ArchestrA™ Integrated Development Environment (IDE) User's Guide

Revision C

Last Revision: 8/23/05

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

Before You Start

About This Book

This book describes the user interface and functions of the ArchestrA™ Integrated Development Environment (IDE). This book is organized in the following fashion:

- Introduction: Includes detailed descriptions of elements of the IDE's user interface.
- Working with Objects: Includes importing and managing object templates, managing toolsets in the IDE's **Template Toolbox**, and creating, configuring and deploying object instances.
- Working with Object Editors: Includes using editors to configure objects.
- Working with History: Includes the relationship between Industrial Application Server objects and the history function provided by InSQL.
- Working with Alarms and Events: Includes events and alarms, the differences between the two and how to configure objects for reporting them through the InTouch alarm provider.
- Enhancing an Object's Functionality: Includes using the **Scripts**, **UDAs** and **Extensions** pages of an object editor to extend an object's functionality beyond its original capabilities.
- Working with Containment: Includes using object containment to create complex real-world devices.
- Working with References: Includes working with reference strings, the object **Properties** dialog box and finding lost objects.
- Working with Security: Includes configuring security for the ArchestrA environment.
- Galaxy Management: Includes the Galaxy Repository's SQL Server database, backing up Galaxies, creating and deleting Galaxies and configuring a Galaxy time master node.
- ArchestrA Redundancy: Includes using the redundancy functions in the ArchestrA infrastructure to provide strategic duplication of critical components in your application.
- Index

For more information on using ArchestrA, refer to your *Factory Suite A^2 Deployment Guide.*

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

Documentation Conventions

This documentation uses the following conventions:

Convention	Used for
Initial Capitals	Paths and filenames.
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 chapter(s) in your *ArchestrA IDE User's Guide* for a possible solution to any problem you have with the IDE. If you need to contact technical support for assistance, have the following information available:

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

C H A P T E R 1

Introduction

ArchestrA™ is the next generation architecture for supervisory control and manufacturing information systems. It is an open and extensible system of components based on a distributed, object-oriented design.

This document describes how to use the main configuration tool in the ArchestrA system, the Integrated Development Environment (IDE).

This chapter describes how to begin using the IDE and the basic IDE user interface.

For more information on using ArchestrA, see your *Factory Suite* A^2 *Deployment Guide.*

Contents

The IDE

The IDE is the integrated design and development tool where you configure and deploy all ArchestrA objects to target computers. It maintains and configures the objects in your application and the underlying infrastructure that supports your application.

With the IDE, you can import new types of objects (templates) into the Galaxy, configure instances of those template objects and deploy them to computers on your network. Multiple users can work concurrently in the same Galaxy on different sets of objects from different IDEs.

The IDE can be installed on any computer that has ArchestrA's Bootstrap software installed.

Starting the IDE

Before you can open the IDE, you need the following:

- At least one Galaxy in your Galaxy Repository.
- A valid license for your Galaxy Repository.

For more information about licensing requirements, refer to your *Licensing Guide* for the Industrial Application Server.

You might be required to login if security is enabled on the selected Galaxy. The ArchestrA security system controls user access to

- all parts of the IDE
- the utilities in the System Management Console
- the runtime environment

If you are denied access to the IDE or any operation typically available in the IDE, see the administrator of your ArchestrA security environment about additional privileges. See Working with Security for more information about the security system.

Use the steps below to start ArchestrA.

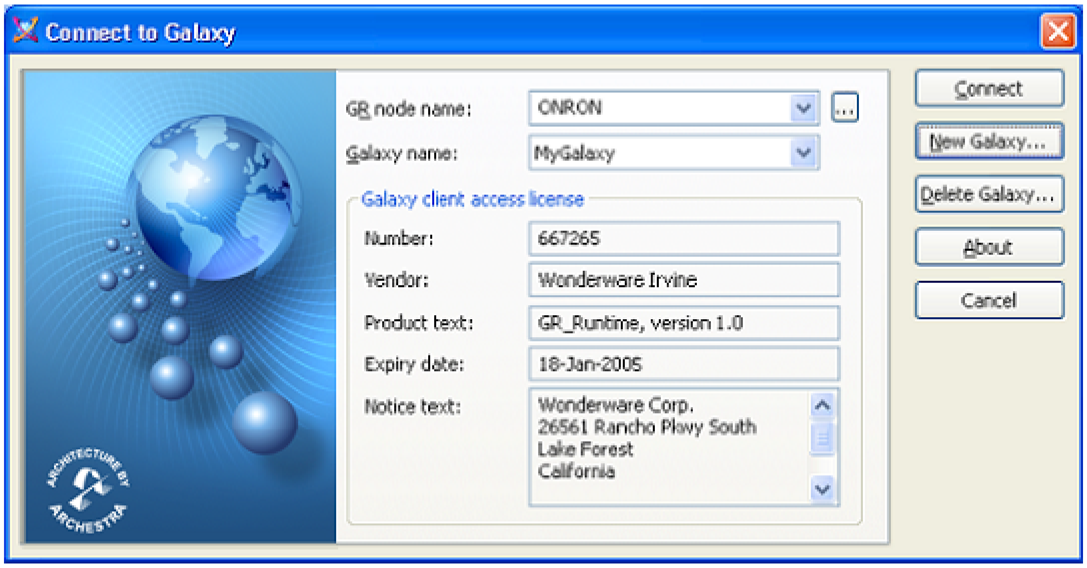
To start ArchestrA

1. To start the IDE, click `Start`, point to `Programs` and then to `Wonderware`. Click `ArchestrA IDE`.

Note Parts of the IDE's user interface look or function slightly different when Industrial Application Server is installed on different versions of Microsoft® Windows®. The procedures in this manual show a Windows XP environment.

Connecting to a Galaxy

Each IDE session requires connection to a specified Galaxy. You cannot start the IDE without opening a Galaxy. When you start the IDE, the **Connect to Galaxy** dialog box appears.



GR node name: ONRON

Galaxy name: MyGalaxy

Galaxy client access license

Number: 667265

Vendor: Wonderware Irvine

Product text: GR_Runtime, version 1.0

Expiry date: 18-Jan-2005

Notice text: Wonderware Corp.
26561 Rancho Pkwy South
Lake Forest
California

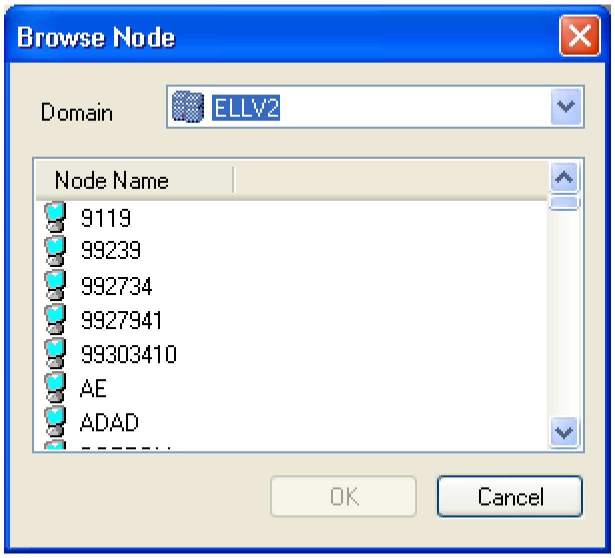
This dialog box has three groups:

- Galaxy Repository/Galaxy connect selections: The **GR Node Name** list and **Galaxy Name** list.
- Buttons: **Connect**, **New Galaxy**, **Delete Galaxy**, **About** and **Cancel**.
- Licensing information

If the **Galaxy Name** list is empty, you do not have a Galaxy on the computer shown in the **GR Node Name** box. Before you can start the IDE, either browse for a Galaxy on another node or create a new Galaxy.

To browse other nodes

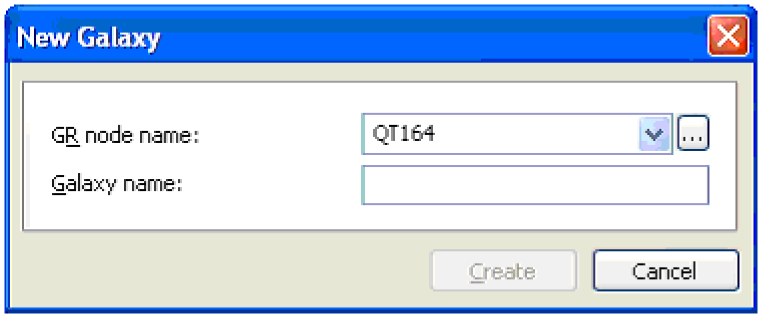
1. On the **Connect to Galaxy** dialog box, click the browse button [...]. The **Browse Node** dialog box appears.



