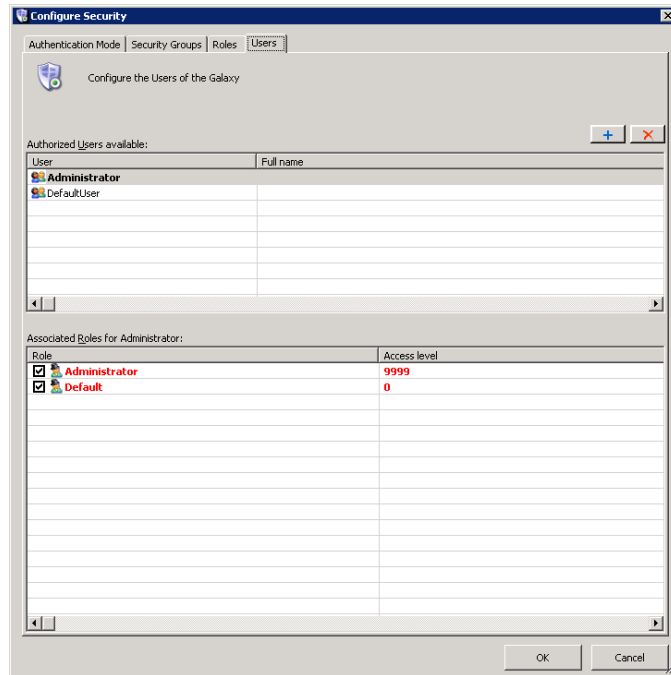
- Assign the objects you want the new group to have access to. Click the **Default** group. Drag the objects to the new security group.
- Click **OK** or click the **Roles** tab.



- 4 On the **Roles** page, do the following:
 - Create a new role by clicking the **Add** button. Type a name for the new role in the **Roles Available** pane. Role names can be up to 512 alphanumeric characters, including a period.
 - Select the **General** and **Operational Permissions** for the new role.

Important If a role is given “Can Modify Deployed Instances” permission, make sure “Can Create/Modify/Delete...” permissions in the System Configuration, Device Integration Objects, and Application Configuration groups are also selected. This provides the role with check in and undo checkout abilities.

- Click **OK** or click the **Users** tab.



5 On the **Users** page, do the following:



- Create a new user by clicking the **Add** button.

If you selected authentication as Galaxy, type a name for the user. User names can be up to 255 alphanumeric characters with no spaces.

If you selected an OS-based authentication, click the **Browse** button in the **Roles Available** pane, select an existing user and click **OK**. The user name appears as `.\<username>`.

While viewing Application Server events and alarms in InTouch, the “.” appears as the user’s domain if it is a local name. Otherwise, it appears as `<domain name>\<username>`.

Note **Users** and **Roles** in bold red text are invalid with the selected **Authentication Mode**.

- 6 When you are done, click **OK**. You are prompted to log on to the currently open Galaxy.

Assigning Users to Roles

After you create users and roles, you can assign users to roles. On the **Users** page, all users in the Galaxy and the roles they are assigned are listed.

By default, the new user is associated with the Default role but not the Administrator role. This cannot be changed as every user belongs to the Default role. Double-click in a text box to change, if needed.

Note All users are automatically assigned to the Default role in addition to any new roles you create and assign to them. The best way to manage permissions is to limit the permissions of the Default role to those permissions you want everyone to have. Then, add users to other roles with permissions for other parts of Application Server.

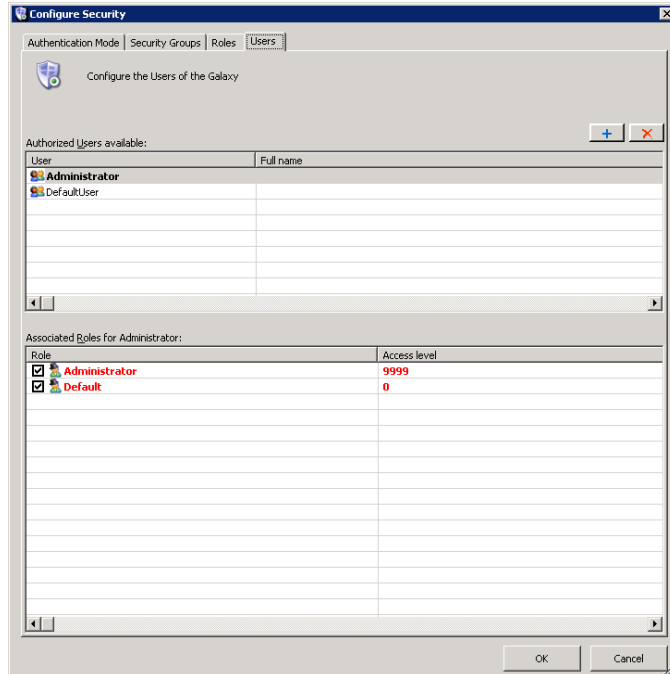
The Administrator user can log on any authentication mode except when security is disabled. When logged on as Administrator on the Galaxy Repository node, you can change the password of any Galaxy user without providing the old password.

- In Galaxy authentication mode, you can edit the **User Name** in the **Change Password** dialog box.
- In OS-based authentication modes, the **User Name** of the OS user is shown. User information cannot be edited.

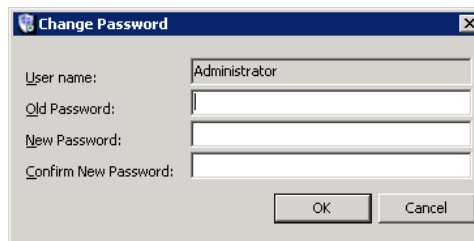
You can assign users to more than one role.

To assign a role to a user

- 1 On the **Galaxy** menu, click **Configure** and then click **Security**. The **Configure Security** dialog box appears.
- 2 Click the **Users** tab.



- 3 Select the user in the **Authorized Users available** area. Select a role in the **Associated Roles for <user name>** area.
- 4 Provide each user with a password by clicking **Change Password**. The **Change Password** dialog box appears.



Important If an OS-based security mode is selected on the **Authentication Mode** page, changing a user's password changes the OS password for the user. Any ArcestrA password may be set to a maximum of 127 characters.


- 5 Enter the correct information. This information is used in the configuration, administration and run-time environment to authenticate users.
- 6 Click **OK**.

Deleting Security Groups

You can delete a security group you no longer need. Before you can delete a security group, make sure no objects are associated with it.

You cannot delete the Default security group.

To delete a security group


- 1 On the **Galaxy** menu, click **Configure** and then click **Security**. The **Configure Security** dialog box appears. Click the **Security Groups** page.
- 2 On the **Security** page, select the group you want to delete.
- 3  Click the **Delete** button.

Deleting Roles

You can delete roles you no longer need. You cannot delete a role if any users are associated with it.

You cannot delete the Default and Administrator roles.

To delete a role


- 1 On the **Galaxy** menu, click **Configure** and then click **Security**. The **Configure Security** dialog box appears. Click the **Roles** tab.
- 2 On the **Role** page, select the role you want to delete.
- 3  Click the **Delete** button.

Deleting Users

You can delete users you no longer need.

Note You cannot delete a user who is currently logged on.

To delete a user

- 1 On the **Galaxy** menu, click **Configure** and then click **Security**. The **Configure Security** dialog box appears. Click the **Users** tab.
- 2 On the **User** page, select the user you want to delete.
- 3  Click the **Delete** button.

About OS Group-based Security

If you use OS Group-based Authentication Mode, make sure you understand the Windows operating system, particularly its user permissions, groups and security features. ArcestrA OS Group-based security uses these Windows features. For more help, see the Microsoft online help or a 3rd party book about Windows security.

When you use local OS Groups as Roles, each node within the Galaxy must have the same OS Users, Groups, and user-group mappings to get the same level of access to the user at each node.

Connecting to a Remote Node for the First Time

A newly-added user working on a computer with no access to the Galaxy Repository node cannot write to an attribute on a remote node if that user has never logged on to the remote node. This is true even if the user is given sufficient run-time operational permissions to do writes. To enable remote writing capabilities, log on to the remote node at least one time.

Cached Data at Log In

If you log on to ArcestrA on a workstation that belongs to Domain A and Domain Controller A fails, locally cached login data is used on subsequent log on attempts. When the domain controller returns to operation, your log on fails during the time period that trusts are being reestablished by the controller.

If, during the controller outage, your username/password data changed, you can use the old log on data if you intend to work locally.

If you want to perform remote operations, you should log on with the new log on data. If that fails, the trusts are being reestablished by the controller, and you should retry at a later time.

The user's full name is not available to any client (for example, an InTouch window) if the domain controller is disconnected from the network when the user logs on to ArcestrA for the first time.

If the user previously logged on to ArcestrA when the domain controller was connected, the user's full name is still available to the client from data stored in cache even if the domain controller is disconnected when the user subsequently logs on to ArcestrA.

Mixed or Native Domains

The list of domains and user groups appears differently in the group browser depending on whether your domain is configured as a Mixed or Native domain.

Your unique listing maps to the list of domains and user groups you see when you use the Windows tool for managing your domain. A domain group configured as “Distribution” instead of “Security” cannot be used for security purposes.

Using InTouch Access Levels Security

The **Roles** page includes the **Access Level** column. The **Access Level** is an InTouch function.

In InTouch, access levels are a schema for prioritizing run-time functions. In the ArcestrA security model, it only maps to InTouch values and has no prioritizing characteristics.

The maximum value is 9999 and the minimum is 0 (zero). If a user is assigned more than one role with different access levels, the higher access value is passed to InTouch.

Chapter 11

Working with Languages

Application Server language features enable you to develop applications that can be switched to another language at run time. To enable run-time language switching, you must:

- Configure multiple languages for the application.
- Export your application text for offline translation.
- Translate one or more exported dictionary files.
- Import one or more translated dictionary files.

This section describes the features and procedures for configuring languages and enabling run-time language switching.

Defining and Configuring Galaxy Languages

You can set the Galaxy default language and you can define additional languages for purposes of switching symbol and alarm comment languages at run time.

The default language is set and languages are added only at the Galaxy level.

Graphics Language Switching

Run-time language switching applies to all portions of the graphics or animations that you create or configure using the ArcestrA Symbol Editor or InTouch WindowMaker. The language to display is set when the application is being shown at run time.

You can only view translations at design time for ArcestrA symbols in the Symbol Editor. You cannot switch languages in WindowMaker at design time to see the translations.

Alarm Comment Language Switching

The alarm comment language switching feature allows you to format and export the configured alarm comments from attributes in a Galaxy to external files. The purpose of exporting the alarm comments to external files is to produce InTouch-compatible localized files to support language switching of alarm comments in the InTouch runtime environment.

You can import the alarm comment files back into the Galaxy after translation for convenient updating and re-export.

If you change an alarm comment after exporting the alarm comments, you must re-export the alarm comments. The translated alarm comments imported into InTouch and displayed at run time do not change dynamically when the alarm comment is edited.

Workflow

A typical workflow for a large Galaxy might consist of the following:

- 1 Export the Galaxy elements to be translated as a single file that is more manageable by the translator.
- 2 Import the translated file back into the Galaxy.
- 3 Export the translated file by Areas for use by InTouch applications.

- 4 Import the translated language file into InTouch.
 - In stand-alone InTouch, you can perform a single import operation.
 - In managed InTouch, you must import the language file into each managed InTouch application.
- 5 For InTouch applications that only target subareas of the Galaxy, exporting by Area is a more optimal workflow.

Important Exporting, translating and importing the translated files back into the Galaxy are important steps for translating and managing translated files. Note that run-time language switching will not function if you do not import the translated language file as described in number 4.

Configuring Languages for a Galaxy

The language settings of the Galaxy control which languages are available to symbols and alarm comments. You cannot add a language at the symbol or attribute level. Languages are only added at the Galaxy level using the IDE.

When you open a symbol for editing using the Symbol Editor or open a managed InTouch application in WindowMaker, the language settings are retrieved from the Galaxy.

For example, you configure English and French languages for the Galaxy. You open symbol S1 in the Symbol Editor. Symbol S1 is now configured with English and French languages. Using the IDE, you add German to the list of configured languages. You must close the Symbol Editor and open symbol S1 again to see the German language available for the symbol.

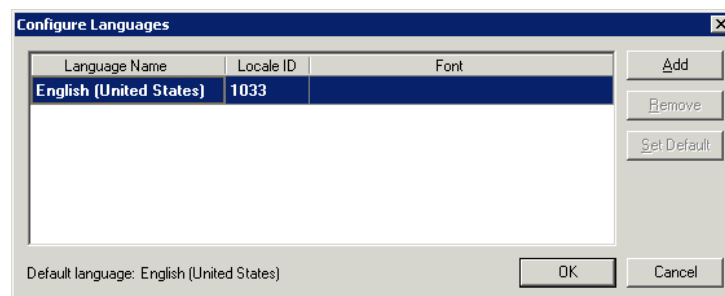
For more information about language switching in ArcestrA symbols, see the *Creating and Managing ArcestrA Graphics User's Guide*.

Adding a Language to a Galaxy

Every Galaxy is associated with a base language. You must configure any additional languages that you want to support.

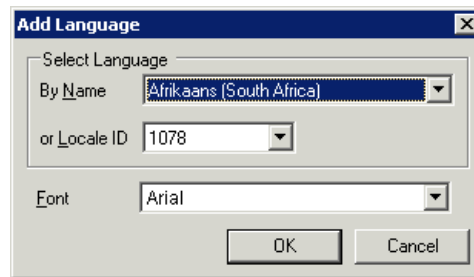
To add a language for a Galaxy

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to add a language.
- 2 On the **Galaxy** menu, point to **Configure**, and then click **Languages**. The **Configure Languages** dialog box appears.

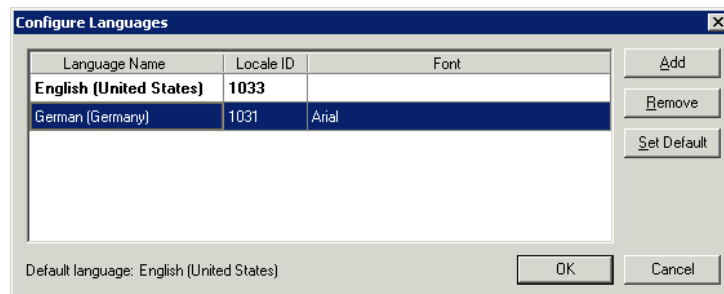


The **Configure Languages** dialog box shows the base (default) language of the Galaxy.

- 3 Click **Add**. The **Add Language** dialog box appears.



- 4 Specify the language and font for the translated text.
 - In the **By Name** or the **Locale ID** list, click the language to add. If you select the language by name, the corresponding locale ID appears in the **Locale ID** list, and vice versa.
 - In the **Font** list, click the name of the font.
- 5 Click **OK** to close the **Add Language** dialog box. The language you configured is listed in the **Configure Languages** dialog box.



- 6 To add more languages, repeat steps 3 through 5.
- 7 To remove a language, select the row in the list and then click **Remove**.
- 8 To specify a particular language as the default, select the row in the list and then click **Set Default**.
- 9 When you are done, click **OK**.

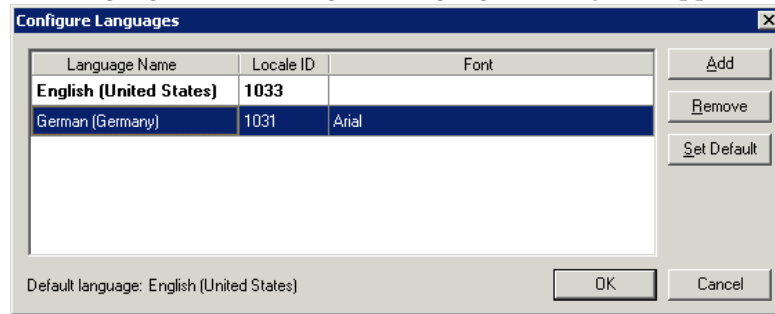
Removing a Language from a Galaxy

You cannot remove the default language. At least one language must be configured for a Galaxy.

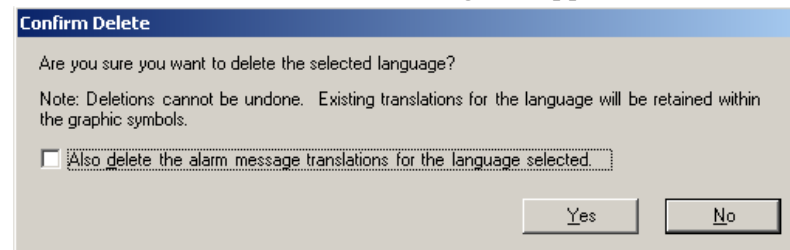
To remove a language for a Galaxy

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to remove a language.

- On the **Galaxy** menu, point to **Configure**, and then click **Languages**. The **Configure Languages** dialog box appears.



- Select the row of the language to remove and then click **Remove**. A **Confirm Delete** dialog box appears.



- The dialog box provides an additional language deletion option. Select the checkbox to delete alarm comment translations for the language selected. Leave the checkbox empty (unselected) if you want to keep the alarm comment translations for the selected language.
- Click **YES** to confirm deletion of the selected language as well as the alarm comment translations for that language if also selected.

Modifying the Font for a Language

The default font for all languages is Arial. You can change the font setting for a language that you have already added. You cannot edit the font for the default language.

The configured font for a language is used in design time when a specific translation is supplied for a language. It is also propagated to all translations for secondary languages when the fonts are not specifically overridden by you.

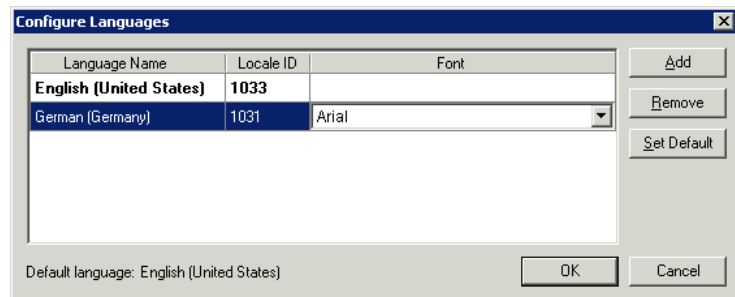
For example, if you create a text element and provide translation for the secondary language without modifying the font, the text is shown using the font from the Galaxy. However, if you manually modify the font, then the text will always use the chosen font. This same behavior also applies to run time.

For an ArcestrA symbol to show the updated font from the Galaxy, WindowViewer must be restarted. No notification is provided to WindowViewer when the Galaxy font changes.

All managed InTouch applications are updated to use the new font for the selected language.

To change the font settings for a configured language

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to change the font for a configured language.
- 2 On the **Galaxy** menu, point to **Configure**, and then click **Languages**. The **Configure Languages** dialog box appears.



- 3 In the list of languages, select the target language.
- 4 In the **Font** column, double-click on the name of the font so that a drop-down arrow appears.
- 5 Click the name of the new font.
- 6 Click **OK**.