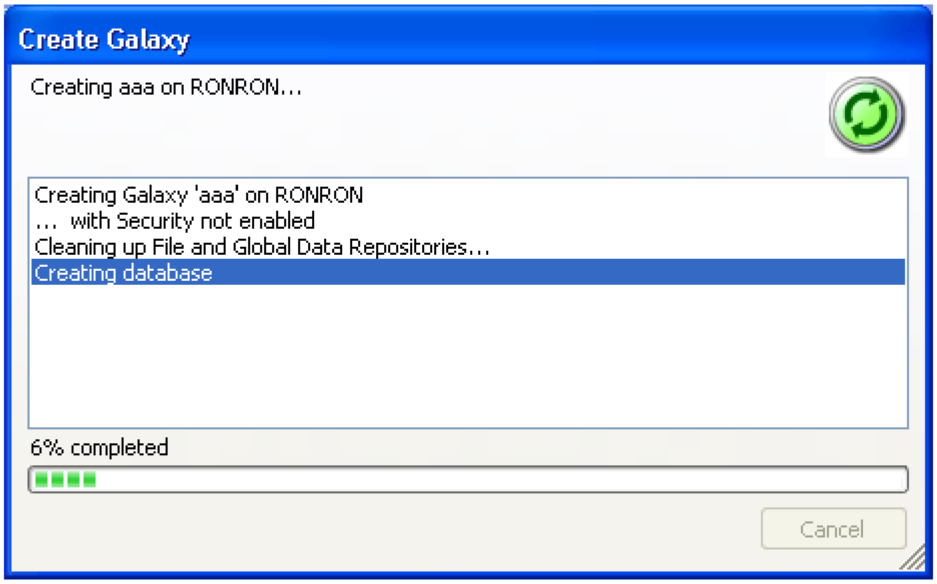
2. Do the following:
   - In the **Domain** box, select a domain.
   - In the **Node Name** box, select a computer.
   - Click **OK** to place the selected node name in the **GR Node Name** box of the **Connect to Galaxy** dialog box.

To create a new Galaxy

1. On the **Connect to Galaxy** dialog box, click **New Galaxy**. The **New Galaxy** dialog box appears.



2. Do the following:
   - From the **GR Node Name** list, select the computer on which you want to create the new Galaxy.
   - In the **Galaxy Name** box, type the name of Galaxy you want to create.
   - Click **OK** to begin creating the new Galaxy. The **Create Galaxy** progress box appears.



Creating aaa on RONRON...

Creating Galaxy 'aaa' on RONRON
... with Security not enabled
Cleaning up File and Global Data Repositories...
Creating database

6% completed

Note New Galaxies are created with no security. They also have the following characteristics: two users (DefaultUser and Administrator, both with full access to everything), two security roles (Default and Administrator, both with full privileges) and one security group (Default). See Working with Security for more information about the security model and setting security for your new Galaxy.

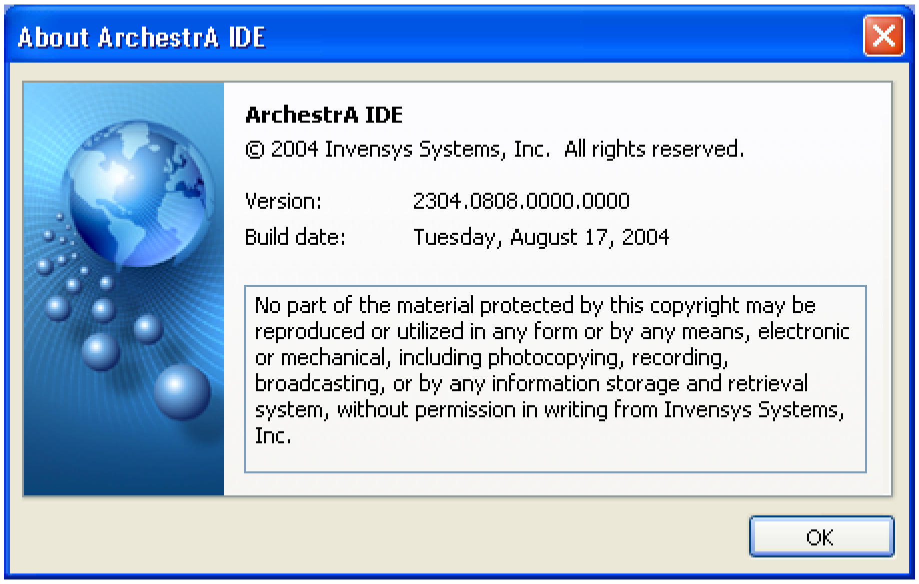
3. Click **Close** to return to the **Connect to Galaxy** dialog box.
4. After the **GR Node Name** and **Galaxy Name** options are correct, click **Connect** to start the IDE and to connect to that Galaxy.

   If you previously created one Galaxy on the GR node shown, the Galaxy's name is automatically shown. Click **Connect** to start the IDE and to connect to that Galaxy. If you previously created more than one Galaxy on the GR node shown, the most recently accessed Galaxy name is shown. Select the desired Galaxy from the **Galaxy Name** list and click **Connect** to start the IDE and to connect to that Galaxy.

Note If security is enabled, the Login dialog box appears before the IDE is started. If security is disabled, you are automatically logged in as a "DefaultUser" user. See Login/Logout for more information about the **Login** dialog box.

To delete the Galaxy shown in the **Galaxy Name** box, click **Delete Galaxy**.

To determine software version, copyright and other information about the ArchestraA IDE, click **About**. The **About ArchestrA IDE** dialog box appears. The following image is shown as an example.


About ArchestrA IDE

ArchestrA IDE
© 2004 Invensys Systems, Inc. All rights reserved.

Version: 2304.0808.0000.0000
Build date: Tuesday, August 17, 2004

No part of the material protected by this copyright may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, broadcasting, or by any information storage and retrieval system, without permission in writing from Invensys Systems, Inc.

The **Galaxy Client Access License** group of the **Connect to Galaxy** dialog box shows the following information:

- License Type
- License Number
- Vendor
- Product Text
- Expiry date (if any)
- License Notice Text

Licensing

You must have a FactorySuite Development (FS Dev) license on the development node. This license contains information about the I/O count you are allowed. You cannot configure or access the Galaxy database information using a configuration tool such as IDE or the Galaxy Tag browser from WindowMaker on a development node without a license.

The Galaxy license in the GR node, also called the AppServer license, defines the number of platforms and IO points the system is licensed for. The I/O count in the development license must be equal to or greater than the I/O count for your Galaxy.

If the I/O count associated with the FactorySuite Development license is not equal to or greater than the I/O count in the Galaxy license in your GR node, you cannot connect to the specified GR node.

If the I/O count is not equal to or greater than the I/O count for your GR node, you cannot open the IDE.

If a license-related message appears when you opening the IDE, you have a problem with your license. This message indicates one of the following conditions:

- No license is installed on the development node.
- Your license expired.
- You exceeded the licensed I/O count or number of WinPlatforms.
- The number of I/O points specified in your development node license is less than the I/O points specified in your Galaxy license.

The License Utility can help you correct these problems. Until the problem is resolved, you cannot:

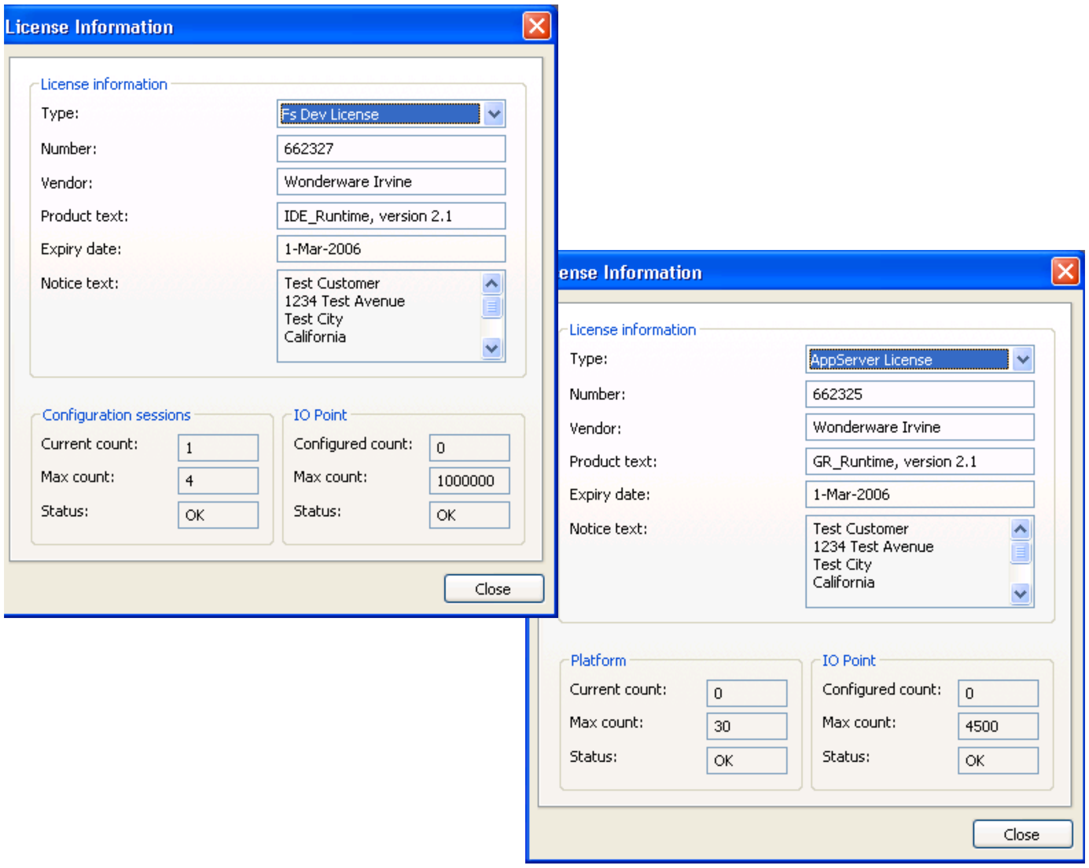
- Open the IDE.
- Connect to existing Galaxies.
- Create new Galaxies.

After you update your license, you can connect to your Galaxy and open the IDE with no further problems. For more information about correcting any licensing problems, contact your distributor.

Note If a license expires while you are using the IDE, you cannot connect to the Galaxy the next time you open the IDE.