Changing the Default Language for a Galaxy

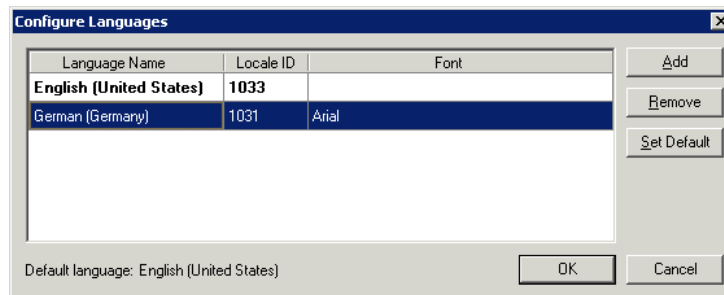
When you change the default language for the Galaxy:

- The new default language is shown when the Symbol Editor opens.
- The base language for translation export is changed to the new default language.
- You cannot import translations for that language.
- Symbols opened in design time use the default language when showing text that does not have a specific translation in a secondary language.
- Symbols shown at run time use the default language if specific translated values are not provided for the currently viewed language.
- Thumbnails for symbols are shown using the default language.

The default language for managed InTouch applications is always the same as the InTouch installed language.

To change the default language for a Galaxy

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to change the default language.
- 2 On the **Galaxy** menu, point to **Configure**, and then click **Languages**. The **Configure Languages** dialog box appears.



The **Configure Languages** dialog box shows the base language of the Galaxy.

- 3 In the list of languages, select the target language.
- 4 Click **Set Default**.
- 5 Click **OK**.

Note Changing the default language does not change the alarm comment default language. Alarm comments are always stored as the Galaxy Repository default language, which is the locale of the operating system on which the Galaxy is created.

Exporting Symbol Text for Offline Translation

If your application has many strings, you typically send the text strings for your graphic symbols out for bulk translation. You can export symbol strings for translation and modify them using a text editor, an XML editor, or a spreadsheet program like Microsoft Excel.

We recommend that you do not make changes to symbols while symbol text is being translated.

If you make changes to your application after you export your dictionary files, you must export the dictionary file again. For more information, see [Exporting Symbol Text to an Existing Dictionary File](#) on page 260.

You can only export the text strings for one language at a time. You cannot export text strings for the default language.

Each symbol in a Galaxy is only exported one time, no matter how many times it is referenced.

When you export the text, you specify a folder for the dictionary files. Creating a new folder to export phrases for each language makes it easy to manage dictionary files. For example, ...*Galaxy1*\My German Files\.

Also, all exported files have the following convention: *<GalaxyName>_<LanguageID>.xml*. If you will be exporting language data for different objects at different times, use separate target directories to prevent subsequent exports from overwriting the first export.

When you export the dictionary for an application, the files are .xml files that you can edit using Microsoft Excel 2003 or later.

Types of Language Dictionary Files

A language export creates the following types of dictionary files:

- Dictionary file for all Graphics Toolbox symbols, template symbols, and AutomationObject symbols. The file naming convention is: *<GalaxyName>_<LanguageID>.xml*. For example, if the Galaxy name is TestSample and the language being exported is French (Language ID = 1036) then the file name is TestSample_1036.xml.
- Dictionary file for each managed InTouch application.
- Dictionary file for each SmartSymbol in managed InTouch applications.

Note The alarm comment export files are named and formatted differently from symbol dictionary files. For further information see *Exporting Alarm Comments for Offline Translation* on page 269.

Exporting Language Data for All Symbols in a Galaxy

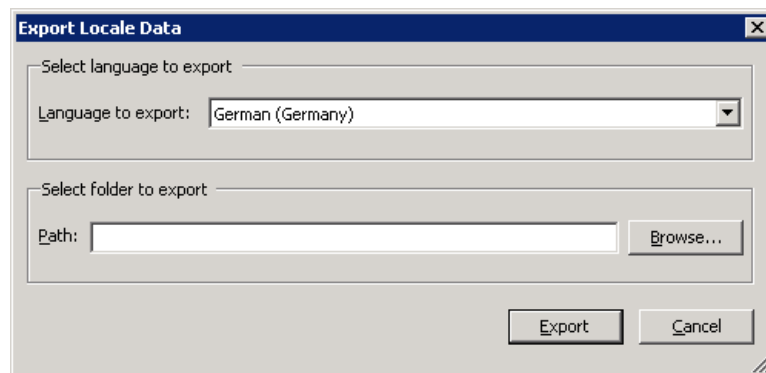
You can export language data for all symbols in a Galaxy at one time. The export contains language data for:

- Graphic Toolbox symbols.
- All symbols contained in AutomationObject templates, except for an \$InTouchViewApp template.
- All symbols contained in AutomationObject instances.

The export operation only applies to symbols that are checked in.

To export language data for all symbols

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to export symbol text.
- 2 On the **Galaxy** menu, point to **Export**, and then click **All Symbols**. The **Export Locale Data** dialog box appears.



- 3 Configure the export settings.
 - In the **Languages to export** list, select the language dictionary to export. The default language is not listed.
 - In the **Path** box, type the folder to which you want to export the dictionary. Click **Browse** to select an existing folder or create a new folder.
- 4 Click **Export**. The export progress is shown.
- 5 Click **Close**.

Exporting Language Data for Specific Objects

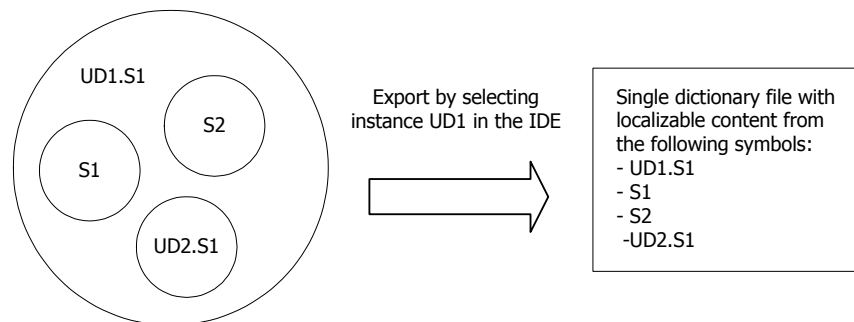
You can export language data for these types of Galaxy objects:

- Graphic ToolBox symbols
- AutomationObject templates and instances
- \$InTouchViewApp templates and instances.

All symbols in an instance or template are exported.

When you export the language data for a symbol, the language data for all symbols referenced by the symbol is also exported. The exported symbols can be referenced through direct embedding or through a show symbol animation.

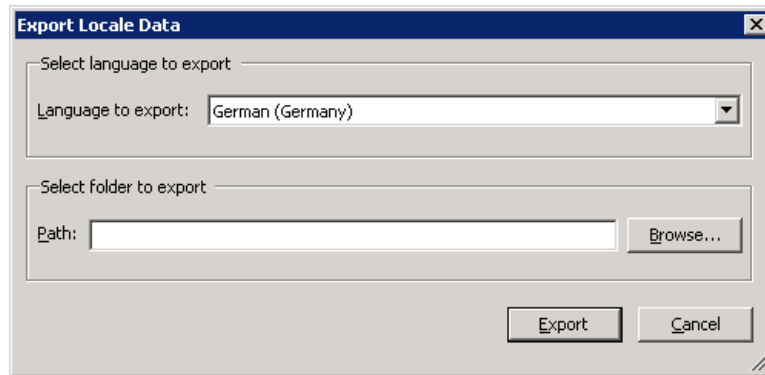
For example, a Galaxy has Symbol1 and Symbol2 defined in the Graphic Toolbox. There are two instances called UserDefined1 and UserDefined2. UserDefined1 includes Symbol1. UserDefined2 includes Symbol1 and Symbol2. The instance symbol UserDefined1.Symbol1 embeds Symbol1 and Symbol2 from the Graphics Toolbox and one instance symbol UserDefined2.Symbol1. If you select the instance UserDefined1 and export the language data, then the language data would also be exported for the symbols Symbol1, Symbol2, and UserDefined2.Symbol1.



To export language data for specific objects

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to export symbol text.
- 2 Select one or more objects to export.

- 3 On the **Galaxy** menu, point to **Export**, and then click **Localization(s)**. The **Export Locale Data** dialog box appears.



- 4 Configure the export settings.
 - In the **Language to export** list, select the language dictionary to export. The default language is not listed.
 - In the **Path** box, type the folder to which you want to export the dictionary. Click **Browse** to select an existing folder or create a new folder.
- 5 Click **Export**. The export progress is shown.
- 6 Click **Close**.

Exporting Symbol Language Data for a Managed InTouch Application

You export language data for a managed InTouch application using the IDE. You cannot export translations from within the InTouch HMI.

The export includes strings for:

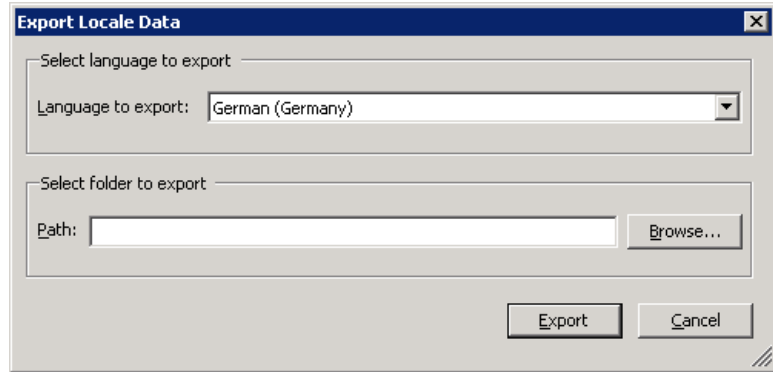
- All InTouch windows
- All SmartSymbols
- ArcestrA symbols referenced by the \$InTouchViewApp template

The export causes a cascade export of all referenced ArcestrA symbols.

When you export language data for a managed InTouch application, the default language for the Galaxy is ignored. The InTouch default locale is always the installed InTouch locale. If the InTouch installed locale and the Galaxy default language are not the same, the export is skipped for the selected InTouchViewApp and a message is logged.

To export language data for a managed InTouch application

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to export symbol text.
- 2 Select one or more managed InTouch applications.
- 3 On the **Galaxy** menu, point to **Export**, and then click **Localization(s)**. The **Export Locale Data** dialog box appears.



- 4 Configure the export settings.
 - In the **Language to export** list, select the language dictionary to export. The default language is not listed.
 - In the **Path** box, type the folder to which you want to export the dictionary. Click **Browse** to select an existing folder or create a new folder.
- 5 Click **Export**. The export progress is shown.
- 6 Click **Close**.

Exporting Symbol Language Data for a Published InTouch Application

If you export languages for a published InTouch application, the following are not included:

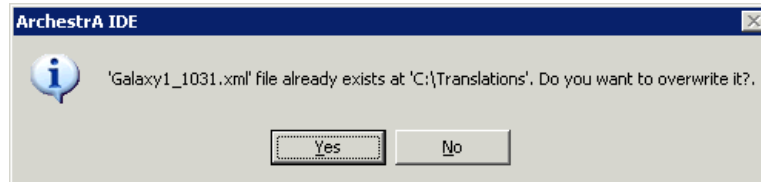
- ArcestrA embedded symbols
- Custom property overrides
- String overrides.

The export only includes native InTouch translations.

Exporting Symbol Text to an Existing Dictionary File

If you change your application after you translate the text strings, you need to export the text again. For more information, see [Exporting Symbol Text for Offline Translation](#) on page 255.

If you export more than one time to the same directory, a message box appears.



If you click **Yes**, the existing .xml files are deleted and the current text for symbols in the Galaxy are exported as a new .xml file.

Any existing translations for a symbol are reflected in the new .xml file.

Translating Exported Symbol Language Files

The procedures and tools for translating the exported language files are similar for both symbol text and for alarm comment text. However, there are important differences. Procedures for each are described in this section.

Translating Exported Symbol Text Dictionary Files

After you export the dictionary file containing your application text, use Microsoft Excel 2003 or later to edit the text or, for very large Galaxies, Excel 2007.

To translate an exported dictionary file

- 1 Open the XML file in Excel. The **Open XML** dialog box appears.
- 2 Click **As an XML** list, then click **OK**. A message may appear informing you that an XML schema will be created.
- 3 Click **OK**.

The XML file opens in Excel with columns for the:

- Phrases in your application.
- Translated phrases from the translator.
- Translated font name.
- Translated font properties.
- Translated font size.
- Base font properties.
- Base font size.
- Context, phrase ID, language ID and foreign language ID.

1	Phrase	Translation	Translate	Translated	Translate	BaseFont	BaseFontPro	Base	Context
2	Do not edit	Provide translat	Can edit	Can edit	Can edit	Do not edit	Do not edit	0	Galaxy1
3	#					Tahoma		8	AnalogHiLo:Si
4	#					Tahoma		8	AnalogHiLo:Si
5	#					Tahoma		8	AnalogHiLo:Si
6	#					Tahoma		8	AnalogHiLo:Si
7	#					Tahoma		8	AnalogHiLo:Si
8	LABEL					Tahoma		8	AnalogHiLo:Ls
9	LOW					Tahoma		6	AnalogHiLo:Lc
10	HIGH					Tahoma		6	AnalogHiLo:Hi
11	LABEL					Tahoma		10	AnalogMeterI
12	UNITS					Tahoma		8	AnalogMeterI

Important Only modify data in the **Translation**, **TranslatedFontSize**, **TranslatedFontName**, and **TranslatedFontProperty** columns. Do not change any column header. Do not insert or delete rows.

- 4 Type the language-specific text in the **Translation** column in the row that corresponds with the base language string in the **Phrase** column.
- 5 If necessary, change the font parameters for the translated strings. If you only provide a translation, the Galaxy-configured font for the language is used to render the text after the translation is imported. If you specify a font, it overrides the Galaxy-configured font.
 - In **TranslatedFontName** column, type the font name.
 - In the **TranslatedFontProperty** column, type the notation for the font properties:
B = **bold**
I = *italic*
U = underline
For example, if you want the text to be bold, type **B** in the **TranslatedFontProperty** column. If you want the text to be bold and underlined, type **BU** in the **TranslatedFontProperty** column.
- 6 Save the file using XML Data as the file type.

Important If you save as another file type, such as XML Spreadsheet, Excel changes the schema and the Galaxy cannot load the file. If you change the name of the XML file, the file will not import properly into the Galaxy.

Importing Translated Symbol Language Files

You can import alarm comment language files for re-export by area, to facilitate the export of new, untranslated alarm comments, and to facilitate re-export of previously translated alarm comments that have been changed.

For symbol text, you must import the translated dictionary files for each language to enable run-time language switching for those languages. All dictionary files for a given language should be placed in the same folder.

You can import files for only one language at a time. When you import, you select the desired language and specify the files to import.

Importing Translated Symbol Dictionary Files

All affected symbols and relevant objects are checked out before the import begins and are checked in when the import is done. If an affected symbol or object is already checked out, a message appears in the progress dialog box, and the import is skipped for the checked out object.

For a published InTouch application, only the native InTouch translations are imported.

You can configure how you want Galaxy and symbol/object mismatches handled during the import.

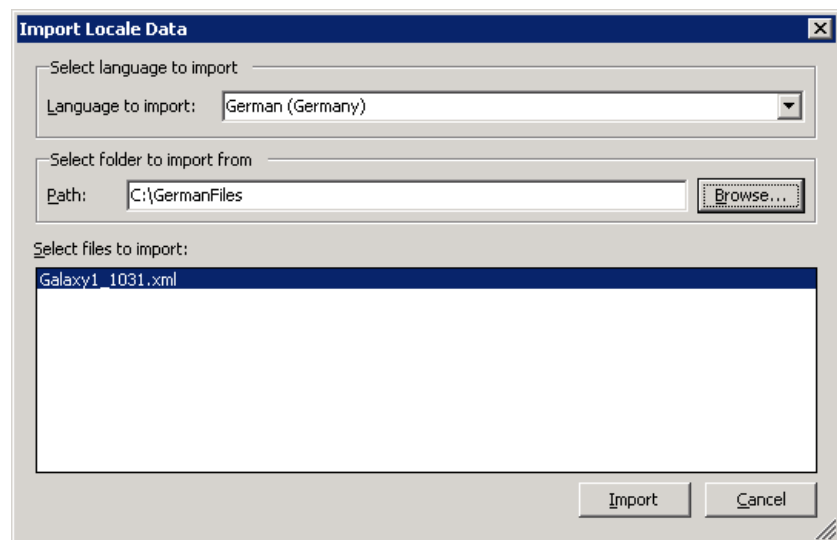
The import is skipped if:

- If default locale specified in the .xml translation file does not match the current default locale of the Galaxy.
- An InTouchViewApp's installed locale does not match the current default locale of the Galaxy.

Important You cannot cancel the import after it starts.

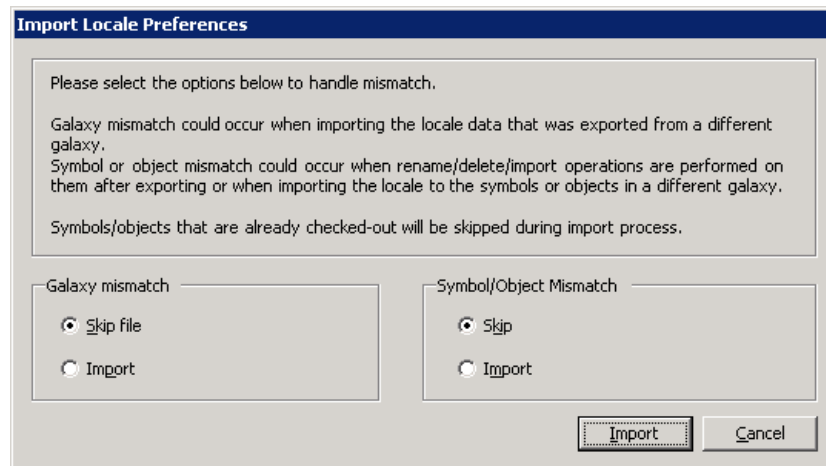
To import a translated dictionary file

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to import symbol text.
- 2 Close all editors and check in all Galaxy objects.
- 3 On the **Galaxy** menu, point to **Import**, and then click **Localization**. When a message appears, click **OK**. The **Import Locale Data** dialog box appears.



- 4 Configure the import settings.
 - In the **Language to import** list, select the language dictionary to import.

- In the **Path** box, specify the folder that includes the dictionary file to import.
 - In the **Select files to Import** box, select the .xml files to import. Only files that include the current Galaxy name and the locale ID for the selected language are shown.
- 5 Click **Import**. The **Import Locale Preferences** dialog box appears.