To check your license

1. Double-click the **License** icon at the bottom of the IDE's Main Window to open the License Information dialog box.



License Information

License information

Type:	Fs Dev License
Number:	662327
Vendor:	Wonderware Irvine
Product text:	IDE_Runtime, version 2.1
Expiry date:	1-Mar-2006
Notice text:	Test Customer 1234 Test Avenue Test City California

Configuration sessions		IO Point	
Current count:	1	Configured count:	0
Max count:	4	Max count:	1000000
Status:	OK	Status:	OK

ense Information

License information

Type:	AppServer License
Number:	662325
Vendor:	Wonderware Irvine
Product text:	GR_Runtime, version 2.1
Expiry date:	1-Mar-2006
Notice text:	Test Customer 1234 Test Avenue Test City California

Platform		IO Point	
Current count:	0	Configured count:	0
Max count:	30	Max count:	4500
Status:	OK	Status:	OK

2. From the **License type** list, select the license you want to see more information about. To see information about the GR node license, select **AppServer**. To see more about the development node license, select **Fs Dev License**.

   License Information group

   - **Number**: License number.
   - **Vendor**: Name of software vendor.
   - **Product Text**: The software name and version number.
   - **Expiry Date**: Expiration date for your license.
   - **Notice Text**: Information about the software vendor.

   Configuration Sessions group (Fs Dev License) or **Platform** area (AppServer License)

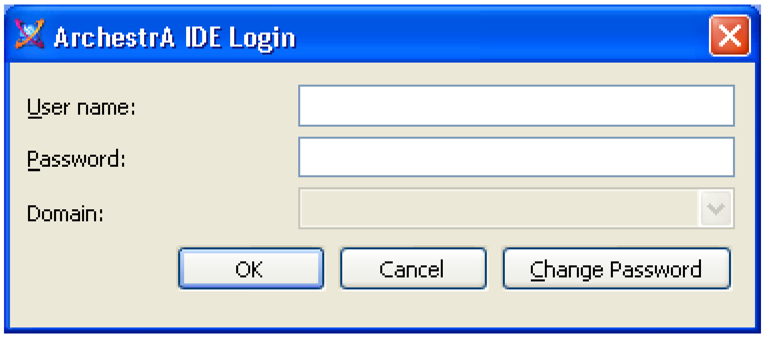
   - **Current Count**: Number of deployed WinPlatforms in your Galaxy.
   - **Max Count**: Maximum number of deployed WinPlatforms allowed by your license. If your license has no I/O and Platform Count feature line, N/A appears.
   - **Status**: Relationship between the current and maximum WinPlatform count values. Three statuses are possible: OK, Exceeded (deployed WinPlatform count exceeds the maximum allowed by your license) and DEV (your license has no I/O and Platform Count feature line).

   IO Point group

   - **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. Three statuses are possible: OK, Exceeded (configured I/O points count exceeds the maximum allowed by your license) and DEV (your license has no I/O and Platform Count feature line).

Login/Logout

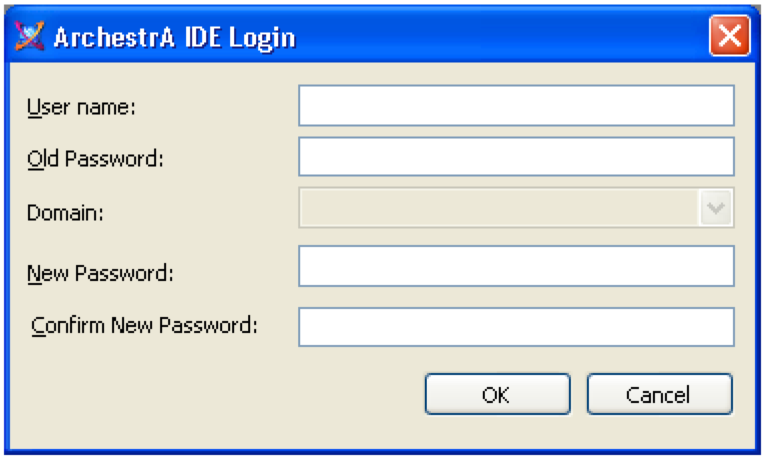
If you open a connection to a security enabled Galaxy, the **ArchestrA IDE Login** dialog box appears.



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, unless when the domain is local.

Click **OK**. Your login data is validated by the Galaxy Repository being accessed (and perhaps the operating system), and the IDE is started. If the GR does not recognize either your user name or password, a message appears.

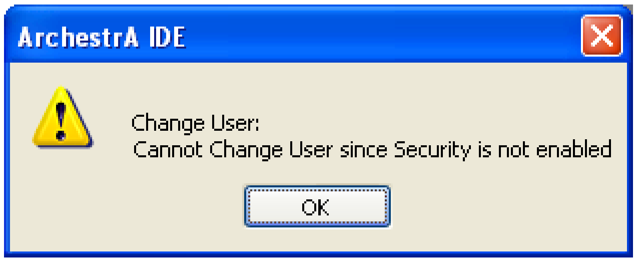
Click **Change Password** on the **ArchestrA IDE Login** dialog box to open the following dialog box. This button is unavailable until you enter a valid user name and password.



For instructions about setting up security in the ArchestrA environment, see Working with Security.

Changing Users

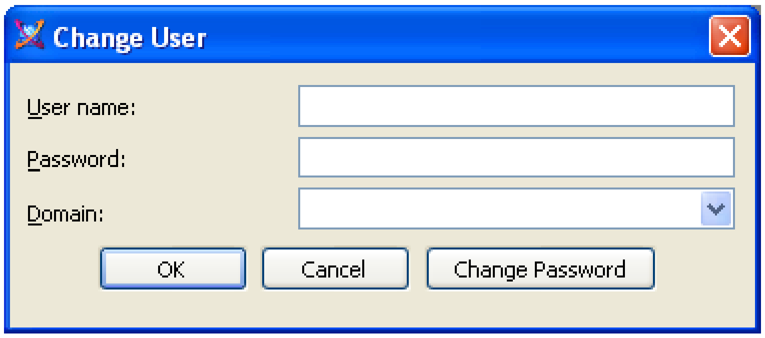
At any time during an IDE session, you can change user accounts by clicking **Change User** on the **Galaxy** menu. If the Galaxy has open security, this function does nothing and the following message appears.


ArchestrA IDE

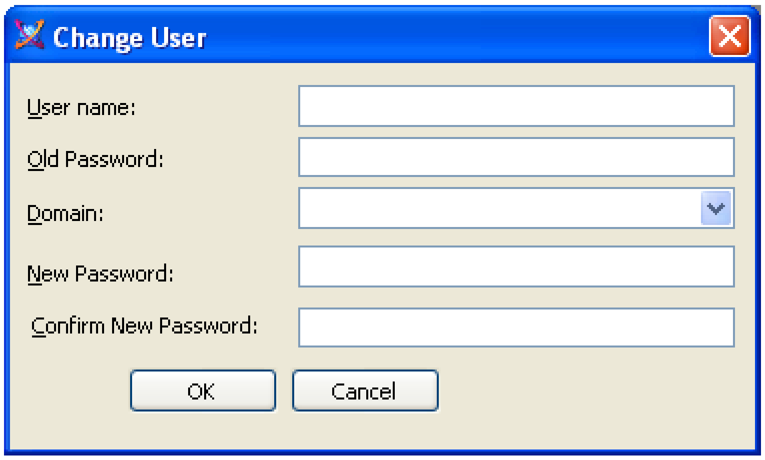
Change User:
Cannot Change User since Security is not enabled

Click **OK**. All users in an open security environment are treated as the Defaultuser by the Galaxy.

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



Enter login data for the new user and click **OK**. Click **Change Password** to change the password for the new user. The following dialog box appears.



Using a Desktop Shortcut

You can automate connection to a frequently used Galaxy with a desktop shortcut that specifies a Galaxy Repository and Galaxy.

To create a desktop shortcut

1. Right-click on your computer's desktop. Point to **New** and click **Shortcut**.
2. In the **Create Shortcut** dialog box, click **Browse** and do the following:
   - Browse to the installation path of the IDE's executable (`aaIde.exe`).
   - Select `aaIde.exe` and click **Open**.
   - Click **Next**.
   - Type a name for the shortcut. Click **Finish**.
3. Right-click the new shortcut and click **Properties** on the shortcut menu.
4. On the **Shortcut** page, place your cursor at the end of the **Target** box. Type a space, the Galaxy Repository node name, a backslash ("\") and then the Galaxy name.
5. Click **OK**.

To use the shortcut, double-click it on your desktop. If the shortcut contains invalid data, a message appears. Click **OK** to open the **Connect to Galaxy** dialog box. If the shortcut's Galaxy data is invalid, the **Galaxy Name** box displays the last accessed Galaxy.

Important! A desktop shortcut cannot contain login information. If security is enabled, double-clicking the shortcut opens the **Login** dialog box.

Starting Multiple IDEs

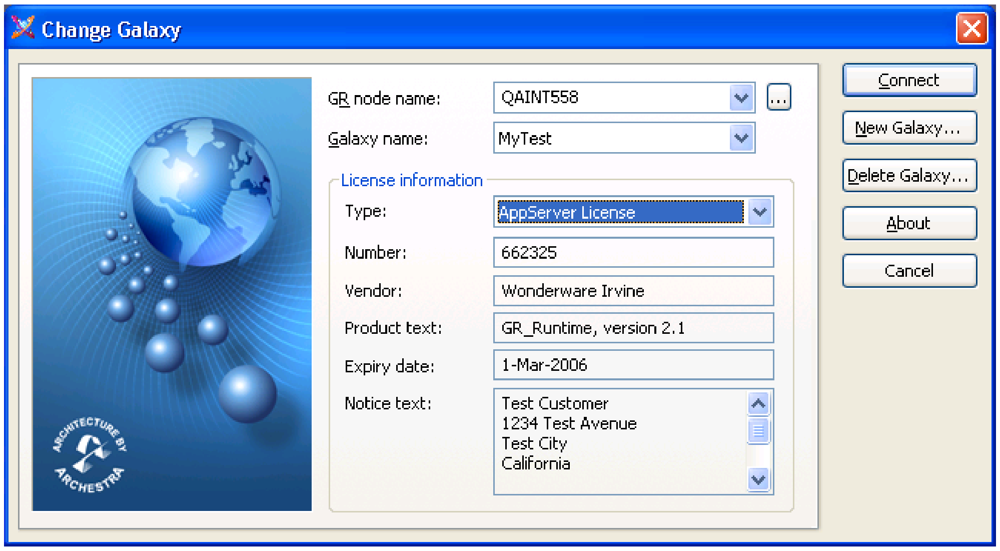
Multiple users of the IDE can connect to and configure the same Galaxy at the same time. To do this, each individual user starts the IDE and connects to the Galaxy in a normal way.

The ArchestrA environment ensures:

- Objects checked out by one user cannot be edited by another user. All users can determine who has checked out an object in the Galaxy.
- Configuration changes made and checked in by one user are displayed in the IDE of all other users.

Changing Galaxies

To change from one Galaxy to another, click **Change Galaxy** on the **Galaxy** menu. The **Change Galaxy** dialog box appears.



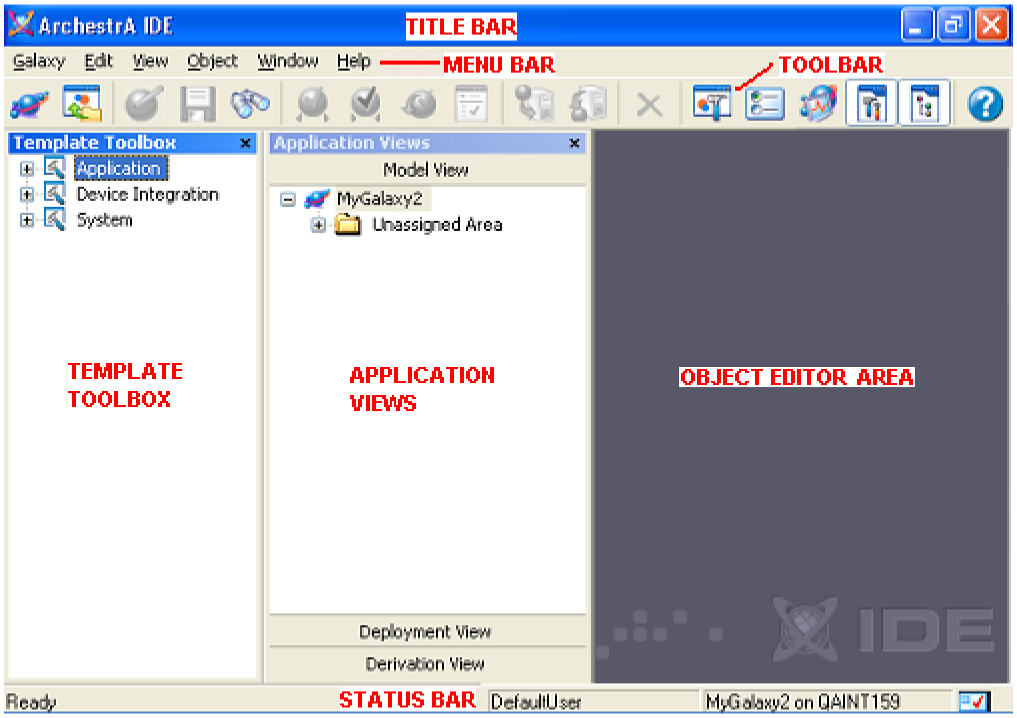
GR node name: QAINT558
Galaxy name: MyTest
License information
Type: AppServer License
Number: 662325
Vendor: Wonderware Irvine
Product text: GR_Runtime, version 2.1
Expiry date: 1-Mar-2006
Notice text: Test Customer
1234 Test Avenue
Test City
California

In this dialog box, you can select another Galaxy from the **Galaxy Name** option, another Galaxy Repository node from the **GR Node Name** option, create a new galaxy with the **New Galaxy** button, or delete the current galaxy with the **Delete Galaxy** button. To change galaxies, select the appropriate **GR Node Name** and **Galaxy Name**, and then click the **Connect** button.

All other options are the same as those on the **Connect to Galaxy** dialog box.

The IDE User Interface

The primary user interface of the IDE is the Main Window. Use it to create your application and deploy it on your domain.



The Main Window consists of the following elements:

- Title bar
- Menu bar
- Toolbar
- **Template Toolbox** pane
- **Application Views** pane
- Object editor area
- Status bar

Each of these elements appear by default.The Operations pane is not shown initially.

The title bar shows the name of the utility.

The other elements of the Main Window are described next.

Menu Bar

The IDE menu bar has the following menus: **Galaxy**, **Edit**, **View**, **Object**, **Window** and **Help**.

Depending on which object or Main Window element is in focus, what condition the element is in or whether certain functions are logically permitted, some menu options are unavailable. The following is a description of all menu options:

Galaxy menu: Provides Galaxy or user-level global options.

- **New**
  - **Instance**: Creates an instance of the template currently in focus.
  - **Derived Template**: Creates a derived template of the template currently in focus.
  - **Template Toolset**: Creates a new template toolset. This option opens the **New Toolset** dialog box.
- **Open**: Opens the editor of the object currently in focus.
- **Open Read-Only**: Opens the editor in read-only mode of the object currently in focus. There are several conditions that can place this restriction on opening an object’s editor, including: the object is checked out to someone else or you do not have configuration permissions in general or specifically for the object in question.
- **Close**: Closes the editor currently in focus. You are prompted to save changes made in the editor.
- **Import**
  - **Automation Object(s)**: Imports an object definition file (`.aapdf` or `.aapkg` extension). This option opens the Microsoft **Open** dialog box, from which one or more files can be selected at a time. See Importing Objects.
  - **Script Function Library**: Imports a library of script functions (`.aaSLIB`, COM .dll, COM .tlb, COM .olb, or `.NET` .dll files) into your Galaxy. Script function libraries provide operations for use within scripts. If the script function’s name is identical to one previously imported into the Galaxy, you are prompted to replace the current one.
  - **Galaxy Load**: Imports Galaxy configuration data from a comma separate variable file (`.csv` extension) that contains previously exported objects (see **Galaxy Dump** below). This option opens the **Open** dialog box where you navigate to the desired file.
- **Export**
  - **Automation Object(s)**: Exports the object(s) in focus to a package file (.aaPKG). This option opens the **Save As** dialog box. The exported file contains the attributes and configuration values of the original object(s). See Exporting Objects.
  - **All Automation Objects**: Exports all objects in the Galaxy to a package file (`.aaPKG`).
  - **Script Function Library**: Shares a library of script functions with another script writer. This option opens the **Export Script Function Library** dialog box.
  - **Galaxy Dump**: Saves the configuration data of a Galaxy to a comma separated variable file (`.csv` extension). You can change attribute values for objects and create new objects within the file by editing it in Excel. You can upload (see **Galaxy Load** above) the configuration data back into the Galaxy. This option opens the **Save** dialog box where you can name the `.csv` file and specify a location.

Note When you open the .csv file in Excel, make sure you select **Delimited** text and then **Comma**. When you are done making your changes, save the file as a .csv so it will correctly import back into Industrial Application Server.

- **Save**: Saves all configuration data in the editor currently in focus.
- **Save All**: Saves all configuration data in all editors currently open.
- **Configure**
  - **Security**: Opens the **Configure Security** dialog box where you can set all ArchestrA environment security parameters. See Working with Security for more information.
  - **View Security**: Opens the **Configure Security** dialog box in read-only mode.
  - **Time Master**: Opens the **Configure Time Master** dialog box where you can select a node on your network that serves as a time master for all other nodes.
  - **Customize Toolsets**: Opens the **Customize Toolsets** dialog box where you can specify which template toolsets appear in the **Template Toolbox**.
- **Galaxy Status**: Opens the **Galaxy Status** dialog box, showing various statistical counters related to the objects in your Galaxy.
- **Properties**: Opens the **Properties** dialog box where you can view general properties, attributes, references, cross-references, change log, operational limits and errors/warnings associated with the object currently in focus.
- **Change Galaxy**: Selects another Galaxy to open. This option opens the **Change Galaxy** dialog box.
- **Change User**: Changes the logged in user of the current session of the IDE. This option opens the **Change User** dialog box. If no ArchestrA security is enabled, using this option shows a message stating there is no security enabled.
- <Recent Galaxies List>: Connects to a recently-accessed Galaxy. This list shows the four most recently opened Galaxies, listed with the most recent first. Clicking on a Galaxy in the list opens the **ArchestrA IDE Login** dialog box if security is enabled.
- **Exit**: Closes an IDE session. You are prompted to save unsaved object editor configuration data. User preferences (for example, **Application Views** pane positioning) persist for the logged in user.

Edit menu: Provides object and user-level options.

- **Rename**: Puts the name of the object in focus in renaming mode.
- **Rename Contained Name**: Opens the **Rename Contained Name** dialog box where you can provide another contained name for the object in focus.
- **Delete**: Deletes the object currently in focus.
- **Find**: Opens the **Find** dialog box, in which you can set search criteria for finding objects by name and type.