- 6 In the **Galaxy mismatch** area, configure how you want Galaxy mismatches handled. A Galaxy mismatch occurs when you try to import a translation file that was exported from a different Galaxy. The Galaxy name in the translation .xml file is used to match the current name of the Galaxy.
- Click **Skip file** to skip all the files that do not contain the current Galaxy name in the file name.
 - Click **Import** to import all the selected files, regardless of what the Galaxy name is in the .xml filename.
- In the **Symbol/Object Mismatch** area, configure how you want symbol and object mismatches handled. A symbol or object mismatch occurs when the name of the symbol and the internal ID (GObjectID) of the symbol do not match what is within the .xml file. Objects include AutomationObjects and InTouchViewApp objects.
 - Click **Skip** to skip the symbols and objects that have mismatch names or mismatch IDs in the .xml file

- Click **Import** to import a symbol or object only if it has a matching name or matching ID. If the name resolves to one object, and the ID resolves to another object, then the import is skipped.

For examples, see Examples of Symbol or Object Mismatch Handling during Language Imports on page 265.

- 7 Click **Import**. The import progress is shown.
- 8 Click **Close**. The **Import Language Dictionary Files** dialog box appears.
- 9 Click **Check In**. The check in progress is shown.
- 10 Click **Close**. A summary of the import is shown.
- 11 Click **OK**.

Examples of Symbol or Object Mismatch Handling during Language Imports

The following table shows an example of handling mismatch conditions for a toolbox symbol. The bold name/ID is the matching name/ID in the .xml file and current Galaxy during import.

	Symbol Name and ID in the Galaxy while Exporting	Symbol Name and ID in the Galaxy while Importing	Change made after export and before import	Skip Option Selected	Import Option Selected
1	S1, 100	S1, 100	No change	Import to S1	Import to S1
2	S1, 100	S2, 100	Rename S1 to S2	Skip	Import to S2
3	S1, 100	S1, 200	Delete S1, Create/Import S1	Skip	Import to S1
4	S1, 100	S1, 200 S2, 100	Rename S1 to S2 Create/Import S1	Skip	Ambiguous, skip import
5	S1, 100	No S1 and No 100 in the Galaxy	Deleted S1	Skip	Skip import

The following table shows an example of handling mismatch conditions for an AutomationObject symbol. The bold name/ID is the matching name/ID in the .xml file and current Galaxy during import.

	Symbol Name and ID in the Galaxy while Exporting	Symbol Name and ID in the Galaxy while Importing	Object Name and ID in the Galaxy while Importing	Change made after export and before import	Skip Option Selected	Import Option Selected
1	Pump1, 10	S1, 100	Pump1, 10	No change	Import to S1 only if Pump1 has S1 with an ID of 100	Import to S1 only if Pump1 has S1 with an id of 100
2	Pump1, 10	S1, 100	Pump2, 10	Rename Pump1 to Pump2	Skip	Skip import if S1 and 100 pointing to two different symbols in Pump2 (ambiguous) Import to S1 only if Pump2 has S1 or a symbol with id of 100.

	Symbol Name and ID in the Galaxy while Exporting	Symbol Name and ID in the Galaxy while Importing	Object Name and ID in the Galaxy while Importing	Change made after export and before import	Skip Option Selected	Import Option Selected
3	Pump1, 10	S1, 100	Pump1 , 20	Delete Pump1 Create and Import Pump1	Skip	Skip import if S1 and 100 pointing to two different symbols in Pump1 (ambiguous) Import to S1 only if Pump1 has S1 or a symbol with id of 100.
4	Pump1, 10	S1, 100	Pump1 , 20 Pump2 , 10	Rename Pump1 to Pump2 Create and Import Pump1	Skip	Ambiguous, skip import
5	Pump1, 10	S1, 100	No Pump1 and No 10 in the galaxy	Deleted Pump1	Skip	Skip import S1

Language Data Handling for Galaxy Operations

If you import or export objects, all language data within the associated symbols is imported or exported.

You can export all objects in the Galaxy, selected instances/templates, or selected symbols from the Graphics Toolbox. All language data is exported as part of the graphics definition, regardless of the languages configured in the Galaxy.

You can import Galaxy objects that contain language data from the same Galaxy or a different Galaxy. All language data in the ArcestrA graphics is imported, regardless of what languages are configured for the target Galaxy.

No ArchestrA symbol or InTouchViewApp language data is read or modified during the import operation.

If you import an unmanaged InTouch application, the configured locales in the Galaxy are applied to the unmanaged InTouch application. No element translations in the InTouch application are removed during import, even though languages may no longer be visible in the InTouch HMI or available at run time in WindowViewer.

Exporting Alarm Comments for Offline Translation

As with symbol text export for translation, you typically send the alarm comment text out for bulk translation. You can export alarm comments and modify them using a text editor or a spreadsheet program such as Microsoft Excel. Once translated the files can be reimported to Application Server and can be imported by InTouch for run-time alarm comment language switching.

Guidelines and Recommendations

The following guidelines and recommendations will help you make best use of the alarm comment language switching feature:

Important The exported file name is generated automatically and must not be changed.

Organizing Your Export

- You can only export the alarm comment text for one language at a time.
- When you export text, you specify a folder for the language files. We recommend that you create a separate folder for each language. For example, ...\`Galaxy01\ChineseFiles\`.
- For large Galaxies, exporting files for each area, after being translated and re-imported in the Galaxy, will perform faster when the alarm clients access those files.
- For small Galaxies or for InTouch applications, we recommend that you export the entire Galaxy for translation.

Translation File Formatting and Editing

Microsoft Excel 2007 is the recommended spreadsheet program. Excel 2007 provides enhanced formatting capabilities to convert the text file into tabular format. Also, Excel 2007 supports larger worksheets. See Exporting Alarm Comments from Very Large Galaxies on page 271.

Reimporting

- We highly recommend reimporting the translated files to the Galaxy after translation. Although reimport is not mandatory, you can use the reimport to optimize the export files to be used in the InTouch application.

- If you make changes to your alarm comments after you export comment text, you must export the alarm comments again. For this reason, we further recommend using the import functionality in Application Server to reimport the language file after each translation. This practice will avoid the need to retranslate alarm comments you have already translated.

For further information and procedures, see the following sections in this chapter:

Exporting Alarm Comments from Very Large Galaxies on page 271

Exporting Alarm Comments by Area on page 272.

Importing Translated Alarm Comment Language Files on page 276

About the Alarm Comments Language File

The alarm comment language switching feature exports alarm comments to a .txt file. That file can be edited directly, but working in a text editor typically is more difficult and prone to errors.

- All files export as Unicode. We recommend that you save the exported files as Unicode.
- We recommend that you open the .txt file in Microsoft Excel 2003 or later for editing and save it as a .txt file.

The language file name is not user-configurable. The file naming convention for alarm comment export is:

Galaxy_<GalaxyName>_<localeID>_Alarm_Comments.txt
t. or

*Galaxy_<GalaxyName>_<AreaName>_<localeID>_Alarm
_ Comments.txt*, where the locale ID is the selected
language decimal identification number.

For example, if the Galaxy name is “TestSample” and the exported language is Chinese, the language file name would be *Galaxy_TestSample_2052_Alarm_Comments.txt*.

For example, if the Galaxy name is “TestSample” the Area name is “Area001”, and the exported language is German, the language file name would be *Galaxy_TestSample_Area001_1031_Alarm_Comments.txt*

See [Translating Exported Alarm Comment Language Files](#) on page 274 for further information about alarm comment language files.

Exporting Alarm Comments from Very Large Galaxies

You can export language data in one operation for all alarm comments in a Galaxy. However, very large Galaxies with numerous alarm comments require longer export processing time and can result in a very large .txt language file. We recommend the following best practices for handling large Galaxies and a large number of alarm comments:

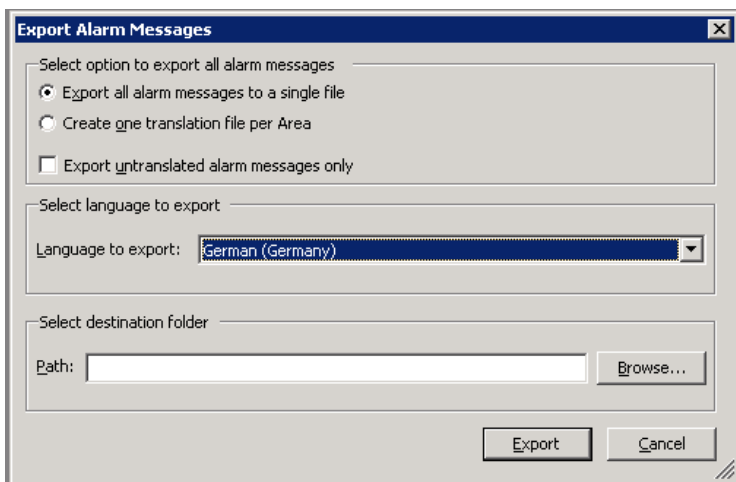
- Export alarm comments by Area rather than for the entire Galaxy to a single language file. See [Exporting Alarm Comments by Area](#) on page 272.
- If you prefer to export all Galaxy alarm comments to a single file, we recommend using Microsoft Excel 2007 rather than Excel 2003. MS Excel 2007 supports 1-million-line worksheets whereas Excel 2003 supports 65-thousand-line worksheets.

Exporting All Galaxy Alarm Comments

You can export language data for all alarm comments in a Galaxy at one time. The export operation only applies to objects that are checked in.

To export all Galaxy alarm comments to a single file

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to export alarm comments.
- 2 On the **Galaxy** menu, select **Export**, then select the **Localization** menu option, and then select the **All Alarm Messages** menu option. The **Export Alarm Messages** dialog box appears.



- 3 Configure the export settings:
 - a In the **Select option to export all alarm messages** area, select the **Export all alarm messages to a single file** radio button.
 - b Check the **Export untranslated alarm messages only** check box if you have already exported, translated and re-imported your language file and you want to export only newer, untranslated messages. Otherwise, leave the check box unchecked.
 - c In the **Language to export** pull-down list, select the language to export. You can export only one language at a time.
 - d In the **Path** box, type the folder to which you want to export the language file. Click **Browse** to select an existing folder or create a new folder.
- 4 Click **Export**. The export progress box appears.
- 5 Click **Close** when the export completes.

Exporting Alarm Comments by Area

You can export all alarm comments by Area or you can export Alarm comments for a specific Area.

Using File Names

The

Galaxy_<GalaxyName>_<AreaName>_<localeID>_Alarm_Comments.txt file name convention applies to specific Area alarm comment exports. Area names will be included the file name. For example, given a Galaxy name “TestSystem”, an Area name “Area001”, and an export to Chinese (language designation 2052), the language file name would be *Galaxy_TestSystem_Area001_2052_Alarm_Comments.txt*.

Exporting Objects Not Assigned to an Area

System Objects typically are not assigned to an Area. These are:

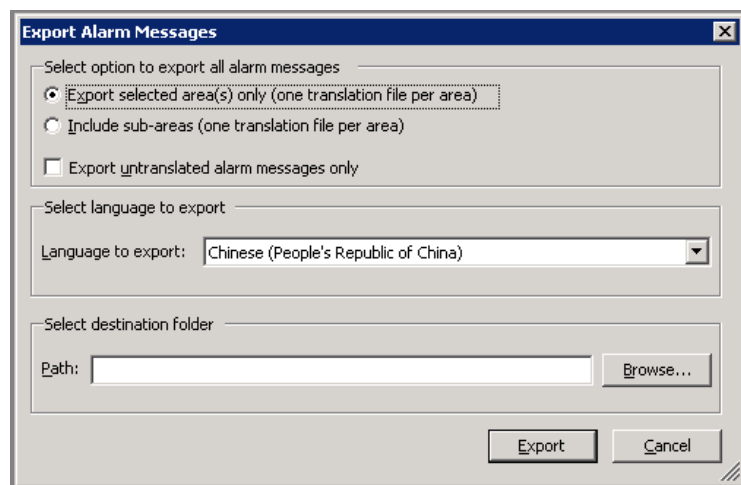
- Platform Objects
- Device Integration Objects
- Engine Objects

For purposes of language export, these System Objects form their own “Area” or become notification distributors for the purpose of sending alarms. Exporting System Objects follows these important conventions:

- System Object can only be exported from the Galaxy menu. They cannot be selected and exported using a context menu.
- System Objects can only be exported by exporting alarm comments for all Areas.

To export alarm comments for all Areas.

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to export alarm comments.
- 2 On the **Galaxy** menu, select **Export**, then select the **Localization** menu option, and then select the **All Alarm Messages** menu option. The **Export (Area) Alarm Messages** dialog box appears.



- 3 Configure the export settings:
 - a In the **Select option to export all alarm messages** area, select the **Export selected area(s) only** radio button, or select the **Include sub-areas** radio button to include all sub areas for each selected Area.
 - b Check the **Export untranslated alarm messages only** check box if you have already exported, translated and re-imported your language files by Area and you want to export only newer, untranslated messages. Otherwise, leave the check box unchecked.
 - c In the **Language to export** pull-down list, select the language to export. You can export only one language at a time.
 - d In the **Path** box, type the folder to which you want to export the language file. Click **Browse** to select an existing folder or create a new folder.
- 4 Click **Export**. The export progress box appears.
- 5 Click **Close** when the export completes.

To export alarm comments for a selected Area or Areas

- 1 Using the ArchestrA IDE, open the Galaxy for which you want to export alarm comments.
- 2 Expand the **Application View** tree and select the Area(s) you want to export.
- 3 Right click to view the context menu. Select **Export** on the context menu, then select **Localization**, and then select **Alarm message(s) of selected Area(s)**.

As an alternate method to using the context menu, you can select the Area you wish to export, then use the Galaxy menu as in the preceding procedures.

- 4 Configure the export settings:
 - a In the **Select option to export all alarm messages** area, select the **Export selected area(s) only** radio button, or select the **Include sub-areas** radio button to include all sub areas for each selected Area.
 - b Check the **Export untranslated alarm messages only** check box if you have already exported, translated and re-imported your language files by Area and you want to export only newer, untranslated messages. Otherwise, leave the check box unchecked.
 - c In the **Language to export** pull-down list, select the language to export. You can export only one language at a time.
 - d In the **Path** box, type the folder to which you want to export the language file. Click **Browse** to select an existing folder or create a new folder.
- 5 Click **Export**. The export progress box appears.
- 6 Click **Close** when the export completes.

Translating Exported Alarm Comment Language Files

Alarm comments are exported to a .txt file, located in the selected directory. You can edit the files with a text editor, but we recommend opening and editing them in Microsoft Excel 2003 or Excel 2007.

Using the format-as-table feature in Excel greatly simplifies editing the specific text to be translated

To translate an exported alarm comment file

- 1 Open the .txt file in Excel. The Excel **Text Import Wizard** appears. Follow the wizard instructions to import the text file in tab delimited, general column data format. The file appears unformatted in Excel.

	A1	fx	Column1								
	A	B	C	D	E	F	G	H	I	J	K
1	Column1	Column2	Column3								
2	Unique Phrases:										
3											
4	Phraselid (IDefault Me Translated Message (EDIT THIS COLUMN)										
5	1	<<< NO C translated message in Germany for <<NO CONFI MESSAGE>>									
6	2	The UserDefined object provides a starting point for creating custom built									
7	objects that include Discrete and Analog Attributes, UDAs, Scripts,										
8	Extensions german translation for big alarm message										
9	3	ttttt	german translation for weird ttttt alarm message								
10	5	The Area represents a plant area and allows grouping of objects for									
11	modeling and alarm reporting."										
12	6	me.ShortDesc									
13											
14	Alarms:										
15											
16	Phraselid (IAlarm (DO NOT EDIT)										
17	6	Area_002.udbool1									
18	5	Area_002.udbool2									
19	3	UserDefined_001.abc									
20	2	UserDefined_001_001.RickAlarm									
21	1	WinPlatform_001.CheckpointFileCorruptionAlarm									
22											
23											

- 2 Format the text as a table in Excel:

- a Select cell A5.
- b Select the **Home tab**, then select **Format as Table**.
- c Select the color pattern you want and click **OK** in the **Format as Table** dialog box. Excel will format the region to be edited.

	A64	fx				
	A	B	C	D		
1	Column1	Column2	Column3			
2	Column1	Column2	Column3			
3	Unique Phrases:					
4						
5	Phraselid (DO NOT EDIT)	Default Message (DO NOT EDIT)	Translated Message (EDIT THIS COLUMN)			
6	1	<<< NO CONFIGURABLE MESSAGE >>>	translated message in Germany for <<NO CONFI MESSAGE>>			
7		The UserDefined object provides a starting point for creating custom built objects that include Discrete and Analog Attributes, UDAs, Scripts,				
8	2	Extensions, or Contained objects.	german translation for big alarm message			
9	3	ttttt	german translation for weird ttttt alarm message			
10		The Area represents a plant area and allows grouping of objects for				
11	5	modeling and alarm reporting.				
12	6	me.ShortDesc				
13						
14	Alarms:					
15						
16	Phraselid (DO NOT EDIT)	Alarm (DO NOT EDIT)				
17	6	Area_002.udbool1				
18	5	Area_002.udbool2				
19	3	UserDefined_001.abc				
20	2	UserDefined_001_001.RickAlarm				
21	1	WinPlatform_001.CheckpointFileCorruptionAlarm				

The Excel display is divided into four regions:

- Column 1 contains Phrase ID numbers used internally in the language file. Do not edit this column.
 - Column 2 contains the exported alarm messages. Do not edit this column.
 - Column 3 contains the alarm message translations. Edit this column with your translation text.
 - Below the alarm comment text columns are the alarms and phrase IDs, which map to column 1 and are used internally in the language file. Do not edit this area.
- 3 Translate the alarm comment text. Type the alarm comment translations into the appropriate rows in column 3.

Important Only edit the text in column 3, translations. No other text in the file may be edited. You can edit the original alarm comment text only within the ArcestrA IDE.

- 4 Save the file as a .txt file in the directory you originally selected for export.

Important Do not change the file name or the file type, otherwise it will not import correctly into the Galaxy.

Importing Translated Alarm Comment Language Files

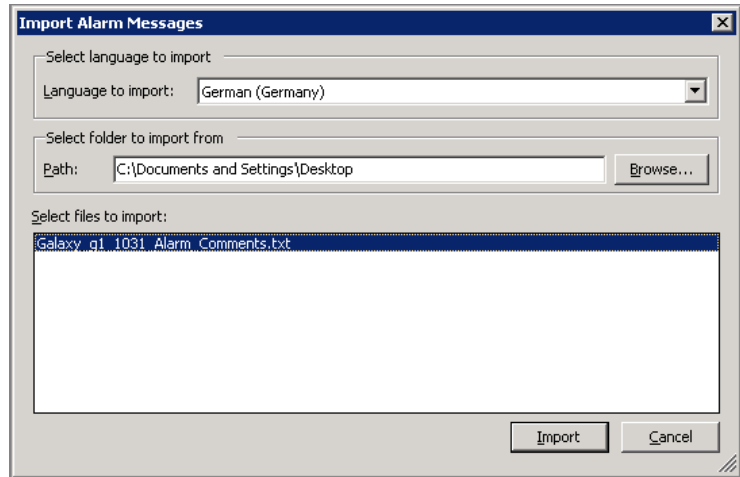
You can import translated alarm comment language files only from the Galaxy menu, not from the context menu for specific Areas.