- **User Information**: Sets global preferences for the logged in user, such as default system objects. This option opens the **Configure User Information** dialog box.

View menu: Provides options for showing or hiding elements of the IDE. This menu provides options for controlling the Main Window appearance. On your initial IDE login, all three Main Window components listed below are visible. When the user logs in again, he or she sees the previously saved user preference settings.

- **Application Views**: Puts into focus the **Application Views** part of the Main Window.
- **Template Toolbox**: Puts into focus the **Template Toolbox** part of the Main Window.
- **Operations**: Shows the **Operations** pane. To close it, click the **x** button.
- **Status Bar**: Toggles the status bar in the Main Window on and off.

Object menu: Provides object-related options.

- **Check Out**: Checks out an object from the Galaxy so that you can maintain sole authority to configure that object. No one connected to the Galaxy can change the configuration of the object until you check it back in to the Galaxy.
- **Check In**: Checks in to the Galaxy Repository an object which was previously checked out. This option opens the **Check In Object** dialog box.
- **Undo Check Out**: Reverses a previous check out without affecting the configuration of the object in question nor its change log. The result of this option is the object can be checked out by anyone connected to the Galaxy.
- **Override Check Out**: Disables the checked out flag on the object in focus. This option typically requires special permissions, and should be used in only those circumstances in which it is certain that the person who originally checked out the object is not currently configuring the object.
- **Validate**: Validates the configuration of one or more objects. This option opens the **Operations** pane. For more information, see Validating Objects.
- **View in Object Viewer**: Opens the Object Viewer utility, where you can view the attributes of the selected object. This option is available when the selected object is deployed.
- **Deploy**: Deploys the object or objects currently in focus to the nodes their configurations denote. This option opens the **Deploy Object** dialog box.
- **Undeploy**: Undeploys the object or objects currently in focus from the nodes that currently host them. This option opens the **Undeploy Object** dialog box.
- **Assign To**: Opens the **Assign To** dialog box, in which you specify which object the selected object is assigned to.

- **Unassign**: Removes the assignment already set for the selected object, and placing it in the **Unassigned Host** folder of the **Deployment View** or the **Unassigned Area folder** of the **Model View**. When selected in the **Derivation View**, all parent relationships (host, area and containment) are unassigned. This option opens the **Unassign** dialog box where you can monitor the unassignment process. This option fails if the selected object is currently deployed.
- **Set as Default**: Sets a system object, such as a WinPlatform, AppEngine or Area, as the default for assigning other objects.
- **Upload Runtime Changes**: Uploads the selected, deployed object’s configuration to the Galaxy. This option is useful when changes to certain writeable attributes are made in the configuration environment, but at a later time, the runtime object’s configuration is determined to be preferred. This option overwrites the configuration data in the Galaxy with runtime data.

Window menu: Provides options for manipulating the object editor area of the Main Window. This menu is available if at least one editor is open.

- **Cascade**: Typical Windows option for cascading (layering) multiple object editors.
- **Tile Horizontally**: Typical Windows option for showing all open object editors at one time in horizontal fashion.
- **Tile Vertically**: Typical Windows option for showing all open object editors at one time in vertical fashion.
- **Close All**: Closes all open object editors. If any data changed on any editor, you are prompted to save those changes individually for each editor.
- **Windows**: Opens the **Windows** dialog box where you can manage all open configuration editors. This option is available when at least one editor is open.
- <Open Windows List>: Selects and puts into focus an open object editor. This list appears when at least one editor is open.

Help menu:

- **Help Topics**: Opens the IDE Help file.
- **Object Help**: Opens the Help documentation for the object currently in focus.
- **About ArchestrA IDE**: Opens the **About ArchestrA IDE** dialog box, in which you can view copyright, software version and other information about the IDE.

Toolbar

The IDE toolbar consists of icons for quick access to frequently used options. Each icon is cross-references to menu bar options (see Menu Bar for descriptions. The names associated with each are based on the tooltip that appears when you point to each icon.

Icon	Name	Equivalent Menu Bar option
	Change Galaxy	**Change Galaxy** on **Galaxy** menu
	Import Automation Object(s)	**Automation Object(s)** on **Import** submenu of **Galaxy** menu
	Open	**Open** on **Galaxy** menu
	Save	**Save** on **Galaxy** menu
	Find	**Find** on **Edit** menu
	Check Out	**Check Out** on **Object** menu
	Check In	**Check In** on **Object** menu
	Undo Check Out	**Undo Check Out** on **Object** menu
	Properties	**Properties** on **Galaxy** menu
	Deploy	**Deploy** on **Object** menu
	Undeploy	**Undeploy** on **Object** menu
✕	Delete	**Delete** on **Edit** menu
	Customize Toolsets	**Customize Toolsets** on **Configure** submenu of **Galaxy** menu
	User Information	**User Information** on **Edit** menu

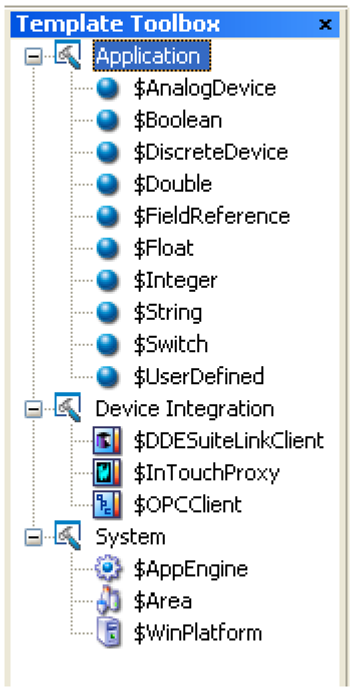
Icon	Name	Equivalent Menu Bar option
	Galaxy Status	**Galaxy Status** on **Galaxy** menu
	Template Toolbox	
	Application Views	
	Help Topics	**Help Topics** on **Help** menu

The availability of the above icons depends on which part of the IDE's Main Window is in focus, whether a particular action is allowed, or whether something changed in the configuration environment.

Template Toolbox

This part of the Main Window lists template toolsets, which contain object templates. The **Template Toolbox** contains a tree view of template categories in the Galaxy. Double-click a category to open that toolset and see the object templates it contains.

By default, the template category tree view is collapsed. After you log in to a Galaxy, the **Template Toolbox** is loaded with the categories open from the last login session.



A new Galaxy is automatically populated with the templates shown above.

All template names are preceded by a dollar sign ($). Each template provides a unique set of functionality you can use as a building block in creating your application.

Use templates to create object instances. Each template has a set of attributes that are transferred to those instances. Each attribute has a default value that is also transferred. In some cases, attribute values can be reconfigured in the instance. This template-instance relationship is central to the robust and reusable nature of the IDE.

Shortcut Menu for the Template Toolbox

The shortcut menu for templates includes the following (see Menu Bar for descriptions of these options):

- **Open**
- **Open Read-Only**
- **Check Out**
- **Check In**
- **Undo Check Out**
- **Override Check Out**
- **Validate**
- **New**
    - **Instance**
    - **Derived Template**

- **Delete**
- **Rename**
- **Assign To**
- **Unassign**
- **Export**
    - **Automation Object(s)**
    - **Galaxy Dump**
- **Object Help**
- **Properties**

Application Views

The **Application Views** pane shows the object-related contents of the Galaxy in three different ways: **Model**, **Deployment**, and **Derivation**. The **Model View** is the default view when the IDE is first opened.

Note In each **Application View**, the names of the default objects are in bold type. The default objects are a single Area, WinPlatform, and AppEngine that newly-created instances are assigned to (**Deployment View**) or associated with (**Model View**). The assignment or association is based on the kind of instance you create. For example, creating an instance of an ApplicationObject automatically associates it with the default Area. Creating an instance of an AppEngine automatically assigns it to the default WinPlatform and associates it with the default Area. You can reassign the new instance by dragging and dropping it onto another object.

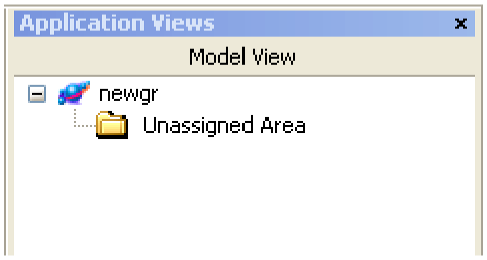
For each object shown in any view, the follow conditions are listed:

- The object's deployment status:
    - not deployed
    - deployed
    - pending configuration update
      The object is currently deployed and has been reconfigured since deployment.
    - pending software update.
      The object is deployed and its template code modules have been upgraded.
- The object's configuration status: good, warning or bad.
- The object's check out status: checked out or checked in to the Galaxy database.

When you switch from one **Application View** to another, the selected object in the first view is automatically selected in the second view. This is true only 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 view most accurately represents an application perspective of the your processes (for example, specific process areas, tanks, valves, pumps and their relationships based on containment).



The tree structure functions like a standard MS Explorer tree. 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 way:

- The top of the tree is the Galaxy.
- Top-level Areas are shown under the Galaxy.
- Within each Area, contained Areas are listed. Multiple levels are allowed.
- Objects that belong to an Area are listed under it.
- Objects contained by other objects are listed under their respective containers. Multiple levels are allowed.

Note Objects implicitly belong to the same Area as the object that contains them. Also, some object hierarchical 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.

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

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

To associate one object with another, drag it. To break that association, drag it to another object or to the **Unassigned Area** folder. An inappropriate assignment match is not allowed and the universal “not” symbol is shown.