To import alarm comment language files, you must first have exported alarm comments for translation of some or all of the alarm comments.

To import a translated alarm comment language file

- 1 Using the ArcestrA IDE, open the Galaxy for which you want to import translated alarm comments.

- 2 On the **Galaxy** menu, select **Import**, then select the **Localization** menu option, and then select the **Alarm Message(s)** menu option. The **Import Alarm Messages** dialog box appears.



- 3 Configure the import settings:
 - a In the **Language to import** pull-down list, select the language to import. You can import only one language at a time.
 - b In the **Path** box, type or browse to the folder where you previously exported your language file. The available language files appear in the **Select files to import** list.
 - c Select a file or files to import.
- 4 Click **Import**. The import progress box appears.
- 5 Click **Close** when the import completes.

Note If an alarm name or message do not match, the alarm comment import will display a message for each failed alarm message import.

Re-exporting Alarm Comments

After exporting and translating alarm comments, you can re-import them to ArcestrA to help maintain up-to-date alarm comment language files.

After adding new alarm comments that require translation or after modifying existing, already translated alarm comments, you must re-export the language file to update the translations.

Exporting New Untranslated Alarm Comments

In the **Export Alarm Messages** dialog box, select the **Export untranslated alarm messages only** check box. This creates a file named `_untranslated.txt`. For example, exporting untranslated alarm comments in Chinese for a Galaxy name “TestSystem” would create a file named `Galaxy_TestSystem_2052_Alarm_Comment_Untranslated.txt`.

Exporting Modified Existing Alarm Comments

You can make changes to alarm comments that you previously exported, translated and imported back into ArcestrA. You must re-export the alarm comment language file(s) to update the translations.

In the **Export Alarm Messages** dialog box, select the same language option and folder you selected for the previous export. When asked, confirm that you want to overwrite the existing file.

Testing the Language Switching Functionality at Run Time

We recommend that you test the run-time language switching functionality. Follow the procedures outlined in this chapter to enable run-time language switching in your application. Then import the appropriate language file into InTouch. The import step applies to both stand-alone and managed InTouch applications.

To test the language switching functionality

- 1 Open the application in WindowViewer.
- 2 On the **Special** menu, point to **Language**, and then click the name of the language to switch to.

The information from the corresponding translated dictionary file (if one exists and has been imported into InTouch) loads and appears.

- 3 When you are done, click **Close**.

Related Topics

Configuring Languages for a Galaxy

Chapter 12

Managing Galaxies

You can back up and restore Galaxies, change the Galaxy you are working with, delete a Galaxy, and export and import all or part of a Galaxy.

If you want to create a Galaxy, see [Creating a New Galaxy](#) on page 17.

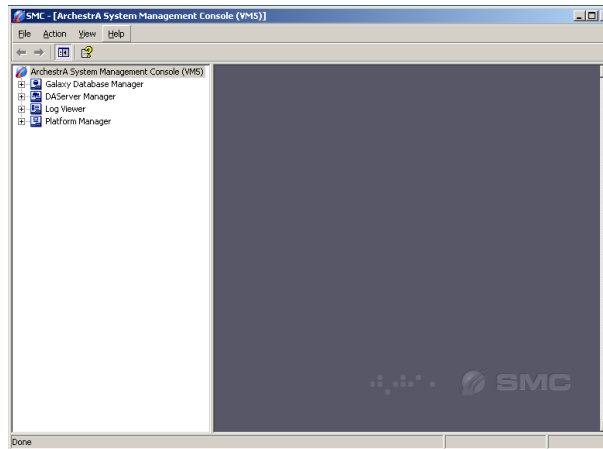
Backing Up and Restoring Galaxies

Periodically, you should back up your Galaxy. Backing up your Galaxy helps if you have a computer failure or other problem. If there is a failure, you can restore your Galaxy from the backup.

Use the Galaxy Database Manager to back up and restore your Galaxy. The Galaxy Database Manager is part of the suite of ArcestrA System Management Console utilities.

To open the Galaxy Database Manager

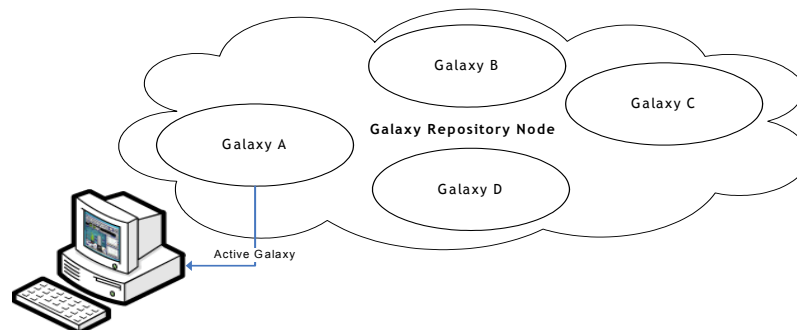
- ◆ Click **Start**, point to **Programs** and then to **Wonderware**, and then click **System Management Console**.



See the Galaxy Database Manager documentation for more information about backing up or restoring your Galaxy.

Changing Galaxies

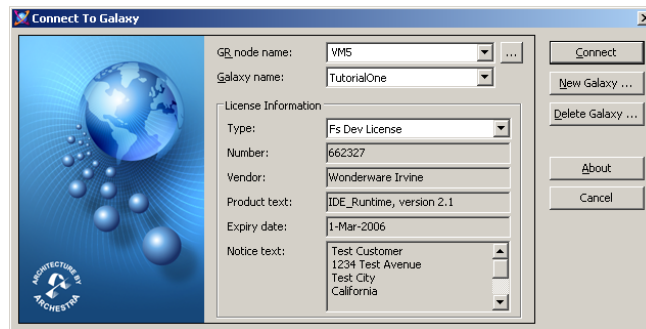
You can have many Galaxies in the Galaxy Repository. If you are a systems integrator, you have at least one Galaxy for every client.



For a Galaxy to deploy objects to other computers in the Galaxy, the GR must host a platform defined in that Galaxy. Because of this, you can only deploy one Galaxy from the Galaxy Repository at a time to the computers on your network. For more information about deploying, see [Deploying Objects](#) on page 139.

To change from one Galaxy to another

- 1 On the **Galaxy** menu, click **Change Galaxy**. The **Change Galaxy** dialog box appears.



- 2 Do one of the following:
 - Select another Galaxy from the **Galaxy Name** list.
 - Select another Galaxy Repository node from the **GR Node Name** list.
- 3 Click **Connect**.

Deleting a Galaxy

You can delete a Galaxy. Deleting a Galaxy removes all of the Galaxy information from your computer.

Before you delete a Galaxy, you must undeploy all objects within it. For more information about undeploying objects, see [Undeploying Objects](#) on page 144.

Make sure you select the right Galaxy to delete. After you delete a Galaxy, you cannot undelete it. You can only recreate it by restoring from a backup. For more information, see [Backing Up and Restoring Galaxies](#) on page 279.

To delete a Galaxy

- 1 Undeploy all objects from the Galaxy you want to delete.
- 2 Close any Galaxy and connected IDE you have open.
- 3 On the **File** menu, click **Change Galaxy**. The **Connect to a Galaxy** dialog box appears.
- 4 Select the Galaxy you want to delete, except the connected Galaxy.
- 5 Click **Delete Galaxy**.
- 6 At the prompt, click **Yes**.
- 7 When the Galaxy is deleted, click **Close**.

Exporting a Galaxy Dump File

You can export instances and their configuration data in a Galaxy to a comma-separated values file with a `.csv` file name extension. After you export the Galaxy to a `.csv` file, you can edit it with Microsoft Excel 2000 or later. This makes it very easy to do large editing changes, such as search and replace.

Exporting only exports instances. Templates cannot be exported in `.csv` file format.

The `.csv` file contains the configuration for the checked-in versions of the selected instances and the checked-out instances of the user who does the Galaxy dump. If an instance is checked out by another user when you export the Galaxy, the checked in version is exported.

The file contains only those attributes that are unlocked and configuration time-writable, one column per attribute. The following are not exported:

- Scripts libraries are not exported. Scripts within an object are exported.
- Attributes that are not text-based are not exported. For example, type `QualifiedStruct` is not exported.
- Custom object online help files are not exported.

See [Looking at the Galaxy Text Dump File Structure](#) on page 283 for specific information about the structure of the `.csv` file. Before you start, make sure you know what instances you want to export to a file.

To export objects to a Galaxy dump file

- 1 In the **Application views** area, select at least one instance. Shift+click to select multiple instances. You can export all instances derived from a template by selecting the template.
- 2 On the **Galaxy** menu, click **Export** and then click **Galaxy Dump**. The **Galaxy Dump** dialog box appears.
- 3 Browse to the location of the `.csv` file to which you want to dump the selected instances. Type the name of the file. Click **Save**.
- 4 When the Galaxy export process is done, click **Close**. A `.csv` file is created containing the selected objects and configuration data.

You can open this file in a text editor like Notepad or in Microsoft Excel 2000 or later. When you have finished editing the file, save it as plain text `.csv` (comma delimited) file to import back into the Galaxy.

Looking at the Galaxy Text Dump File Structure

The Galaxy .csv file has a specific structure. This section describes that structure.

Objects are organized in the .csv file based on the template each is derived from. A header row per template indicates the instance's columns' reference.

Other information is organized in columns. This makes it easy for you to read the information and carefully make any changes you need. You can easily import the .csv file into a text editor or into Microsoft Excel.

Row	A	B	C	D	E	F	G	H	I	J	K	L
1	Created on: 7/4/2005 12:09:48 PM from Galaxy: Splash123											
2												
3	:TEMPLATE=\$ADevAlarm											
4	:TagName	Area	SecurityGr	Container	Contained	ShortDesc	ExecutionF	ExecutionF	UDAs	Extensions	CmdData	Auto
5	ADevAlarm_001	Default				The Anal	None		<UDAlInfo>	<Extensio	<CmdData	FALSE
6												
7												
8	:TEMPLATE=\$ADiscAlarm											
9	:TagName	Area	SecurityGr	Container	Contained	ShortDesc	ExecutionF	ExecutionF	UDAs	Extensions	CmdData	Auto
10	ADiscAlarm_001	Default				The Switch	None		<UDAlInfo>	<Extensio	<CmdData	FALSE
11												
12												
13	:TEMPLATE=\$AROCArm											
14	:TagName	Area	SecurityGr	Container	Contained	ShortDesc	ExecutionF	ExecutionF	UDAs	Extensions	CmdData	Auto
15	AROCArm_001	Default				The Anal	None		<UDAlInfo>	<Extensio	<CmdData	FALSE
16												
17												
18	:TEMPLATE=\$AValueAlarm											
19	:TagName	Area	SecurityGr	Container	Contained	ShortDesc	ExecutionF	ExecutionF	UDAs	Extensions	CmdData	Auto
20	AValueAlarm_001	Default				The Anal	None		<UDAlInfo>	<Extensio	<CmdData	FALSE
21												
22												
23	:TEMPLATE=\$CBoolean											
24	:TagName	Area	SecurityGr	Container	Contained	ShortDesc	ExecutionF	ExecutionF	UDAs	Extensions	CmdData	UDABool
25	CBoolean_001	Default				The FieldR	None		<UDAlInfo>	<Extensio	<CmdData	FALSE

Add comments by adding a line with a semi-colon as the first character in the comment.

Host Attributes

Galaxy dump files contain a column for the Host attribute of the objects being dumped. In the case of Platform objects, Host is always the name of the Galaxy from which the object is being dumped.

This data is ignored in subsequent Galaxy Load operations because the Host for Platform objects is automatically the name of the Galaxy into which it is being loaded, regardless of the name of the Galaxy from which it was dumped.

Using a text editor, you can delete the Host attribute column, like any other data in the Galaxy dump file. This has no effect on Platform objects in subsequent Galaxy Load operations because they take the Galaxy name as their Host.

About Quotation Marks and Carriage Returns

Carriage returns in scripts associated with dumped objects are replaced with `\n` in the `.csv` file. If you edit the dump file, **do not** delete the `\n` characters. If you edit scripts in the dump file, use `\n` for a carriage return. This character set is interpreted as a carriage return when the dump file is used in a Galaxy Load operation.

When editing a script in a dump file, use `\\n` if you want to include the backslash character (`\`) followed by the letter `n` in a script. This character set is not converted to a carriage return in a Galaxy Load function.

Be careful when adding or editing quotation marks in the `.csv` file. Type all single quotation marks as two single quotation marks and surround the entire string with opening and closing quotation marks.

Make sure the string contains an even number of quotation marks. When the object is loaded in a Galaxy Load operation, the extra quotation marks are stripped from the string.

For example, if you want to enter `3"Pipe` as a Short Description, add a second quotation mark (`3""Pipe`) and then surrounding quotation marks (`"3""Pipe"`).

Time Formats in Excel

If you edit a Galaxy dump file in Microsoft Excel, be careful typing time entries. Excel can change the time format and the resulting entries do not work when you reload the changed Galaxy.

Galaxy Load accepts two formats:

- `DAYS HH:MM:SS:SSSSSS` - the number of `DAYS` is followed by a space.
- `HH:MM:SS:SSSSSS` - Excel automatically changes the entry to an incompatible format.

When you type time entries in Excel, use the following format:

`DAYS HH:MM:SS:SSSSSS`.

For example:

`0000 01:02:12.123:1`

`04:03:06.12:120`

`22:66:88:123456`

Importing a Galaxy Load File

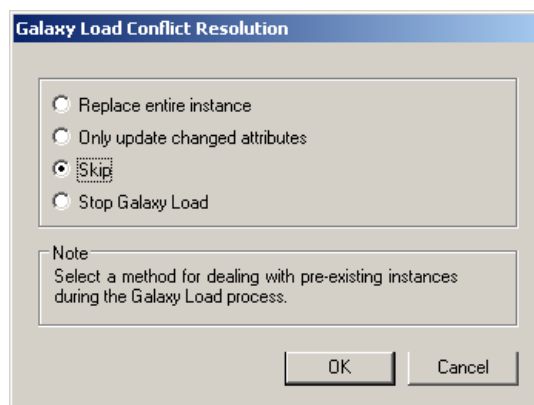
After you are done editing a .csv file, you can import it back into your Galaxy.

A load file contains only instances. Templates cannot be dumped or loaded. See Looking at the Galaxy Text Dump File Structure on page 283 for more information about the contents of the .csv file.

Note A comment line in a .csv file created in Microsoft Excel can create an unintended default-value object. To avoid this, open the .csv file in Notepad to make sure the comment line does not contain quotation marks.

To import a .csv file

- 1 On the **Galaxy** menu, click **Galaxy Load**. The **Galaxy Load** dialog box appears.
- 2 Browse to find the .csv file that contains the objects and configuration data you want to import. Select the file and click **Open**.
- 3 The **Galaxy Load Conflict Resolution** dialog box appears. Use it to resolve conflicts that can occur if objects you want to load already exist in the Galaxy.



- 4 Select one of the following:
 - **Replace entire instance** if an instance of an object with the same name already exists and you want to replace it entirely with the object in the import file.
 - **Only update changed attributes** if an instance with the same name already exists and you want to replace only the attributes of the object where the values are different.

- **Skip** if an instance with the same name already exists and you want to keep the version already in the Galaxy.
 - **Stop Galaxy Load** if an instance with the same name already exists and you want to cancel the Galaxy Load.
- 5 Click **OK**. A progress box appears showing the Galaxy load process.
- When the load is done, all objects changed or created during the Galaxy Load process are checked in.

Synchronizing Time across a Galaxy

Some of the Application Server functions like scripting, alarming, and historizing depend on all member computers of a Galaxy synchronized to the same time. A time master is a Network-Time-Protocol Server that provides a time that other nodes on your network can synchronize with.

The time master can be a non-ArchestrA node or one within the Galaxy. The ArchestrA nodes in the Galaxy periodically synchronize their clocks to the time master.

Using Time Synchronization in Windows Domains

The system administrator of the Windows 2000 domain may have time synchronization configured already. If this is the case, configuring a Galaxy Time Master is unnecessary and can conflict with the existing time synchronization.

Time synchronization in your Galaxy is critical if one or more nodes run the Windows XP operating system. Windows XP supports a network authentication protocol that requires time synchronization between two nodes to enable communication between them. This protocol fails the communication between the nodes if the time on the two nodes differs by a predetermined amount (for example, five minutes).

If a node in your Galaxy runs Windows XP operating system and its system time is beyond the allowed variance, ArchestrA operations, such as deployment, may fail.

Synchronization Schedule

The designated node clock serves as the master clock for all timestamping functions. Time synchronization is based on Microsoft's Windows Time Service. ArcestrA does not implement its own time synchronization algorithm.

The default synchronization period is one time every 45 minutes until three successful synchronizations occur. After three successful synchronizations, synchronization runs one time every eight hours.

All WinPlatforms begin synchronizing the time on their node when they are deployed.

You can specify a time master node in another time zone. The time on each Application Server node is set to the time specified on the node in the other time zone.

Required Software

If a time master or a time client node runs either Windows 2003 Server or Windows XP software, you must install a Microsoft hotfix on that computer before you can synchronize to a time master. Install the appropriate hotfix according to the following table.

Operating System	Hotfix
Windows XP	WindowsXP-KB823456-x86-ENU.exe
Windows 2003 Server	WindowsServer2003-KB823456-x86-ENU.exe

Contact Microsoft through its Product Support Services to get these hotfixes.

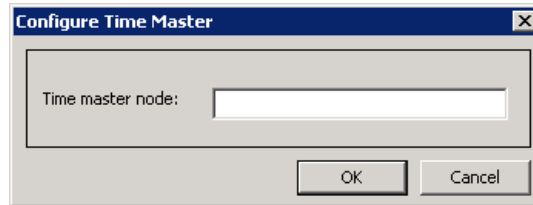
If your time master computer is a non-Galaxy node, regardless of operating system, you must also change certain Registry keys. Contact Wonderware Technical Support to obtain the files that can set those Registry keys.

Important You must complete the software and Registry updates before configuring a node as a time master.

Before you start, you need the fully qualified node name in the format of: `nodename.domain.organization`.

To configure a time master node

- 1 Apply all hotfixes and make the Registry changes as explained previously.
- 2 On the **Galaxy** menu, point to **Configure** and then click **Time Master**. The **Configure Time Master** dialog box appears.



- 3 Type the node name in the **Time Master Node** box.
- 4 Click **OK**.

Hosting Multiple Galaxies in One Galaxy Repository

You can create and configure multiple Galaxies in a single Galaxy Repository on the same computer. You can configure one Galaxy from an IDE and then change Galaxies and configure the second one using the same IDE.

Note If you try to deploy objects from the second Galaxy to a computer that hosts deployed objects from the first Galaxy, the deploy fails.

Managing Licensing Issues

Your license controls access to the Galaxy Repository. If a license-related message appears when you open the IDE, there is a problem with your license. The message appears as a result of one of the following conditions:

- A license is not installed.
- Your license expired.
- You exceeded the licensed I/O count or number of licensed WinPlatforms.
- The number of I/O or WinPlatforms specified in your IDE development license is more than the I/O or WinPlatforms specified in your Galaxy license.

Note If a license expires while you are using the IDE, you cannot connect to the Galaxy the next time you open the IDE.

Viewing License and End-User License Agreement Information

Use the License Utility to view information about your license and End-User License Agreement. Until any problems are resolved, you cannot:

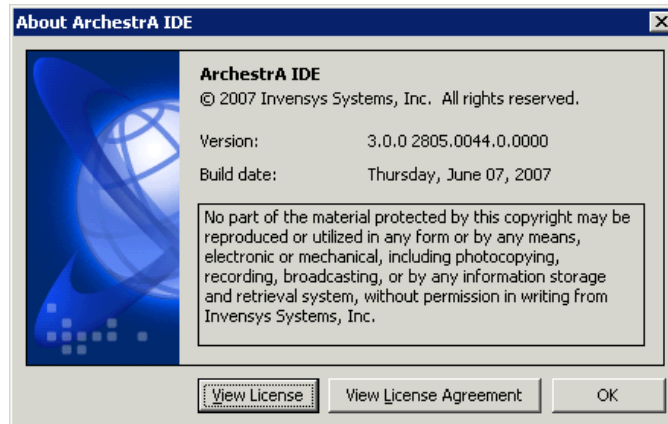
- Open the IDE
- Connect to existing Galaxies
- Create new Galaxies

After you update your license, you can connect to your Galaxy and open the IDE with no further problems. For more information about updating your license, see [Updating a License](#) on page 294.

To view ArchestrA IDE version information

- 1 On the **ArchestrA IDE** menu, click **Help**.

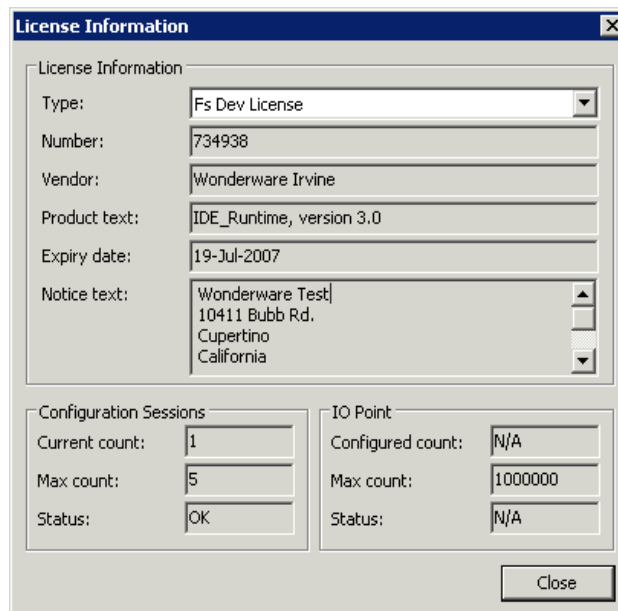
- Click **About ArchestrA IDE**. The **About ArchestrA IDE** dialog box appears, showing the copyright notice, version, and build date for your installed ArchestrA IDE application.



- Click **OK**.

To view your license information

- On the **About ArchestrA IDE** screen, click the **View License** icon. The **License Information** dialog box appears.



- In the **Type** list, select the license you want to view.
- In the **License Information** area, check your **Expiry date**.

4 Select **Fs Dev License** to view:

- In the **Configuration Sessions** area:

Current count	The number of IDE sessions currently open.
Max count	Maximum number of configured sessions allowed by your license.
Status	Relationship between the configured and maximum I/O point count values.

You see one of three states:

- OK: Everything is fine.
 - Exceeded: Configured I/O points count exceed the maximum allowed by your license.
 - DEV: Your license has no I/O and Platform Count feature line.
- In the **IO Point** area

Configured count	Number of configured I/O points in your Galaxy.
Max count	Maximum number of configured I/O points allowed by your license. If your license has no I/O and Platform Count feature line, N/A appears.
Status	Relationship between the configured and maximum I/O point count values.

You see one of three states:

- OK: Everything is fine.
- Exceeded: Configured I/O points count exceed the maximum allowed by your license.
- DEV: Your license has no I/O and Platform Count feature line.

5 Select **App Server License** to view:

- In the **Platform** area:

Current Count	Number of deployed WinPlatforms in your Galaxy.
Max Count	Maximum number of deployed WinPlatforms allowed by your license. If your license has no I/O and Platform Count feature line, N/A appears.
Status	Relationship between the current and maximum WinPlatform count values.

You see one of three states:

- **OK:** Everything is fine.
- **Exceeded:** Deployed WinPlatform count exceeds the maximum allowed by your license.
- **DEV:** Your license has no I/O and Platform Count feature line.

- In the **IO Point** area:

Configured Count	Number of configured I/O points in your Galaxy.
Max Count	Maximum number of configured I/O points allowed by your license. If your license has no I/O and Platform Count feature line, N/A appears.
Status	Relationship between the configured and maximum I/O point count values.

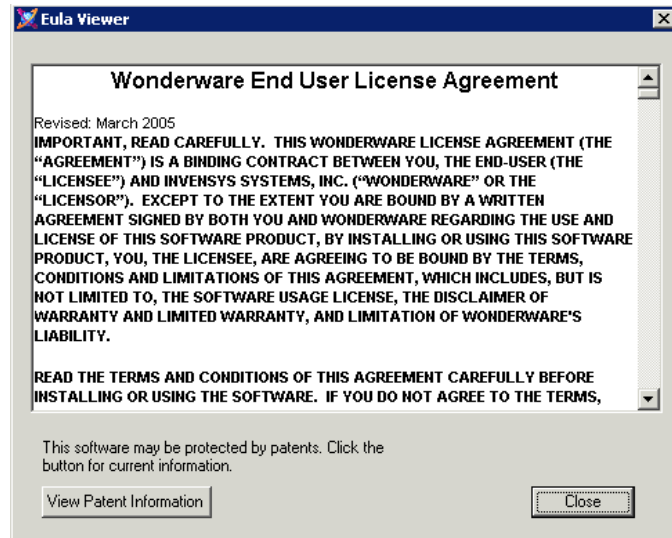
You see one of three states:

- **OK:** Everything is fine.
- **Exceeded:** Configured I/O points count exceed the maximum allowed by your license.
- **DEV:** Your license has no I/O and Platform Count feature line.

6 When you are done, click **Close**.

To view the end-user license agreement

- 1 On the **About ArchedrA IDE** dialog box, click **View License Agreement**. The **Eula Viewer** dialog box appears. Scroll down to read the terms and conditions of the agreement.



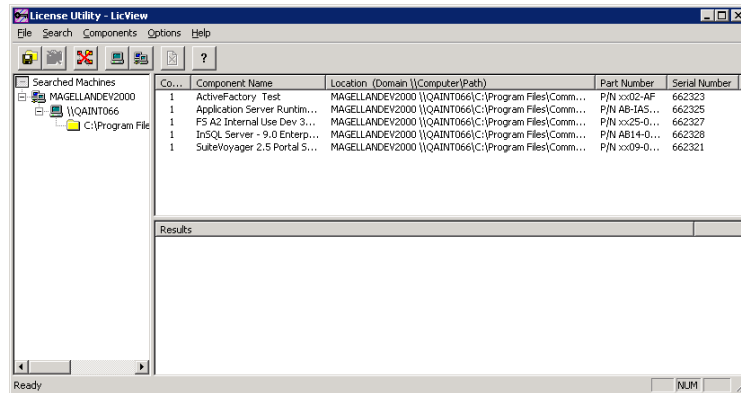
- 2 Click **View Patent Information** to read the patent information. You must have an Internet connection to view the patent information.
- 3 When you are done, click **Close**.

Updating a License

After you get your new license from Wonderware, you must install it to update your Galaxy Repository license. Follow the steps below.

To update your license

- 1 Start the **License Utility**. On the Windows **Start** menu, point to **Programs, Wonderware**, then to **Common**, and then click **License**. The **License Utility** appears.



- 2 On the **File** menu, click **Install License File**. Browse to the folder where you saved the `.lic` file.
- 3 Select the `.lic` file and click **Open**. The **Destination Computer for Installation** dialog box opens.
- 4 Do the following:
 - In the **Domain** box, type the name of the domain in which the computer resides.
 - In the **Computer** box, type the name of the computer on which you want to install the license file.
 - Click **OK**.

If a license file exists on that domain, the **Installing a License File** dialog box appears.

- To overwrite the license file, click **Overwrite**.
- To add the new license file information to the existing license, click **Append**.
- When you are done updating the license file, you can see the results of the installation in the **Results** pane.

Disk Space Requirements

After Application Server is installed, certain operations require at least 100 MB of available disk space. These operations include

- Creating a Galaxy
- Deploying objects
- Importing and exporting objects
- Loading and dumping a Galaxy
- Restoring and backing up a Galaxy.

This minimum requirement applies to the Galaxy node as well as any remote IDE nodes.

If your computer has less than 100 MB of available hard disk space, running any of these operations can result in erratic behavior.

Managing Communication between Galaxy Nodes

You can use Application Server in a distributed environment. All computers with ArcestrA-enabled software installed must be able to communicate with each other.

Two items must be considered to ensure communication between nodes:

- ArcestrA user accounts
- Multiple network interface cards in computers

All Platforms communicate with several attributes on the Galaxy Repository (GR) Platform to detect configuration changes. This communication sends NMX heartbeats from each Platform to the GR Platform. For example, the attributes include “time of last deploy,” and “time of last configuration change.”

In addition to NMX heartbeats, all Bootstraps on each Platform in the Galaxy send each other heartbeats. These heartbeats are for Platform status information, occur every two seconds, and are configurable.

About ArcestrA User Accounts

Communication occurs with an ArcestrA-specific user account set up during the initial installation of each ArcestrA component on each computer, including the IDE.

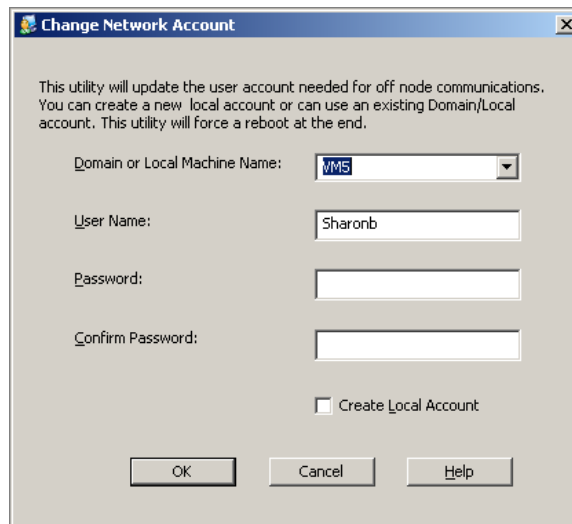
WARNING! The user account that is used for ArcestrA communication is a standard Windows operating system account located on the local computer or on a domain. Do not delete this account with operating system account management tools. If you do, the IDE stops functioning.

If you delete the ArcestrA user account on an IDE node, you must recreate it with the Change Network Account utility. You must have Administrator privileges on the computer to make changes with the Change Network Account utility.

Before you start, find out the user name and password that is created on all computers with ArcestrA-enabled software installed.

To recreate an ArcestrA user account

- 1 Start the **Change Network Account** program by clicking **Start**, point to **Programs**, and click **Wonderware**. Click **Common** and then click **Change Network Account**.



- 2 Do one of the following:
 - In a single-node ArcestrA system, create any account.
 - In a multi-node ArcestrA system, create the same user account with the same user name and password on all other ArcestrA computers.

- 3 Click **OK**. After you recreate the user account, the Microsoft Windows security component on your computer can take several minutes to update this information on the ArcestrA Galaxy node. Until that happens, your IDE might not function properly. Restarting the Galaxy node applies this update immediately.

Using Multiple Network Interface Cards

You can use nodes with more than one network interface card (NIC). These nodes must be configured properly so they can communicate with other ArcestrA nodes.

If a node contains multiple NICs for redundancy reasons, see [Working with Redundancy](#) on page 303 for more information.

If a multiple NIC computer in your Galaxy uses only one NIC, you need to disable all cards except the supervisory network.

Defining the Order of the NIC

For any multiple NIC ArcestrA node to communicate with all other Galaxy nodes, you must define the correct order of network connections in the network services of the computer.

Before you start configuring multiple network cards, you need the IP address for the second and further nodes.

To define the correct order

- 1 Open the **Network and Dial-up Connections** dialog box and rename each card with a clearly identifiable function, for example, `Supervisory Net` and `PLC net`.

Opening the **Network and Dial-up Connections** dialog box varies, depending on the version of Windows you are using.

- 2 Order the network cards properly. On the **Advanced** menu, click **Advanced Settings**. In the **Advanced Settings** dialog box, click the up and down arrows to define the correct order of Connections.

The first connection in the list must be the supervisory network card. If a computer contains more than two network cards, for example, a supervisory connection, a PLC connection, and a Redundancy Message Channel (RMC) connection for ArcestrA redundancy, the supervisory net must be listed first and the others can be listed in any other position.

- 3 Click **OK**.

Configuring the IP Address and DNS Settings

After you specify the order of the network cards, you are ready to configure several other parameters to ensure successful node-to-node communications in the ArcestrA environment.

You must configure the IP address and DNS settings as follows for each network card to function properly.

To configure the IP address and DNS settings

- 1 In the **Network and Dial-up Connections** dialog box, right-click the network connection and click **Properties** in the shortcut menu. The **Properties** dialog box for this connection appears.
- 2 In the list of components used by this connection, select **Internet Protocol (TCP/IP)** and click **Properties**. The **Internet Protocol (TCP/IP) Properties** dialog box appears.
- 3 For the supervisory network, select **Obtain an IP address automatically**.

For the other network connections, select **Use the following IP address**. Type the correct IP address for the secondary and further cards. See your network administrator for the proper settings for the remainder of the parameters in this group.

Important If you are using the Vista or Windows Server 2008 operating systems, you must configure the NICs in a certain way. For more information, see *Configuring Multiple NICs for the Vista and Windows Server 2008 Operating Systems* on page 299.

- 4 Click **Advanced**. The **Advanced TCP/IP Settings** dialog box appears. Click the **DNS** tab.
- 5 For the supervisory network, select **Register this connection's addresses in DNS**.
For the other network connections, clear this check box.
- 6 Click **OK**.

Configuring Multiple NICs for the Vista and Windows Server 2008 Operating Systems

The Vista and Windows Server 2008 operating systems identify and assign network profiles during computer startup and each time a connection changes. The three profiles currently in use in these Microsoft operating systems are:

Domain profile: Active only when the computer can authenticate with a domain controller on all active interfaces (LAN, wireless, VPN, etcetera) The domain profile may be more or less restrictive than the other two profiles depending on network security policies.

Private profile: Active whenever the network type for all active network connections on the computer are identified as private networks. The private profile typically is used in a more trusted environment and is less restrictive than the public profile to allow for network discovery.

Public profile: Active in all other circumstances. The public profile typically is more restrictive than the private profile because the computer often is connected to the Internet in an insecure location. Network discovery and remote access are disabled rather than explicitly blocking specific traffic.

For the Vista and Windows Server 2008 operating systems, all firewall exceptions are turned off if one or more network interface cards (NICs) are configured as "Public Network."

Also, when the IP address of a network card changes, Vista and Windows Server 2008 automatically update the NIC to "Public Network." If a computer has two or more NICs, even if some of the NICs are configured as "Private Network", if any of the NICs are configured as "Public Network", the firewall exceptions will be disabled.

This is particularly important for computers configured to run as a redundant pair. If the TCP/IP properties for a network card are set to obtain an IP address automatically, there is a possibility that the network address will change when the computer restarts. The computer may suddenly change from having a "Private Network" to having a "Public Network," and potentially cause the firewall exceptions to be turned off again.

Note An exception to this would occur if the NICs are configured and set to the Public Profile *before* Application Server is installed. The Application Server installation would create the necessary firewall exceptions in the current (Public) profile.

The following describes the key NIC settings for a redundant pair on the Vista operating system. You configure these settings for both the primary and backup computers.

These settings apply if more than two NICs are installed for redundancy (RMC) or other networking purposes, such as networking PLCs.

For detailed instructions on how to configure multiple network interface card binding order settings, see *Defining the Order of the NIC* on page 297 and *Configuring the IP Address and DNS Settings* on page 298.

To configure multiple NICs for the Vista and Windows Server 2008 operating systems

- 1** Select the **Use the following IP address** option and provide an IP address, subnet mask, and default gateway.
 - a** For the primary computer, the default gateway should be the IP address that you configure for the backup computer. For example:
IP address: 100.100.100.94
Subnet mask: 255.255.255.0
Default gateway: 100.100.100.95
 - b** For the backup computer the default gateway should be the IP address that you configure for the primary computer. For example:
IP address: 100.100.100.95
Subnet mask: 255.255.255.0
Default gateway: 100.100.100.94
 - c** A NIC acting as Redundancy Message Control (RMC) does not require a default gateway. The IP address and Subnet mask can be configured as described for the primary and backup computers. Vista and Windows Server 2008 will identify this NIC and assign it a private profile. For example:
IP address 100.100.100.96
Subnet mask: 255.255.255.0
- 2** Make sure that the **Register this connection's address in DNS** option is not selected.
- 3** Make sure that the primary network adapter is first in the binding order. This places it above the RMC network in the binding order.
- 4** Make sure that ALL network cards are configured as "Private Network."

- 5 Configure the RMC NIC to have a persistent (across reboots) Private profile.
 - a Click **Start**, then click **Run**. Enter the command `secpol.msc`. The **Security Settings** window appears.
 - b Select **Network List Manager Policies**.
 - c Select **Unidentified Network**.
 - d Right click, then select **Properties**, and then change the Location type from **Not configured** to **Private**.
 - e Exit the menu and return to Windows.

It may be necessary to restart the computer for the changes in steps 4 and 5 to take effect.

If the two computers are set up properly, you can install the Wonderware software and the OSConfigurationUtility sets up the firewall exceptions appropriately.

If you install the Wonderware software before configuring the NICs properly, configure the NICs as described in this section and then run the OSConfigurationUtility again. To do this, run the OSConfigurationUtility.exe, located in the “C:\Program Files\Common Files\ArchestrA” folder.

To verify that the firewall exceptions are set, open Control Panel and then open the Windows Firewall application. Click the **Exceptions** tab. The list of exceptions should include the executables for the Wonderware products you have installed.

Chapter 13

Working with Redundancy

You can set up and run Application Server in a redundant environment. In Application Server, two types of redundancy ensure continued run-time operation. You can configure redundant:

- AppEngine object pairings for computer or software failures.
- Data acquisition communications to one or more PLCs.