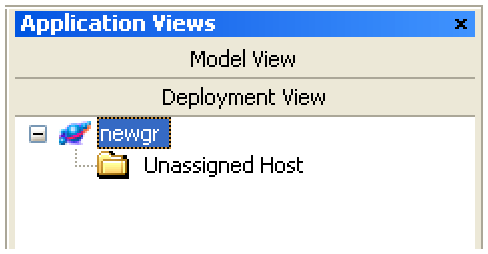
Note You can drag an object instance from any **Application View** to the search results area of the **Find** dialog box. This action lets you filter the drop target to a manageable list for doing an assignment.

For more information about object containment, see Working with Containment.

Note You must undeploy an object that is currently deployed before changing its association from one Area to another.

Deployment View

The Deployment view presents object instances (no templates) in terms of their assignment relationships, and allows the user to organize those objects through a folder structure. This view displays which objects reside on which computers and which objects host them. Since, in the ArchestrA environment, the physical location of objects is not required to approximate the real-world environment it emulates, this view does not represent your physical plant environment.



The tree structure functions like a standard MS Explorer tree, and initially is divided into two hierarchical levels: <Galaxy Name> and the **Unassigned Host** folder.

In the **Deployment View**, objects are displayed in a tree according to their distribution relationships in a multi-node system in the following way:

- 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 (for example, 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 within this view.

Important! 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. Refer to the DINetwork object’s help file for specifics on configuration limits.

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

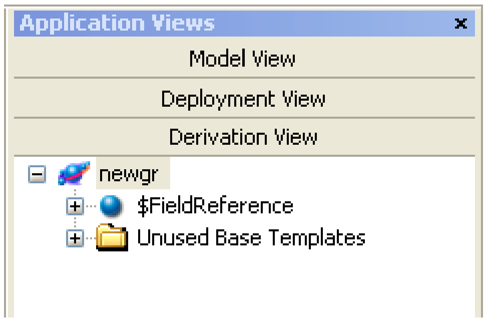
To assign an object to another, drag it onto the host object. An inappropriate assignment match is not allowed (the universal “not” symbol is shown). To unassign an object, drag it to the **Unassigned Host** folder.

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

Note You can drag an object instance from any **Application View** to the search results area of the **Find** dialog box. This action enables you to filter the drop target to a manageable list for doing an assignment.

Derivation View

The **Derivation View** presents objects and templates in terms of their ancestry. An object derived from another object appears in a hierarchy under it.



The tree structure functions like a standard MS Explorer tree, and initially is divided into three hierarchical levels: <Galaxy Name>, <Used Base Templates> (shown as $FieldReference in example above) and the **Unused Base Templates** folder. The $FieldReference template appears because it already has derived templates created for you in a new Galaxy.

In the **Derivation View**, objects are displayed according to their parent-child relationship in the following way:

- 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.
- Unused templates (no associated derived templates or instances) are grouped together in the **Unused Base Templates** folder.

Objects whose names are preceded with a “$” are templates (base or derived). Under each branch of the tree, child objects are listed in alphabetical order. Default objects are displayed in bold.

As in other views, dragging one object onto another in the **Derivation View** associates the two objects based on predefined rules of the object types.

Note You can drag an object instance from any **Application View** to the search results area of the **Find** dialog box. This action enables you to filter the drop target to a manageable list for doing an assignment.

Shortcut Menu for Application Views

The shortcut menu for the **Model**, **Deployment** and **Derivation** views is dynamic depending on which view is currently displayed. In some cases, options are not available. Right-clicking on an object displays the following shortcut menu (see Menu Bar for descriptions of these options):

- **Open**
- **Open Read-Only**
- **Check Out**
- **Check In**
- **Undo Check Out**
- **Override Check Out**
- **Validate**
- **New**
    - **Instance**
    - **Derived Template**
- **Delete**
- **Rename**
- **Rename Contained Name**
- **Deploy**
- **Undeploy**
- **Upload Runtime Changes**
- **Assign To**
- **Unassign**
- **Set as Default**
- **Export**
    - **Automation Object(s)**
    - **Galaxy Dump**
- **Object Help**
- **View in Object Viewer**
- **Properties**

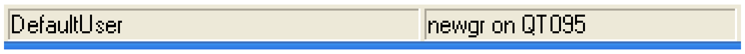
Object Editor Area

This part of the Main Window hosts the configuration editors for the objects in the **Template Toolbox** and **Application Views** panes. The contents of each editor, including options, data and shortcut menus, is provided by each individual object. Each editor provides its own documentation also, describing the function and configuration of that object.

See Working with Object Editors for more information.

Status bar

The status bar shows several different kinds of information, depending on the location of your cursor. When you point to a menu option, the status bar shows a brief description of the option. It also shows the current user name, the name of the Galaxy you are logged into and the name of the Galaxy Repository.


DefaultUser newgr on QT095

Operations Pane

The **Operations** pane shows the progress and results of a set of Galaxy database operations that can be done at the same time as other application-building operations. Currently, validating the configuration of objects is the only operation that uses this pane.

Important! Validation can be done on both templates and instances, but only on those items that are checked in.

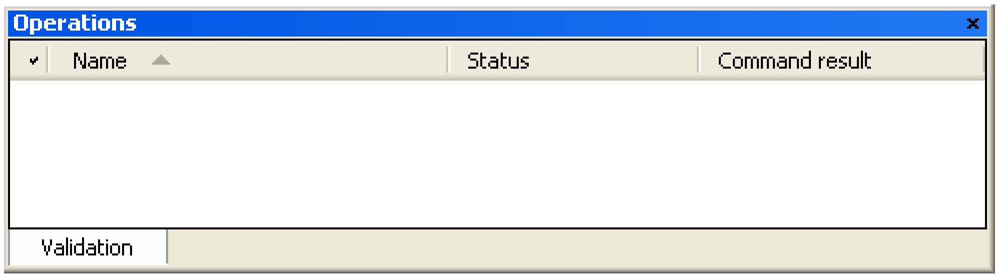
Validating an object checks the object’s configuration, including checking allowable attribute value ranges, compiling its scripts, updating and binding its references, validating its extensions, updating its status, and validating other configuration parameters that are unique to the object.

Note Use validation to validate objects that were configured before you imported any relevant script libraries. Such objects have a status of Bad. Validating these Bad objects corrects references to the script libraries and updates their status to Good.

To open the Operations pane, either

- Right-click on an object (multi-select is allowed) and click **Validate** on the shortcut menu.
- Click **Operations** on the **View** menu.

The following pane appears in the Main Window.



To close the Operations pane, click the **x** close button.

During the validation of an object, its icon and name appear with the status of the operation. The status of the object (**Status** column) is dynamically represented by an icon: no icon indicates Good status, an Error or Warning icon indicates those states. When validation is complete, the Command Result column shows either a "Succeeded" or "Failed" message, which shows additional information about the validation results.

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.

If multiple objects are validated, the list of objects is sorted by object name. Click a column heading to re-sort according to alphanumeric or icon groupings. Use the check mark column heading to sort for objects that are checked out and, therefore, cannot be validated. The object's icon indicates checked out status with a check mark.

You can perform any operation on an object listed in the **Operations** pane that is possible in the **Template Toolbox** or **Application Views**. Right-click on the object and select options from the shortcut menu.

- To open an object's editor from the **Operations** pane, double-click the object.
- To view an object's properties (particularly, the **Errors/Warnings** page of the **Properties** dialog box), double-click its status icon.
- To copy a line of text in the Operations pane list, click **Copy** on the shortcut menu or press Ctrl+C.

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

For more information about object validation, see Validating Objects.

Reorganizing Your Work Area

When you first login to the IDE, the Main Window shows the **Template Toolbox** and **Application Views** docked side-by-side on the left, the toolbar docked at the top, and the object editor area on the right. The **Operations** pane does not appear by default. When you log in again, the Main Window shows these elements you last used them. Each user sees his or her workspace.

The object editor uses all space in the IDE's Main Window that is not used by the **Template Toolbox**, **Application Views** and Operations panes.

The **Application Views**, **Template Toolbox** and **Operations** panes can stand on their own outside of the Main Window if the IDE is not in maximize mode.

Reorganize these elements to suit your needs. These elements can be moved, stacked, and closed using standard Windows move and close conventions. For more information about moving or closing elements, see the Microsoft Windows Help.

Shortcut Keys

Use the following keyboard shortcuts to quickly run commonly used options.

Keyboard Shortcut	Equivalent Menu option
CTRL+N	Create New Instance
CTRL+SHIFT+N	Create New Derived Template
CTRL+O	Open Galaxy
CTRL+F4	Close Galaxy
CTRL+S	Save Galaxy
ALT+ENTER	Galaxy Properties
F2	Rename selected object
SHIFT+F2	Rename Contained Name
DEL	Delete selected object
CTRL+F	Find
F1	Opens Help Topics
CTRL+F1	Opens Object Help
ALT+F4	Closes the Galaxy
CTRL+Z	Undo the last command
CTRL+X	Cut the selected object
CTRL+C	Copy the selected object
CTRL+V or SHIFT+INSERT	Paste the cut or copied object
SHIFT+DELETE	Delete the selected object
CTRL+A	Select All

Allowed Names/Characters

The following table summarizes rules for various names that can exist within an ArchestrA galaxy. Reference strings are included, but for more details, see Working with References. Letters noted below include letters in any language.

Note The following tagnames are reserved and cannot be used for user name, security group, role name, tagname, Galaxy name or contained name:
Me, MyContainer, MyArea, MyHost, MyPlatform, MyEngine and System.

Named item	Max length	Allowed characters	First character restriction	Other rules
Galaxy name	32	letters, digits and special characters	letter only	none
Template name	32 including $	letters, digits and special characters	$	must contain at least one letter, second character cannot be $
Instance name	32	letters, digits and special characters	not $	must contain at least one letter
Hierarchical name	329	letters, digits, special characters and a period (".")	not $	must contain at least one letter
Contained name	32	letters, digits and special characters	not $	must contain at least one letter
Static attribute name or UDA attribute name	329 (including dots)	letters, digits, special characters and a period (".")	none	must contain at least one letter
Dynamic attribute name	329	any character except a space	none	none
Property name	32	letters, digits and special characters	none	none
Toolset name	64	any printable character	not $	none
User name	255	any printable character except "User Name" and special characters	none	none
Security group	32	letters, digits and special characters	not $	must contain at least one letter
Role name	512	letters, digits and special characters	none	none
Script name	32	Letters and/or digits and/or special characters and/or ".".	none	must contain at least one letter

Named item	Max length	Allowed characters	First character restriction	Other rules
Vendor Name	-	any printable character except ? “ / \ < > * | :	-	none

Table Legend

- letter = any letter in the alphabet of any language
- digit = any numerical character
- special character = any graphics character except the following: characters whose ASCII table code is 0 through 32 (non-graphic characters) and . + - * / \ = ( ) ` ~ ! % ^ & @ [ ] { } | : ; ’ , < > ? “ space
- user name special character = \ / [ ] : | < > + = ; , ? * @ space

Node-to-Node Communications

All computers with installed ArchestrA-enabled software must be able to communicate with each other. Two main factors must be considered to ensure communication between nodes: ArchestrA user accounts and multiple NIC computers.

ArchestrA User Accounts

Communication is enabled through an ArchestrA-specific user account set up during the initial installation of each ArchestrA component on each computer, including the IDE.

WARNING! The user account that enables ArchestrA 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 properly.

If you delete the ArchestrA user account on an IDE node, you must recreate it with the Change Network Account utility.

Note You must have Administrator privileges on the computer to make changes with the Change Network Account utility.

To recreate an ArchestrA user account

1. Start the Change Network Account utility by clicking `Start`, point to `Programs`, and click `Wonderware`. Click `Common` and then click `Change Network Account`.

   Note Click **Help** on the **Change Network Account** dialog box for more information about using the options shown.

2. In a single-node ArchestrA system, create any account.

3. In a multi-node ArchestrA system, recreate the same user account with the same user name and password that was created on all other computers that have installed ArchestrA-enabled software.
4. Click **OK**.

Note After you recreate the user account in the **Change Network Account** dialog box, the Microsoft Windows security component on your computer can take several minutes to update this information on the ArchestrA Galaxy node. Until that occurs, your IDE might not function properly. Rebooting the Galaxy node applies this update immediately.

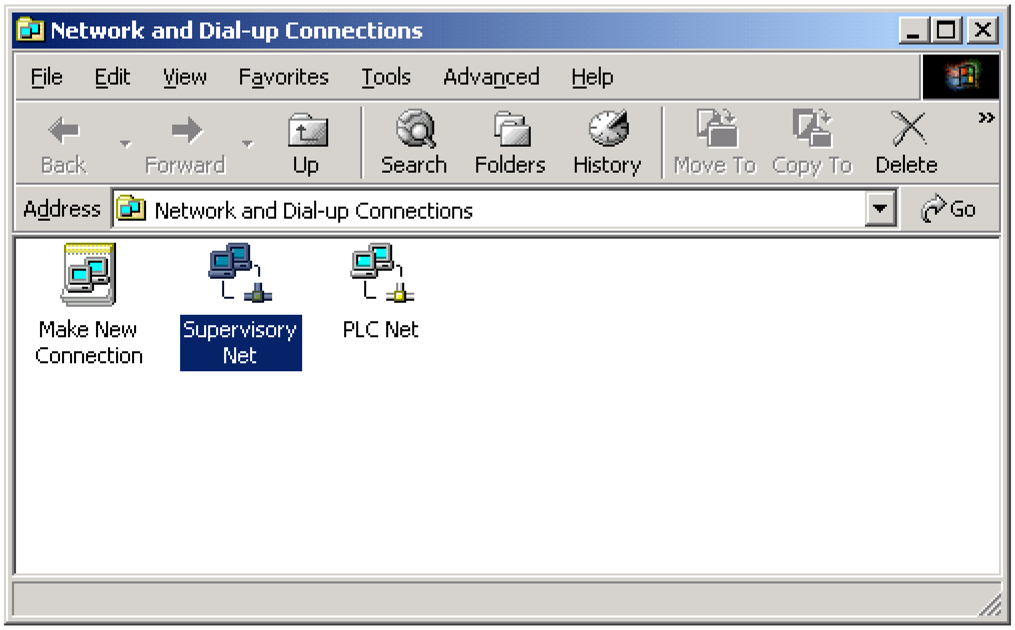
Multiple NIC Computers

Any node in your Galaxy that contains more than one network card must be configured according to the following instructions to ensure communications with other ArchestrA nodes. If a node contains multiple NICs for redundancy reasons, also see Configuring Redundancy in the WinPlatform for more information.

Important! If a multiple NIC computer in your Galaxy uses only one NIC, disable all cards except the supervisory network.

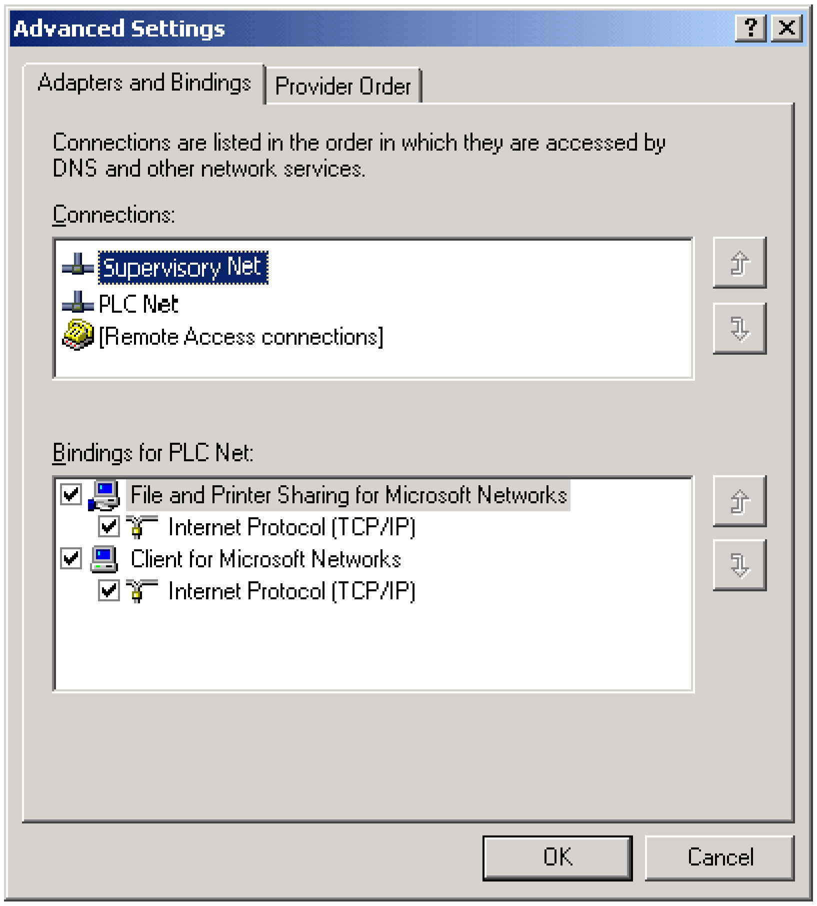
For any multiple NIC ArchestrA node to communicate successfully with all other Galaxy nodes, you must define the correct order of network connections in the network services of the computer.

Open the **Network and Dial-up Connections** dialog box (see image below) and rename each card with a clearly identifiable function (for example, “Supervisory Net” and “PLC net”). Different operating systems provide unique access to the **Network and Dial-up Connections** dialog box (in Windows 2000 Professional, click **Start**; point to **Programs**, **Accessories** and **Communications**; and then click **Network and Dial-up Connections**).


Network and Dial-up Connections
File Edit View Favorites Tools Advanced Help
Back Forward Up Search Folders History Move To Copy To Delete
Address Network and Dial-up Connections
Go
Make New Connection
Supervisory Net
PLC Net

Next, you must order the network cards properly. In the **Network and Dial-up Connections** dialog box, click **Advanced Settings** on the **Advanced** menu. In the **Advanced Settings** dialog box (see image below), use the up and down arrows to define the correct order of **Connections** and click **OK**.

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 an RMC connection for ArchestrA redundancy), the supervisory net must be listed first and the others can be listed in any other position.


Advanced Settings

Connections are listed in the order in which they are accessed by DNS and other network services.