A failover is the condition during which run-time operations are moved from one critical component to another. Failover can occur due to failure conditions or it can be forced manually, called a forced failover.

For more information about redundancy, see the Wonderware Developer Network.

About Redundancy

Application Server provides redundancy in two critical functions:

- AppEngine

You must configure both an AppEngine and two WinPlatforms.

- Data acquisition

You must configure two DIObjects (data sources) and a RedundantDIObject.

Configuring AppEngine Redundancy

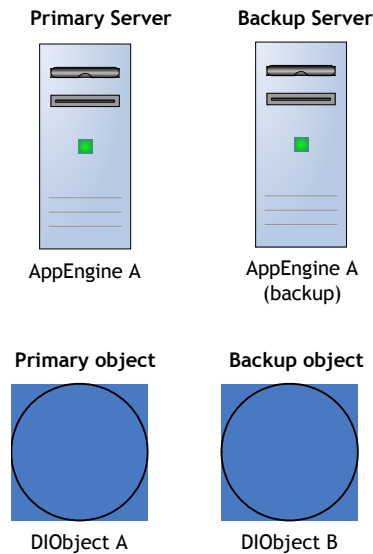
The Primary/Backup AppEngines form a redundancy pair. The ArcestrA infrastructure will generate the backup AppEngine automatically when redundancy is enabled for an AppEngine. Hierarchy of ApplicationObjects can only be assigned to the primary AppEngine. The primary and backup engines need to be assigned to redundancy enabled platforms, and they can be deployed separately.

For data acquisition, the Primary/Backup DIObjects (the data sources) must be separately created, configured, and deployed. Also, you must create, configure, and deploy a RedundantDIObject to control failovers between the two data source objects.

In a redundant system, install and configure the primary OPC server on the backup engine node.

Important For AppEngine redundancy, ArcestrA supports a one-to-one topology in which the computers hosting the Primary and Backup object sets must be connected by crossover cable and have fixed IP addresses.

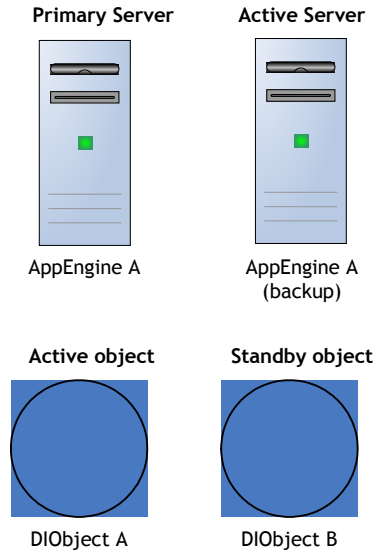
When you configure redundancy, you configure the Primary object and the Backup object.



- **Primary object:** The main or central object that provides the functionality during run time. For AppEngines, this is the object you enable for redundancy. For data acquisition, this is the DIOject you intend to use first as your data source in run time.
- **Backup object:** The object that provides the functionality of the Primary object when the Primary object fails. For AppEngines, this is the object created by the ArchestrA infrastructure when the Primary object is enabled for redundancy. For data acquisition, this is the DIOject you do not intend to use first as your data source in the run-time.

Redundancy during Run Time

When you deploy these objects to a run-time environment, the configuration objects become the Active object and the Standby object.



- **Active object:** The object that is currently executing functions. For AppEngines, it is the object that is hosting and executing ApplicationObjects. For data acquisition, it is the object that is providing field device data through the RedundantDIOBJECT.
- **Standby object:** The object that is waiting for a failure in the Active object or for a force-failover. For AppEngines, it is the object that monitors the status of the Active AppEngine. For data acquisition, it is the object that is not providing field device data through the RedundantDIOBJECT.

In the AppEngine redundancy environment, the Active and Standby objects monitor each other's status and switch when failure conditions occur.

In the data acquisition environment, the RedundantDIOBJECT monitors the status of the two DIOBJECT data sources, and handles the switching from Active to Standby objects.

The relationship between the configuration time (Primary/Backup) and run-time (Active/Standby) object pairs is not static. In the run time, either the Primary or Backup object can be the Active object at any particular time. Whenever one becomes the Active object, the other automatically becomes the Standby.

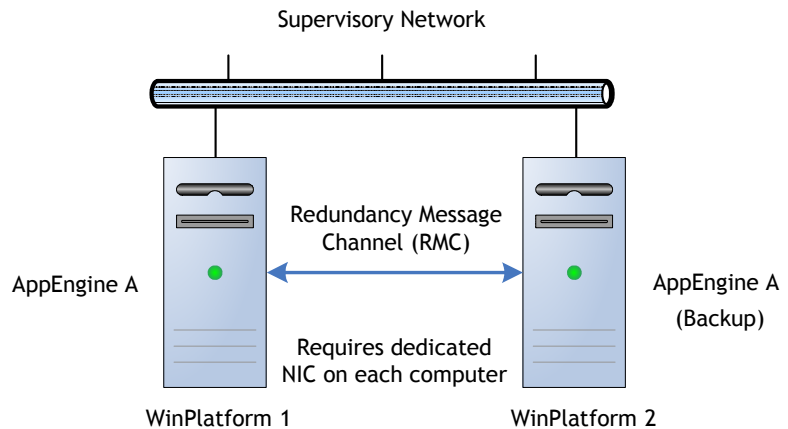
Working with AppEngine Redundancy

You enable AppEngine redundancy in the Primary AppEngine. You must also configure two WinPlatforms for redundancy, one to host the Primary AppEngine and one to host the Backup AppEngine.

Important WinPlatforms hosting redundancy-enabled AppEngines must be deployed to computers running the same operating system.

The configuration of both WinPlatforms should be the same. At a minimum, the store-forward directory configurations must be common to both WinPlatforms:

During configuration, you can assign Primary and Backup AppEngines to the same WinPlatform. To deploy, you must assign the Primary and Backup AppEngines to different WinPlatforms.



Each production computer hosting a redundancy-enabled AppEngine must have a minimum of two network cards. One NIC is for the supervisory network and PLC network, if the computer has only two network cards. The other must be for a dedicated Ethernet connection between computers for the redundancy message channel (RMC).

For information about distributed networks, see [Using Multiple Network Interface Cards](#) on page 297.

The RMC handles redundancy monitoring, message handling and data synchronization between redundant pairs.

Configuring the Redundancy Message Channel

For the redundant pair of AppEngines to successfully communicate with each other, you must define the correct order of network connections in the network services of each computer.

We recommend that you name each card with a clearly identifiable function (for example, “Supervisory Net” and “Redundant Message Channel”).

You must configure at least the following parameters:

- Redundancy Message Channel IP Address
- Redundancy Message Channel Port
- Primary Message Channel

To configure network cards

- 1 Open the **Network Connections** dialog box in Windows. The specifics about opening this dialog box varies, depending on the version of Windows you are working on.
- 2 Click **Advanced Settings**. If the computer name is used in the platform's "node name" box, The first connection in the list must be the supervisory network card. Use the up and down arrows to define the correct order.
- 3 **Connections**.
If a computer contains more than two network cards, the supervisory net must be listed first and the RMC connection can be listed in any other position. For example, your computer might have a supervisory connection, a field device connection, and a RMC connection.
- 4 Configure the DNS settings for the supervisory network card to function properly.
 - On the **DNS** page of the **Advanced TCP/IP Settings** dialog box, select the **Register this connection's addresses in DNS** check box.
 - For the RMC network card to function properly, clear the **Register this connection's addresses in DNS** check box. For more details on these settings, see Using Multiple Network Interface Cards on page 297.
- 5 In Application Server, open the WinPlatform in the Object Editor for the computer you just configured.
- 6 Change the **Redundancy Message Channel IP Address** to the IP address of the RMC connection. See the WinPlatform Help for more information about these parameters.
- 7 Save and close.

Configuring Redundancy

You configure redundancy in AppEngine/WinPlatforms objects using their Object Editors.

The redundancy-related parameters in the AppEngine Object Editor are located on the **Redundancy** and **General** tabs.

An AppEngine that is part of a redundancy pair has a deployment status indicating its own status and that of its partner object. These statuses are visually indicated in the **Application** views.

You can set the status of a redundancy pair to one of the following states:

Pair Deployed	Both Primary and Backup AppEngines are deployed.
Pair Undeployed	Both Primary and Backup AppEngines are undeployed.
Partial Deployed	Either the Primary or Backup AppEngine is deployed and its partner is not deployed. If an AppEngine has a Partial Deployed status, its partner has a Partial Undeployed status.
Partial Undeployed	Either the Primary or Backup AppEngine is undeployed and its partner is deployed. If an AppEngine has a Partial Undeployed status, its partner has a Partial Deployed status.

To create AppEngine redundancy



- 1 In the Primary AppEngine, select **Enable Redundancy** on the **Redundancy** tab.
- 2 Configure the remaining redundancy parameters as needed. See the help file for the AppEngine for specific information about these parameters.
- 3 On the **General** tab, set the **Engine Failure Timeout** option to 2000 milliseconds. You may need to tune this parameter between 2000 ms and the default 10000 ms, depending on the requirements of your application.

Note The actual engine failure time-out is three times the value of this parameter. If you set the parameter to 2000 ms (2 seconds), a failover occurs if the AppEngine fails to communicate with the computer's Bootstrap for 6 seconds. A setting of 10000 ms (10 seconds) can be too long a wait period (30 seconds) for a well-functioning redundancy operation.

- 4 On the **General** tab, clear the **Restart the engine when it fails** check box.

Important If you enable redundancy in an AppEngine, do not select the **Restart the engine when it fails** option on the General page of the AppEngine's editor.

After you save the configuration of the object and check it into the Galaxy, the icon for the object changes. A Backup AppEngine is created with the same configuration as the Primary object.

Icon	Object
	Primary AppEngine
	Backup AppEngine

Configuring Redundancy in Templates

You can create a redundancy-enabled AppEngine template. When you create an instance of the template, both the Primary and Backup instances are created.

The Backup AppEngine is hosted by the Unassigned Host or the default WinPlatform if you specified one in the **Configure User Information** dialog box.

If you change the configuration and check in the Primary AppEngine, the Backup AppEngine:

- Is checked out in the background
- Updates the configuration
- Is checked in without notification to connected clients

Deleting Redundant AppEngines

You can disable redundancy in an AppEngine after the ArcestrA infrastructure creates the Backup object. If you disable redundancy and check in the Primary AppEngine, the Backup is deleted from the Galaxy. The exception is when the Backup is already deployed. In that case the newly configured Primary object cannot be checked in.

To delete a Backup AppEngine from the Galaxy, you must undeploy it first.

There is no redundancy-related configuration required on ApplicationObjects hosted by an AppEngine configured for redundancy. When a system failure occurs, the ApplicationObjects and their attribute values are duplicated on the Standby AppEngine, which becomes the Active AppEngine. For more about failover functionality, see *During Deployment* on page 314.

If the Platform hosting the redundant AppEngine is shutdown in SMC, the AppEngine cannot be flagged as “On failure mark as undeployed” when you undeploy the AppEngine. You see an engine communication error in the **Deploy** dialog box.

Deploying AppEngine Objects

Primary and Backup AppEngines can be deployed together or individually.

- When they are deployed together, regardless of which object is actually selected for deployment, the Primary always becomes the Active and the Backup becomes the Standby.
- When they are deployed individually, the first one deployed becomes the Active.

Hosted ApplicationObjects are always deployed to the Active AppEngine. When deploying the first of a redundant pair of AppEngines, you can cascade deploy all objects it hosts. This operation can be paired with deploying both the Primary and Backup AppEngines at the same time.

Important If you deploy the Backup AppEngine first and then deploy hosted objects to that AppEngine, make sure the network communication to both target computers is good before deploying the Primary AppEngine. Otherwise, errors occur.

In the run-time environment, either the Primary or Backup AppEngine can become the Active or Standby depending upon failure conditions on either computer.

Configuration Requirements

Before deploying the Primary and Backup AppEngines, all configuration requirements must be met.

- Each AppEngine must be assigned to a separate WinPlatform.
- A valid redundancy message channel (RMC) must be configured for each WinPlatform.
- To deploy the Primary and Backup together, select **Include Redundant Partner** in the **Deploy** dialog box. This option is not available when doing the following operations:
 - Cascade deploy from the Galaxy
 - Multiple object selection deploy
 - Deploying the WinPlatform that hosts the Primary or Backup AppEngine

The following table shows a matrix of allowed operations based on specific conditions.

Condition	Deploy Both Primary and Backup Objects	Cascade Deploy Allowed
Backup AppEngine's host WinPlatform configured for failover and deployed	Yes	Yes
Backup AppEngine in error state	Yes	Yes
Backup AppEngine's host WinPlatform not deployed	No	Yes
Backup AppEngine's host WinPlatform not configured for failover and deployed	Yes	Yes
Deploy from Galaxy node	No	Yes
Deploy from WinPlatform hosting Primary AppEngine	No	Yes
Multiple object selection deploy	No	No

Condition	Deploy Both Primary and Backup Objects	Cascade Deploy Allowed
Backup AppEngine's host WinPlatform not configured for failover and not deployed	No	Yes
Deploy from Backup AppEngine	Yes	Yes
Deploy from Primary AppEngine	Yes	Yes

Undeploying AppEngine Objects

Undeploying redundant pairs of AppEngines is just like undeploying any regular object.

You can undeploy the Active and Backup AppEngines separately or as a pair.

Important Undeploying any objects, including redundant AppEngine pairs, does not uninstall code modules for that object from the hosting computer. Code modules are uninstalled only when the WinPlatform is undeployed.

To undeploy as a pair

- 1 Select one of the objects in an Application view.
- 2 On the **Object** menu, click **Undeploy**. The **Undeploy** dialog box appears.
- 3 Select **Include Redundant Partner** in the **Undeploy** dialog box and click **OK**.

During Deployment

During initial deployment of a redundant pair of AppEngines, files are deployed in the following order:

- Code modules and other files for the Primary AppEngine are deployed
- Those files for its assigned ApplicationObjects
- All of these files are deployed to the Standby AppEngine by the Active engine's WinPlatform using the redundancy message channel (RMC)

Note If some or all of these files already exist on the Standby AppEngine's WinPlatform, perhaps, assigned to another AppEngine on that platform, only the delta files are deployed to the Standby AppEngine.

Objects at Run Time

Objects are always assigned to the Primary AppEngine in the configuration environment.

During run time, objects are always deployed to the Active AppEngine whether or not it was initially configured as the Primary object.

All files are deployed by the Active AppEngine's WinPlatform to the Backup AppEngine as described above.

During Run Time

In the run-time environment, the Active and Standby AppEngines first attempt to establish communication across the RMC. This occurs when an AppEngine belonging to a redundant pair first starts. Therefore, if one AppEngine is relocated later to a different WinPlatform, this communication between AppEngines can be reestablished.

During run time, the Active and Standby engines communicate with each other and monitor each other's status.

In the case of a hardware or software failure on the Active computer, the Standby AppEngine becomes the Active one. To move the new Standby AppEngine from its hosting computer, undeploy this AppEngine by selecting the **On failure mark as undeployed** option on the **Undeploy** dialog box. Reassign and redeploy it to a WinPlatform that is configured for redundancy on another computer.

AppEngine Redundancy States

Redundant pairs of AppEngines can have one of the following states at a time:

- **Active:** The state of an AppEngine when it has communication with its partner object, its partner is in Standby-Not Ready, Standby-Sync'ing with Active, or Standby-Ready state. A Standby AppEngine transitions into this state when a failover condition is detected. In this state, an AppEngine schedules and runs deployed objects, sends checkpoint data and sends subscriber list updates to the Standby AppEngine.
- **Active - Standby not Available:** The state of an Active AppEngine when it determines it cannot achieve communications with its partner object. This could mean that checkpoint, subscription and alarm state changes are not successfully transmitted to the Standby object because the partner AppEngine is not deployed, number of heartbeats missed from standby objects exceeds the configured max consecutive number of heartbeats missed, or notification is received that the Standby AppEngine shutdown or is not running. If an AppEngine is in this state, it 1) continues normal execution of hosted objects, 2) cannot be manually switched to Standby state, and 3) while continuing to attempt communicate with the Standby, does not attempt to send data to the Standby object.
- **Determining Failover Status:** The initial state of a redundancy-enabled AppEngine when it is first started. It has not determined yet whether it is the Active or Standby AppEngine. Communication between the two AppEngines is attempted first over the RMC and then over the primary network to make this determination. If communication cannot be made after a certain time-out period, an AppEngine assumes the Active role if it has all of the code modules and checkpoint file data to do so. Continued attempts are made at communicating with its partner.

- **Standby - Missed Heartbeats:** The state of an AppEngine when 1) the heartbeat pings are not received from its Active partner through the RMC, but the number of missed beats haven't reached the maximum consecutive number of heartbeats missed, or 2) the hearbeats from active engine have been missed through the primary channel. When in this state, the Standby object attempts to determine whether or not the Active object failed. If a manual failover is started by using the ForceFailoverCmd attribute, it is processed only if the heartbeats were missed over the primary network and not missed over the RMC.
- **Standby - Not Ready:** The state of an AppEngine when one of several conditions occurs: 1) its lost communications with its partner object or it maintains communications with its partner but missed checkpoint updates or alarm state changes from the Active AppEngine, 2) new objects are deployed to the Active AppEngine and necessary files are not installed on the Standby AppEngine yet, or 3) the Standby AppEngine lost communications over the RMC before it completed synchronizing data. Typically, the AppEngine's partner is in one of the following states: Active-Standby not Available, or Active.
- **Standby - Ready:** The state of an AppEngine when it completed synchronizing code modules and checkpoint data with the Active AppEngine. In this state, the AppEngine monitors for Active AppEngine failure by verifying heartbeat pings received from the Active engine and checks that all files required for execution are in sync with the Active engine. It receives the following from the Active AppEngine: checkpoint change data, subscription-related notifications, alarm state changes, and history blocks.
- **Standby - Sync'ng with Active:** The state of an AppEngine when it is synchronizing code modules with the Active object. If code modules exist on the Standby computer that do not exist on the Active node, they are uninstalled, and likewise, any code modules that exist on the Active node but not on the Standby node are installed. After all code modules are synchronized, the AppEngine transitions to Standby-Sync'd Code state.

- **Standby - Syncing Code:** The state of a Standby AppEngine that successfully synchronized all code modules with the Active object.
- **Standby - Syncing Data:** The state of a Standby AppEngine when all object-related data, including checkpoint and subscriber information, are synchronized with the Active object. An object in this state typically transitions to Standby-Ready state.
- **Switching to Active:** A transitional state when a Standby AppEngine is commanded to become Active.
- **Switching to Standby:** A transitional state when an Active AppEngine is commanded to become Standby. When the active engine is switched to standby, the engine will be restarted. This transition state can be switched off by the user changing the attribute of the engine.
- **Failed:** The state of a redundant partner when its process crashes or is terminated by the user. The AppEngine process can be restarted using System Management Console/PlatformManager.
- **Unknown:** The state of a redundant partner when a communication loss occurs between AppEngines or when the partner AppEngine is stopped, shutdown, or undeployed.

For examples on redundant configuration, see the Wonderware Developer Network.

Troubleshooting

Most troubleshooting happens in the System Management Console. For more information about using the System Management Console, see the *System Management Console User's Guide*.

Certain requirements are validated by the system infrastructure. For example, the order in which you configure an object pair for redundancy is validated.

The following problems can occur when you are using redundancy:

- You can configure an AppEngine for redundancy before configuring its associated WinPlatform. If you do this, you see an error message that the Platform (specifically, the RMC) is not configured yet.
- If the **RMC IP Address** parameter is not configured in both hosting WinPlatforms, then the configuration state of both Primary and Backup AppEngines changes to Error. You also see a message indicating that the host WinPlatform is not configured with the network adapter required for redundant communications. When the RMC IP Address is configured and the WinPlatforms are checked in, the hosted AppEngines are automatically revalidated and the Error state is resolved. If hosted AppEngines are checked out, they are not revalidated.
- If both Primary and Backup AppEngines are assigned to the same WinPlatform and you try to deploy both engines, both the Primary and Backup fail to deploy. You see a message that the Primary and Backup objects must be hosted by different WinPlatforms. Reassign the Backup object to another WinPlatform and deploy it separately.
- If both the **Network Address** and **RMC IP Address** parameters in the WinPlatform's editor refer to the same network card, you get a warning message when you save the configuration. These parameters must refer to different network cards.
- Before restarting a computer that hosts one of a redundant pair of AppEngines (either the Active or Backup), ensure that the Primary Network is connected. A restart while the Primary Network is disconnected makes the Primary Network bind to the RMC's IP address. An incorrect redundancy state occurs, indicating that redundancy functionality is good. Any time you restart a redundancy-enabled computer, check for proper network connections afterwards. For more information, see Using Multiple Network Interface Cards on page 297.

Generating Alarms

When failover conditions occur, the ArcestrA system reports alarms to subscribed alarm clients like InTouch.

These alarms contain the following information:

- The name of the AppEngine reporting the alarm.
- The node name of the AppEngine reporting the alarm.
- The state of the AppEngine.
- The node name of the AppEngine’s partner object.

Depending on what caused the failover, the Standby AppEngine can become the Active AppEngine in an Off scan state and alarms might not be generated.

If the Active AppEngine is shutdown off scan, the checkpointer can transfer that state to the Standby. When the Standby becomes the Active, it starts off scan. When the AppEngine is put on scan, alarms then are generated.

Reported alarms include the following:

Alarm	Previous State	Current State	Alarm Raised When	Alarm Cleared When	Alarm Reported By
Standby Not Ready *	Active	Standby Not Ready	Standby Not Ready	Entering Standby Ready	Active Engine
Standby Not Available	Active	Active Standby Not Available	Active Standby Not Available	Entering Active	Active Engine
Failover Occurred			Standby becomes Active	During the next scan of the Active engine	Active Engine

* The Active AppEngine monitors the status of the Standby through the RMC to determine when to raise this alarm. Also, if the Active AppEngine is in Active-Standby not Available state, this alarm is not generated.

When a failover occurs, the Standby AppEngine that becomes active carries alarms outstanding from the old Active AppEngine. The state of those old alarms, though, will change to reflect the new partner's status.

Timestamps are preserved for alarms, including when:

- The alarm was acknowledged
- The alarm condition went true
- The alarm condition went false

Finally, the following information is preserved for alarms:

- An alarm was acknowledged
- The message input by the operator when the alarm was acknowledged

All alarm state information is collected and sent to the Standby AppEngine at the end of a scan cycle and before being sent to alarm clients.

The sequence of reporting alarms ensures that alarm clients do not report alarms in states that are different from those reported by the Standby AppEngine if the Active AppEngine fails.

Generating History

All active objects (AppEngine and hosted objects) report history data as they normally do in the run-time environment.

Historical data is reported to the historian only from the Active AppEngine.

Losing connectivity with the historian does not cause a failover. The Active AppEngine goes into store-forward mode and caches data every 30 seconds. Store-forward data is synchronized with the Standby AppEngine.

When failover conditions occur, no more than 30 seconds of history data is lost in the transition from Standby to Active status. For more information, see the Wonderware Developer Network.

Working with Data Acquisition Redundancy

The RedundantDIObject monitors and controls the redundant DIObject data sources at the object level. Unlike redundant AppEngines, individual DIObject data sources do not have redundancy-related states. For all practical purposes, they function as standalone objects.

Only one DIObject data source provides field device data through the RedundantDIObject at a time. Both data sources must have commonly-configured DAGroups. These must be reflected in and channeled through the RedundantDIObject, which monitors the two DIObject data sources. It also determines which one is Active at any given time. Both data sources must also have the same item address space.

Configuring Data Acquisition Redundancy

Data acquisition redundancy objects involve two DIObjects and the RedundantDIObject. In data acquisition redundancy, you must configure all three components:

- Primary DIObject data source
- Backup DIObject data source
- Redundant DIObject data source

Because data acquisition redundant components are essentially standalone objects, all valid operations that apply to any other ApplicationObjects apply to the three objects.

All IDE commands, Galaxy Dump and Load functions, and import and export operations are valid on the two DIObject data sources and the RedundantDIObject.

See the online help associated with each DIObject for help in configuring its Object Editor. Also see the help associated with the RedundantDIObject.

Before you can deploy the RedundantDIObject, you must configure at least one scan group. Also, configure only scan, block read, and block write groups shared by the Primary and Backup DIObjects in the RedundantDIObject.

To configure the Redundant DIObject

- 1 On the **General** tab of the Object Editor, set the **Primary DI Source** and **Backup DI Source**.
- 2 Set up the common scan groups.
- 3 Set up the common block read and block write groups on the tabs of the Object Editor.

Deploying Redundant DIOjects

Deployment for data acquisition redundancy is the same as individually deploying unrelated objects. No special conditions apply to each DIOject data source and the RedundantDIOject.

See *Deploying and Running an Application* on page 131 for more information about deploying objects.

What Happens in Run Time

The three objects in the data acquisition redundancy scheme (RedundantDIOject and its two DIOject data sources) work like any other ArcestrA object when deploying, alarming, and historizing. They have no special redundancy-related states or restrictions.

During run time, the RedundantDIOject monitors the status of the DIOject data sources, and handles the switching from Active to Standby object if failure conditions occur.

RedundantDIOject and PLC Connectivity

For the RedundantDIOject, you can use the scan group PingItem attribute to monitor the connection status of the PLC at run time. If you are using the redundancy feature of the RedundantDIOject to communicate with DIOjects, you should configure the PingItem attribute for each scan group.

Glossary

application	A collection of objects in a Galaxy Repository that performs automation tasks. Synonymous with Galaxy. There can be one or more applications within a Galaxy Repository.
Application Engine (AppEngine)	A scan-based engine that hosts and executes the run-time logic contained within AutomationObjects.
ApplicationObject	An AutomationObject that represents some element of your production environment. This can include things like <ul style="list-style-type: none">• An automation process component. For example, a thermocouple, pump, motor, valve, reactor, or tank• An associated application component. For example, function block, PID loop, Sequential Function Chart, Ladder Logic program, batch phase, or SPC data sheet
Application Server	<p>The supervisory control platform. Application Server uses existing Wonderware products such as InTouch for visualization, the Wonderware Historian for data storage, and the device Integration product line like a Data Access Server (DAServer) for device communications.</p> <p>An Application Server can be distributed across multiple computers as part of a single Galaxy namespace.</p>
Application view	The Applications view shows the object-related contents of the Galaxy in four different ways: Model view, Deployment view, Derivation view, and Operations view. The Model view appears when the IDE is opened for the first time.
Archestra	The distributed architecture for supervisory control and manufacturing information systems. It is an open and extensible technology based on a distributed, object-based design.

ArchestrA Object Toolkit	A programmer's tool to create new ApplicationObjects and Device Integration Object (DIOObjects) templates, including configuration and run-time implementations. Includes a tool to build DI Objects and create unique Domain Objects that interact with DIOObjects in the ArchestrA environment.
ArchestrA Symbol	A graphic you create and use to visualize data in an InTouch HMI system. You use the ArchestrA Symbol Editor to create ArchestrA Symbols from basic elements, such as rectangles, lines, and text elements.
area	A logical grouping of AutomationObjects that represents an area or unit of a plant. It is used to group related objects for alarm, history, and security purposes. It is represented by an area AutomationObject.
area object	The system object that represents an area of your plant within a Galaxy. The Area Object acts as an alarm concentrator, and places other Automation Objects into proper context with respect to the actual physical automation layout.
assignment	The designation of a host for an AutomationObject. For example, an AppEngine object is assigned to a WinPlatform object.
attribute	An externally accessible data item of an AutomationObject.
attribute reference string	A text string that references an attribute of an AutomationObject.
AutomationObject	An object type that represents permanent things in your plant, such as ApplicationObject or Device Integration Object (DIOObjects), with user-defined, unique names within the Galaxy. It provides a standard way to create, name, download, execute, and monitor the represented component.
Backup Application Engine	The object created by the ArchestrA infrastructure when the Primary object is enabled for redundancy. See redundancy for further details.
base template	A root template at the top of a derived hierarchy. Unlike other templates, a base template is not derived from another template but developed with the ApplicationObject Toolkit and imported into a Galaxy. All base templates names start with a dollar sign (\$).
Block Read Group	A DAGroup that is triggered by the user or another object. It reads a block of data from the external data source and indicates the completion status.

Block Write Group	A DAGroup that is triggered by the user or another object after all the required data items are set. The block of data is sent to the external data device. When the block write is complete, it indicates the completion status.
Bootstrap	The base ArcestrA service which is required on all ArcestrA computers. It provides the base software environment to enable a platform and allows a computer to be included in the Galaxy Namespace.
change log	The revision history that tracks the life cycle activities of ArcestrA Objects, such as object creation, check in/check out, deployment, and import/export.
change propagation	The ability to create templates which allows each component template to support changes such that a change in one of the elements can be automatically propagated to all — or select, related — object instances.
check in	IDE operation for making a configured object available for other users to check out and use.
check out	IDE operation for the purpose of editing an object. It makes the item unavailable for other users to check out.
Checkpoint	The act of saving to disk the configuration, state, and all associated data necessary to support automatic restart of a running AutomationObject. The restarted object has the same configuration, state, and associated data as the last checkpoint image on disk.
compound object	An ApplicationObject that contains at least one other ApplicationObject.
contained name	An alternate naming convention that when combined with the TagName of the root container object, results in the hierarchical name. For example, for a given object, its Hierarchical Name = Line1.Tank1.InletValve and its Contained Name= InletValve.
containment	A hierarchical grouping that allows one or more AutomationObject to exist within the name space of a parent object and be treated like parts of the parent object. Allows for relative referencing to be defined at the template and instance level.
DAGroup	A data access group associated with Device Integration Object (DIOObjects). It defines how communications are achieved with external data sources. It can be a Scan Group, Block Read Group or Block Write Group.

DAServer Manager (DAS Manager)	The System Management Console (SMC) snap-in supplied by the Data Access Server (DAServer) that provides the required interface for activation, configuration, and diagnosis of the DAServer.
Data Access Server (DAServer)	The server executable that handles all communications between field devices of a certain type and client applications. Similar to I/O Servers but with more advanced capabilities.
Data Access Server Toolkit (DAS Toolkit)	A developer tool that can build a Data Access Server (DAServer).
Deployment	The operation which instantiates an AutomationObject instance in the ArchestrA run time. This action involves installing all the necessary software and instantiating the object on the target platform with the object's default attribute data from Galaxy Repository.
Deployment view	The part of the Application view in the IDE that shows how objects are physically dispersed across Platforms, areas and Engines. This is a view of how the application is spread across computing resources.
derivation	The creation of a new template based on an existing Template.
Derivation view	The part of the Application view in the IDE that shows the parent-child relationship between base templates, derived templates and derived instances. A view into the genealogy of the application.
derived template	Any template with a parent template. Derived templates inherit the attributes of the parent template. You can changes these attributes in the derived template.
Device Integration Object (DIObjects)	An AutomationObject that represents the communication with external devices or software. DI Objects run on an Application Engine (AppEngine), and include DINetwork Objects and DIDevice Objects.
DIDevice Object	An object that represents the actual external device (for example, a PLC or RTU) that is associated with a DINetwork Object. It can diagnose and browse data registers of the DAGroups for that device.
DINetwork Object	An object that represents the network interface port to the device through the Data Access Server (DAServer) or the object that represents the communications path to another software application. It provides diagnostics and configuration for that specific network card.

element	Basic shapes, such as rectangles, lines, and text elements, and controls you can use to create an ArcestrA Symbol to your specifications.
Engine Object	An ArcestrA system-enabled object that contains Local Message Exchange and provides a host for ApplicationObjects, Device Integration Object (DIOObjects) and area objects.
event record	The data that is transferred about the system and logged when a defined event changes state. For example, an analog crosses its high level limit, an acknowledgement is made, or an operator logs in to the system.
export	The act of generating a package file (.aaPKG) extension from persisted data in the Galaxy database. You can import the resulting .aaPKG file into another Galaxy.
FactorySuite Gateway	A Microsoft Windows application program that acts as a communications protocol converter. Built with the ArcestrA DAS Toolkit, FS Gateway links clients and data sources that communicate using different data access protocols.
Galaxy	The entire application. The complete ArcestrA system consisting of a single logical name space (defined by the Galaxy database) and a collection of platform objects, Engine Objects and other objects. One or more networked computers that constitute an automation system. This is referred to as the Galaxy Namespace.
Galaxy database	The relational database containing all persistent configuration information like templates, instances, and security in a Galaxy Repository.
Galaxy Database Manager	A utility to manage your Galaxy. It can back up and restore Galaxies if they become corrupt or to reproduce a Galaxy on another computer. The Galaxy Database Manager is part of the System Management Console (SMC).
GalaxyObject	The object that represents a Galaxy.
Galaxy Repository	The software sub-system consisting of one or more Galaxy databases.
Graphic Toolbox	The part of the IDE main window that shows a hierarchy of graphic toolsets, which contain ArcestrA Symbols and client controls.
hierarchical name	The name of the object in the context of its container object. For example, Tank1.OutletValve, where an object called Tank1 contains the OutletValve object.

Historical Storage System (Historian)	The time series data storage system that compresses and stores high volumes of time series data for later retrieval. The standard historian is the Wonderware Historian.
host	The parent of a child instance in the deployment view. Example: a Platform instance is a Host for an Application Engine (AppEngine) instance.
import	The act of reading a package file (.aaPKG) and using it to create AutomationObject instances and templates in the Galaxy Repository.
instance	An object derived from a template. You deploy instances to the run-time environment.
instantiation	The creation of a new instance based on a corresponding template.
Integrated Development Environment (IDE)	The Integrated Development Environment (IDE) is the interface for the configuration side of Application Server. In the IDE, you manage templates, create instances, deploy and un-deploy objects, and other functions associated with the development and maintenance of the system.
InTouch View	InTouch View clients are InTouch run-time clients that solely use of the Application Server for its data source. In addition, standard InTouch run times can leverage Application Server with the addition of a Platform license.
InTouchViewApp object	Represents an InTouch application in the Wonderware Application Server environment. The InTouchViewApp object manages the check-in, check-out, and deployment of an InTouch application.
I/O count	Number of I/O points being accessed within a Galaxy. I/O points are real I/O and are not equivalent to InTouch tags. I/O count is based on the number of I/O points that are configured through an OPC Server, I/O Server, Data Access Server (DAServer) or InTouch Proxy Object, over the whole Application Server namespace, regardless of how many computers are in the system.
Message Exchange	The object to object communications protocol used by Application Server. Message Exchange includes LMX communication between objects NMX communication between Galaxy nodes.
Model view	The area in the Application view in the IDE that shows how objects are arranged to describe the physical layout of the plant and supervisory process being controlled.

object	Any template or instance in a Galaxy database. A common characteristic of all objects is they are stored as separate components in the Galaxy Repository.
object extensions	The capability to add additional functions to an AutomationObject while not changing the object's original behavior. Can be added to derived templates and object instances. They include Scripts, User Defined Attributes (UDAs) and Attribute Extensions.
Object Viewer	A utility in which you can view the attribute values of the selected object in run time. This utility is only available when an object is deployed. Object Viewer shows you diagnostic information on ApplicationObjects so you can see performance parameters, resource consumption and reliability measurements. In addition to viewing an object's data value, data quality and the communication status of the object, you can also modify some of its attributes for diagnostic testing. Modifications can include adjusting timing parameters and setting objects in an execution or idle mode.
OffScan	The state of an object that indicates it is idle and not ready to execute its normal run-time processing.
OnScan	The state of an object in which it is performing its normal run-time processing based on a configured schedule.
Operations view	The area in the IDE that shows the results of validating the configuration of objects.
package definition file (.aaPDF)	The standard description file that contains the configuration data and implementation code for a base template. File extension is .aaPDF.
package file (.aaPKG)	The standard description file that contains the configuration data and implementation code for one or more objects or templates. File extension is .aaPKG.
Platform Count	<p>Number of computers in the Galaxy. Each Workstation or Server communicating directly with the Application Server requires a platform to be part of the Galaxy Namespace. This includes each InTouch and InTouch View client. Each InTouch Terminal Services Session needs a Application Server Terminal Services Session License.</p> <p>A Platform License includes a per seat FSCAL2000 with Microsoft SQL Server CAL. Stand-alone computers only hosting InSQL Servers or a remote Data Access Server (DAServer) do not need a platform license.</p>

Platform Manager	This utility is an extension snap-in to the ArcestrA System Management Console (SMC). Provides Galaxy application diagnostics by allowing you to view the run-time status of some system objects and to perform actions upon those objects. Actions include setting platforms and engines in an executable or idle mode and starting and stopping platforms and engines.
platform object	An object that represents a single computer in a Galaxy, consisting of a system wide message exchange component and a set of basic services. This object hosts all Application Engines.
Primary Application Engine	The object created by the ArcestrA infrastructure when the Backup object is created through redundancy. See redundancy for further details.
properties	Data common to all attributes of objects, such as name, value, quality, and data type.
proxy object	An AutomationObject that represents an actual product for the purpose of device integration with the Application Server or InTouch HMI. For example, a Proxy object enables the Application Server to access an OPC server.
redundancy	Two computers: One executes objects. The other is a stand by.
RedundantDIObject	Monitors and controls the redundant Device Integration Object (DIObjects) data sources. Unlike redundant AppEngines, individual DIObject data sources do not have redundancy-related states. They function as stand-alone objects.
Redundant Message Channel	A dedicated Ethernet connection which is required between the platforms hosting redundant engines. The RMC is vital to keep both engines synchronized with alarms, history, and checkpoint items from the engine that is in the Active Role. Each engine also uses this Message Channel to provide its health and status information to the other.
reference	A string that refers to an object or to data within one of its attributes.
relative reference	Objects may refer to themselves, containers, hosts or to child objects elsewhere in the parent/child hierarchy using special reserved keywords such as “Me” or “MyContainer”. Relative references continue to work properly even if the object that is referenced is renamed. Examples of relative references are “Me”, “MyArea”, “MyContainer”, and “MyPlatform”.

remote reference	The ability to redirect ArchestraA object references or references to remote InTouch tags. The new script function that redirects remote references at run time is <code>IOSetRemoteReferences</code> .
Scan Group	A DAGroup that requires only the update interval be defined. The data is retrieved at the requested rate.
Scan State	The Scan State of an object in run time. This can be either <code>OffScan</code> or <code>OnScan</code> .
security	Application Server security is applied to IDE, System Management Console (SMC), and the run-time data level. At the run-time data level which centralizes the definition of all permissions to the <code>ApplicationObjects</code> . These <code>ApplicationObjects</code> can be accessed by a variety of clients but the security is centrally defined, allowing ease of maintenance. Users that are allowed to modify these <code>ApplicationObjects</code> at run time are mapped to the objects by user-defined roles. These roles can be mapped directly to existing groups in a Microsoft Domain or workgroup.
System Management Console (SMC)	The central run-time system administration/management product where you perform all required run-time administration functions.
system object	An object that represents an area, platform or engine.
TagName	The unique name given to an object. For example, for a given object, its <code>TagName</code> = <code>V1101</code> and its <code>HierarchicalName</code> = <code>Line1.Tank1.InletValve</code> .
template	An object containing configuration information and software templates used to create a derived template or instance.
Template Toolbox	The part of the IDE main window that shows Toolsets containing templates. The Template Toolbox shows a tree view of template categories in the Galaxy.
Toolset	A named collection of templates shown together in the IDE Template Toolbox.
User Defined Attributes (UDA)	Allow you to add new functionality to an object. An attribute is added to an object at configuration time.
ViewEngine object	Hosts <code>InTouchViewApp</code> objects. The <code>ViewEngine</code> object supports common engine features such as deployment, undeployment, startup, and shutdown.

WinPlatform object An object that represents a single computer in a Galaxy. A WinPlatform object consists of a system-wide message exchange component and a set of basic ArcestrA services. The WinPlatform object hosts the Application Engine (AppEngine).

Index

A

- active object 286
- active object, redundancy 286
- adding custom help to objects 71
- Advanced Communication Management
 - configuring scan modes 134–136
 - description 132
 - saving alarm data 186
 - saving historical data 157
 - setting for Galaxy 134
- AlarmInhibit alarm attribute 183
- AlarmMode alarm attribute 184
- AlarmModeCmd alarm attribute 183
- alarms
 - Area AutomationObjects 201
 - configuring 192
 - configuring for system objects 192–193
 - definition 173
 - disabled 186, 188
 - disabling 186
 - distributors 201
 - enabled 186, 188
 - extensions 114, 122, 200
 - grouping 201
 - individual 185
 - limit type 180
 - notification data 176–178
 - object attributes 183–184
 - propagating timestamps 190–191
 - rate of change type 182
 - setting categories 200
 - silenced 186, 188
 - specifying 200
 - state type 179
 - statistical type 183
 - subscribing 202
 - subscription 202
 - suppressed 186
 - target deviation type 181
 - throttling 189
 - types 179–182
 - undeploying clients 145
- aliases
 - rules for locking in scripts 110
 - using in scripts 109
- allowed IO, viewing license information 272
- analog device objects, output extensions 119
- AppEngine objects
 - deleting redundant 291
 - deploying redundant 291
 - description 41
 - history configuration 162
 - redundancy 287
 - redundancy states 295

- ApplicationObject containment 53
 - applications
 - ArchestrA architecture, ArchestrA 209
 - templates 40
 - using views 21
 - ArchestrA
 - messaging system 209
 - user accounts 278
 - Area AutomationObjects, alarms 201
 - Area object, description 41
 - attributes
 - dynamic 219
 - hardware register 219
 - hidden in UDAs 103
 - historizing 150
 - in runtime, writing to UDAs 66
 - keeping runtime values 143
 - locking in instances 65
 - locking in templates 65
 - locking in UDAs 66
 - property, finding 77
 - runtime 219
 - security 68
 - security locked in parent 69
 - unlocking in instances 65
 - unlocking in templates 65
 - AutomationObjects at runtime 294
- B**
- backup object
 - description 285
 - redundancy 285
 - backup server, redundancy 284
 - base templates
 - creating derived templates 39
 - description 37
 - importing 97
 - modifying 21
 - bit field access 81
 - Boolean
 - alarm extensions 122
 - data types 120
 - label extension 76
- C**
- cascade
 - deploy 137
 - deploy, selecting 139
 - changing, users 34
 - check-in comments, setting 31
 - checking objects
 - in 88
 - out 87
 - clients and alarms 202
 - communication at runtime 294
 - components in Galaxies 13
 - configuration, Object Editor Extension page 200
 - configure user information 31
 - configuring
 - alarm providers 192
 - data acquisition redundancy 301
 - history 150
 - multiple network cards 279
 - network cards 288
 - objects to store history 171
 - redundancy 289
 - security 233
 - user information 31
 - WinPlatforms and AppEngines for history 162
 - configuring bit field access 81
 - configuring computers for redundancy 284
 - configuring redundancy
 - AppEngine 289
 - ordering network connections 288
 - connecting
 - existing Galaxy 17
 - remote node for the first time 241
 - contained
 - names 51, 56, 93
 - templates, creating new 49
 - contained objects
 - naming conventions 57, 60
 - viewing 60
 - containment
 - definition 50
 - examples 58
 - names, scripting 56
 - naming conventions 57, 60
 - relationships, viewing 23, 57
 - templates 50
 - copyright 2
 - creating
 - contained templates 50

- derived templates 48
- extensions 112
- Galaxy, security restrictions 15
- instances 92
- object extensions 113
- scripts 106
- toolsets 46, 47
- UDAs 101
- creating user accounts 278
- crossover cable, wiring redundancy 284
- .csv files
 - characters in 248
 - editing 246
 - importing 249
 - structure 247
 - time formats 248
- custom, help, locating folders 71
- customizing
 - derived templates 48
 - information for a Galaxy 31
 - object help 71
 - your workspace 29

D

- data
 - acquisition redundancy 301
 - types and bit field access 81
- data types, configure history 151
- default, templates 37, 39
- deleting Galaxies, before you start 245
- deploy host objects first 137
- deploying
 - DINetwork objects 137
 - history 156
 - imported objects 100
 - instances 35
 - objects 137
 - redundant diobjects 302
 - templates 35
- deployment
 - error messages 141
 - redundancy 302
- deployment status
 - pair undeployed 289
 - partial deployed 289
 - partial undeployed 289
 - redundancy objects 301
 - viewing icons 138

- Deployment view
 - assignment relationships 24
 - using 24
- Derivation view
 - parental relationships 27
 - using 27
- derivation, viewing objects 27
- derived locked scripts 111
- derived templates 39, 42
 - contained names 56
 - creating 48
 - customizing 48
 - nesting 49
- deriving templates from derived templates 49
- device intergration templates 40
- DIDevice objects, definition 40
- DINetwork objects
 - configuration limits 25
 - definition 40
 - deploying 137
- DIOBJECT data sources 301
- disabled alarms 186, 188
- disabling
 - alarms 186, 188
 - redundancy 291
- disk space requirements 277
- DNS settings, configuring 280
- documentation conventions 11
- dynamic attributes 219

E

- editing
 - objects 61
 - scripts 104
- editing .csv files 247
- editing .csv files, characters in 248
- enabled alarms 186, 188
- enabling alarms 188
- event
 - distributors 201
 - subscription 202
- events
 - definition 173
 - types 174–176
- execution order of objects, setting 71
- expanding tabs 30
- exporting

- Galaxies
 - exporting file structure 247
 - Galaxy dump file 246
 - instances 95, 246
 - object help 246
 - objects 95, 246
 - script function libraries 96
 - script libraries 246
 - scripts 95, 96
 - exporting templates 246
 - extension inheritance 112
 - characteristics 112
 - Extensions page 75, 112
 - extensions, deleting 113
- F**
- failover and redundancy 283
 - field reference object 119
 - finding
 - help folders 71
 - object attributes 77
 - objects 221
 - floating
 - areas 29
 - views 29
 - fonts, languages 256
 - formatting reference strings 215
- G**
- Galaxies
 - adding a language 254
 - backing up 243
 - changing 245
 - changing the default language 257
 - components of 13
 - creating 15
 - default users 228
 - definition 13
 - deleting 245
 - deploying components 14
 - determining status 131
 - disk space requirements 277
 - Galaxy Repository 14
 - language settings 253
 - location on the network 13
 - logging in to 33
 - logging out of 33
 - managing multiple 244
 - naming conventions 16
 - opening with security 17
 - removing a language 255
 - reserved names 16
 - restoring 243
 - specifying GR node name 15
 - switching 245
 - synchronizing time 250
 - synchronizing time in Windows 2000 or XP 250
 - validating 91
 - viewing text dump 247
 - Galaxy Browser 77
 - Galaxy dump files
 - characters 248
 - importing 249
 - time formats 248
 - general Object Editor layout 64
 - generating
 - alarms 299
 - history 300
 - group
 - lock icons 69
 - locking 69
 - security 69
 - grouping alarms 201
- H**
- hardware register attributes 219
 - heartbeats 277
 - help header structure 63
 - hidden attributes, UDAs 103
 - hiding
 - areas 30
 - views 30
 - hierarchical
 - names 51, 57, 94
 - names, viewing 23
 - historized data
 - store forward 158
 - when data is stored 157
 - historizing
 - attributes 150
 - data, installing Wonderware Historian 148
 - history
 - deploying 156
 - undeploying 156

History extension 123
 history, data types, configuring 151
 host attributes 247
 hosting, multiple Galaxies in one Galaxy repository 252
 HTML editors for customizing object help 71

I

I/O licenses 271
 icons, deployment status 138
 IDE objects in runtime 130
 IDE, views 18
 importing
 base templates 97
 .csv files 249
 Galaxy load files 249
 objects 97
 objects and deploying 100
 objects and security groups 99
 objects with scripts 99
 script function libraries 99
 individual alarms 185
 inheriting attributes, parent and child relationships 35
 Input extensions 113
 Input extensions, data quality 121
 InputOutput
 attributes in scripts 116
 extension 115
 InputOutput extensions, data quality 121
 installing Wonderware Historian 148
 instances 37
 creating 92
 defined 37
 deploying 35
 exporting 95, 246
 locking attributes 65
 naming conventions 92
 on other computers 24
 reserved names 93
 unlocking attributes 65
 InTouch
 alarm and event client 203
 references in a Galaxy 222
 IO, viewing allowed 272
 IP address, network settings 280

L

languages
 changing default for a Galaxy 257
 configuring for a Galaxy 253, 254
 configuring for Galaxy 253
 dictionary files 258
 exporting data for all symbols 259
 exporting data for specific objects 260
 exporting for managed InTouch application 261
 fonts 256
 removing from a Galaxy 255
 translating symbol text 258
 .lic files, using 276
 license information, viewing 272
 licensed objects
 objects
 licenses 271
 licensing issues 271
 limit alarms 180
 alarms
 limit type 180
 locked scripts in child objects 111
 locking
 aliases in scripts 110
 scripts 110
 template attributes 65
 UDAs 101
 logging
 in to disconnected networks 241
 in to Galaxies 33
 in to Galaxies, security enabled 33
 out of Galaxies 33

M

managed InTouch application, exporting language data 261
 managing templates with toolsets 45
 manually
 initializing scripts 102
 validating objects 91
 MDAS
 historizing data 149
 using 149
 Message Exchange and attributes 210
 mixed or native domains 242
 Model view, opening 21
 moving

- objects, undeploying 21
- views 29
- multiple
 - accounts per user 229
 - network cards in one computer 279
- multiple Galaxies 244

N

- naming conventions
 - containment 57, 60
 - Galaxies 16
 - instances 92
 - scripts 106
 - templates 49
 - toolsets 46
 - UDAs 103
- nesting templates 37, 52
- network
 - configuring DNS settings 280
 - configuring IP address 280
- network cards
 - multiple in one computer 279
 - redundancy 287
 - setting 279
 - specifying the order 279

O

- object
 - derivation, viewing 27
 - properties, viewing 43
 - relationships, viewing 22
- object attributes, finding 77
- Object Editor
 - getting help 63
 - opening 61
- object help
 - adding images 71
 - exporting 246
 - HTML editors for customizing 71
- Object Information page 70
- objects
 - checking in 88
 - checking out 87
 - deleting 95
 - deploying 137
 - enabling alarms 188
 - exporting 95, 246
 - exporting language data 260

- help folders, finding 71
- importing 97
- in runtime 130
- inheriting extensions 112
- opening Object Editor 61
- redeploying 142
- specific information 119
- undeploying 142
- validating manually 91
- Objects Information page 70
- Operations view, opening 28
- ordering network cards 279
- Output
 - extension 114
 - functionality 119
- Output extensions, data quality 121

P

- parent and child, relationships, viewing 27
- passwords, changing 33
- planning for deploying 129
- Primary AppEngine, viewing
 - attributes 79
- primary object, redundancy 285
- primary server, redundancy 284
- propagating changes, templates 38
- published InTouch application, exporting language data 262

R

- rate of change alarms 182
- rate-of-change alarms 182
- reassigning objects, undeploying 21
- recreating ArcestrA user account 278
- redeploying
 - objects 142
 - objects, uploading changes first 143
- redundancy
 - alarm generation 299
 - AppEngine states 295
 - ApplicationObjects configuration 291
 - configuring templates 290
 - during deployment 294
 - history generation 300
 - pair 284
 - runtime 286
 - setting network cards 287

- templates 290
 - wiring for 284
 - Redundancy, page 289
 - redundant AppEngines, deleting 291
 - redundant engines, undeploying 293
 - redundant partner deploy, selecting 139
 - RedundantDIObject 284
 - reference strings 210
 - references in scripts, validating 89
 - referencing objects, Galaxy Browser 77
 - relationships, viewing parent/child 27
 - renaming
 - contained objects 60
 - contained objects, updating
 - references 61
 - objects 93
 - reorganizing the workspace 30
 - required software 251
 - reserved names
 - instances 93
 - list of 16
 - templates 49
 - resetting
 - the workspace 30
 - workspace 30
 - restoring Galaxies 243
 - return views to the default locations 30
 - reusing existing objects 97
 - roles, deleting 240
 - rules for locking
 - aliases in scripts 110
 - scripts 110
 - runtime
 - and bit field access 81
 - attributes 219
 - configuration, uploading 143
 - how objects deploy from the IDE 130
 - runtime environment, scripting 105
- S**
- script function libraries
 - exporting 96
 - importing 99
 - script libraries, exporting 246
 - scripting
 - containment names 56
 - UDAs 102
 - scripts
 - aliases, using 109
 - automatically starting 102
 - execution types 106
 - exporting 95
 - initializing Startup scripts 102
 - InputOutput attributes 116
 - locked in child objects 111
 - locking rules 110
 - manually initializing 102
 - naming conventions 106
 - timing 105
 - validating 89, 105
 - Scripts page 72
 - security
 - assigning users roles 238
 - deleting roles 240
 - Galaxy users 228
 - groups, default 228
 - groups, importing objects 99
 - icon not available 69
 - icons 69
 - opening Galaxies 17
 - options in attributes 69
 - restricting access to attributes 68
 - restrictions, creating new Galaxies 15
 - security roles 228
 - security groups, deleting 240
 - setting network cards 279
 - setting prompts for check-in
 - comments 31
 - setting, alarms 201
 - silenced alarms 186, 188
 - single scan cycles 119
 - SmartSymbols, exporting language
 - data 261
 - standby object, redundancy 286
 - Startup scripts, initializing 102
 - state alarms 179
 - statistical alarms 183
 - store forward, historized data 158
 - storing history, configuring 171
 - subscribing to alarms 202
 - suppressed alarms 186
 - switch objects 119
 - symbols
 - exporting language data 259
 - translating text 258
 - synchronization schedule 251

- synchronizing
 - time across a Galaxy 250
 - views 30
- synchronizing time across networks 250
- synchronizing time across networks, Windows 2000 or XP 250
- system templates 40

T

- tagnames 51, 56, 93
- tagnames, containment 54
- target deviation alarms 181
 - alarms
 - target deviation type 181
- technical support, contacting 12
- Template Toolbox, location 19
- templates 37
 - alarm extensions 122
 - application 40
 - base 37, 39
 - classes 39
 - configuring redundancy 290
 - contained names 56
 - containment 50
 - creating derived 48
 - default 37, 39
 - defined 37
 - deploying 35
 - derived 39, 42
 - device integration 40
 - exporting 246
 - inheriting extensions 112
 - managing 45
 - naming conventions 49
 - nesting 37
 - nesting levels 52
 - propagating changes 38
 - reserved names 49
 - system 40
- text strings, specifying for Boolean states 76
- throttling alarms 189
- time formats, .csv files 248
- time master
 - specifying 250
 - specifying in Windows 2000 or XP 250
- timestamps
 - propagating in alarms 190–191

- saving as historical data 150
- toolsets
 - creating 46, 47
 - definition 45
 - deleting 47
 - managing 45
 - naming conventions 46
- troubleshooting 297

U

- UDA naming conventions 103
- UDAs
 - adding to objects 101
 - and scripting 102
 - hidden attributes 103
 - locking attributes 66
 - locking in child objects 101
 - page 73
 - scripting 102
- undeploying 142
 - AppEngine objects 293
 - history 156
 - objects 142
 - objects before moving 21
- undeploying redundant engines 293
- undeployment conditions 145
- unlocking, template attributes 65
- updating your license 271, 276
- uploading runtime configuration 143
- user accounts 278
- user defaults, setting 31
- users
 - changing 34
 - default 228
 - deleting 240

V

- validating
 - Galaxies 91
 - manually 91
 - object manually 91
 - objects 89
 - objects, viewing results 28
 - references in scripts 89
 - scripts 89, 105
- viewing
 - attributes in objects 218
 - containment relationships 60

- cross references 219
 - license information 272
 - object properties 43
 - object relationships 22
 - objects, assignment relationships 24
 - references 219
 - viewing your licenses 271
 - views in the IDE 18
 - views, synchronizing 30
- W**
- Windows
 - 2000 domains 250
 - user accounts 278
 - XP domains 250
 - WinPlatform object, configuring for redundancy 287
 - WinPlatform, object 40
 - WinPlatforms, history configuration 162
 - Wonderware Historian, installing 148
 - working
 - extensions 112
 - UDAs 101
 - workspace, resetting 30
 - writing
 - scripts 104
 - to attributes in runtime, UDAs 66