Connections:

- Supervisory Net
- PLC Net
- [Remote Access connections]

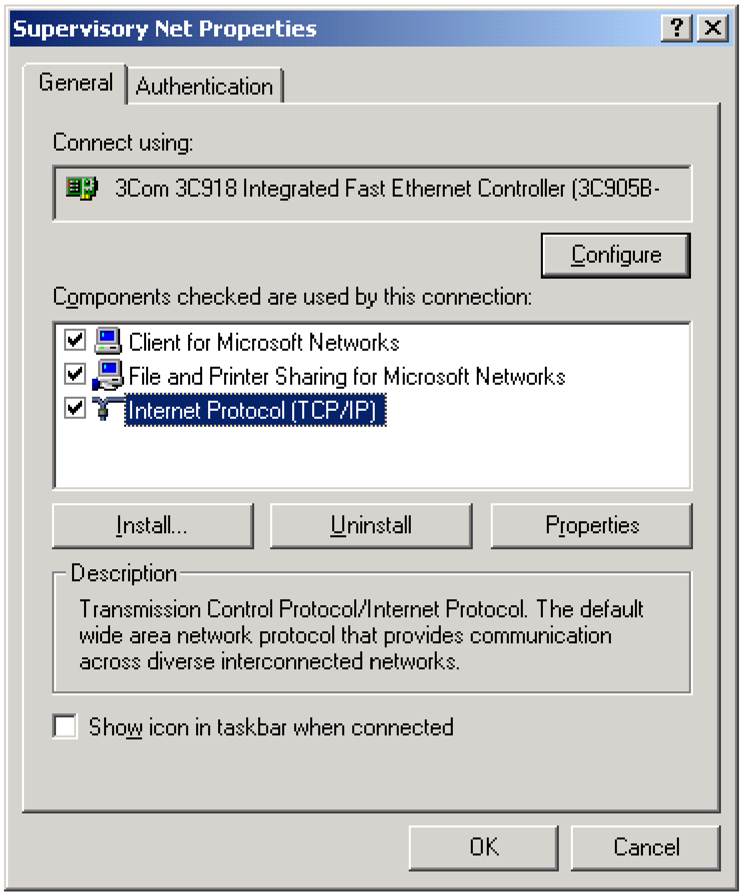
Bindings for PLC Net:

- [✓] File and Printer Sharing for Microsoft Networks
  - [✓] Internet Protocol (TCP/IP)
- [✓] Client for Microsoft Networks
  - [✓] Internet Protocol (TCP/IP)

Several other parameters must be configured to ensure successful node-to-node communications in the ArchestrA environment. You must configure the IP address and DNS settings as follows for each network card to function properly.

Do the following for each network connection

1. In the **Network and Dial-up Connections** dialog box, right-click the network connection and click **Properties** in the shortcut menu. The connection’s **Properties** dialog box (see image below) appears.


Supervisory Net Properties

General | Authentication

Connect using:

3Com 3C918 Integrated Fast Ethernet Controller (3C905B-

Configure

Components checked are used by this connection:

- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] Internet Protocol (TCP/IP)

Install... | Uninstall | Properties

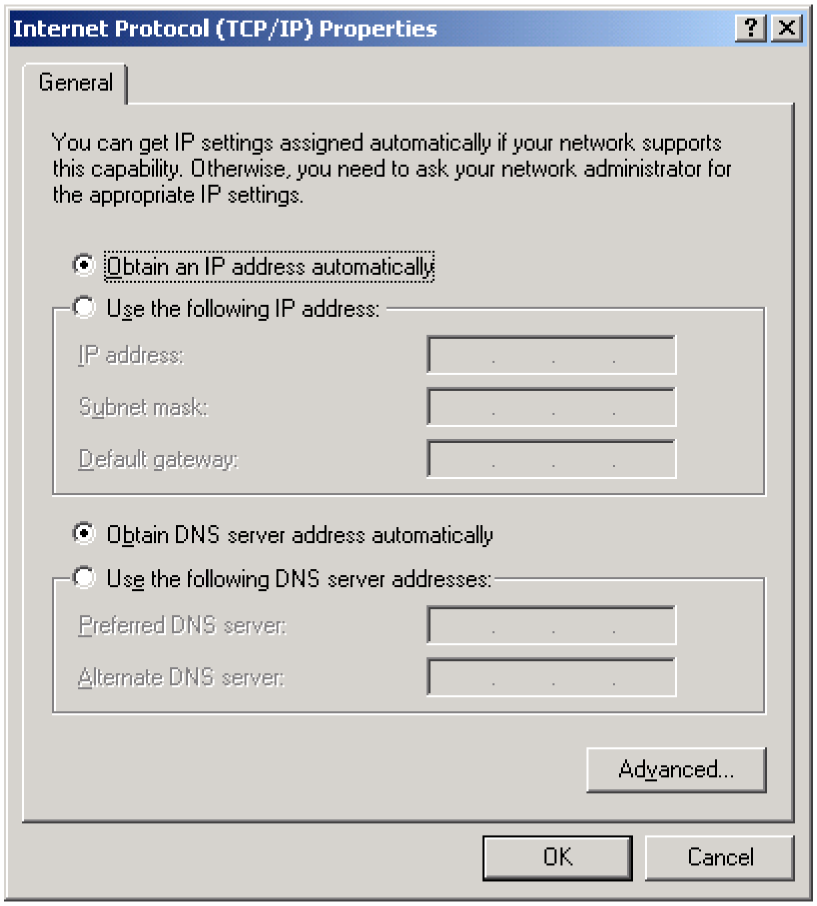
Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

[ ] Show icon in taskbar when connected

OK | Cancel

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 (see image below) appears.


Internet Protocol (TCP/IP) Properties

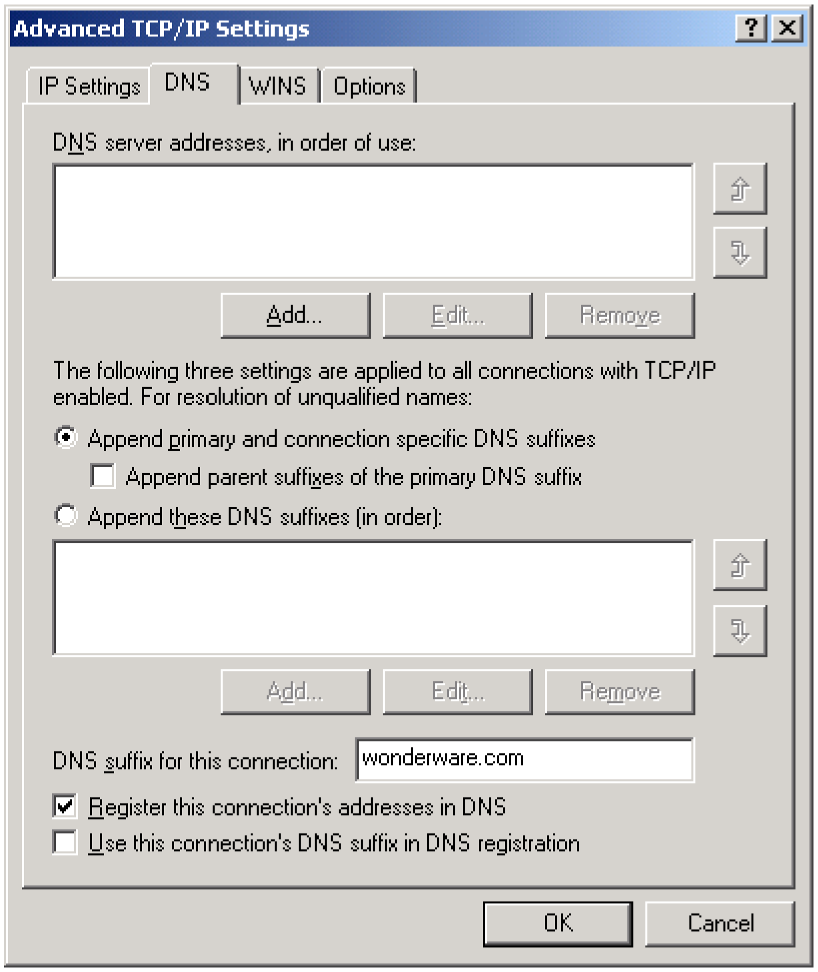
General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

[✓ Obtain an IP address automatically]
Use the following IP address:
IP address: . . .
Subnet mask: . . .
Default gateway: . . .

[✓ Obtain DNS server address automatically]
Use the following DNS server addresses:
Preferred DNS server: . . .
Alternate DNS server: . . .

3. For the supervisory network, select **Obtain an IP address automatically**. For the other network connections, select **Use the following IP address**. Consult your network administrator for the proper settings for the remainder of the parameters in this group.
4. Click **Advanced**. The **Advanced TCP/IP Settings** dialog box (see image below) appears. Click the **DNS** tab.



DNS server addresses, in order of use:

The following three settings are applied to all connections with TCP/IP enabled. For resolution of unqualified names:

[✓ Append primary and connection specific DNS suffixes]

☐ Append parent suffixes of the primary DNS suffix

Append these DNS suffixes (in order):

DNS suffix for this connection: wonderware.com

☑ Register this connection's addresses in DNS

☐ Use this connection's DNS suffix in DNS registration

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**.

Minimum Disk Space Requirements

After Industrial Application Server is installed on your system, certain operations require a minimum of 100 MB of free disk space. These operations include creating a Galaxy, deploying objects, importing and exporting objects, loading and dumping a Galaxy, and 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 free disk space, running any of these operations can result in erratic behavior.

C H A P T E R 2

Working with Objects

Objects in the ArchestrA environment can be categorized into two groups:

- templates
  Templates are the building blocks of your Galaxy application.
- instances.
  Instances created from templates represent the unique, real-world components of your Galaxy application. Instances must be configured and assigned before deployment to target nodes in your application.

This chapter describes how to create, import, configure and manage object templates and instances in the IDE as well as how to manage toolsets in the **Template Toolbox** of the IDE.

Contents

About Objects

The IDE can use any object created with the ArchestrA Object Toolkit. This includes all templates, derived templates and instances derived from templates.

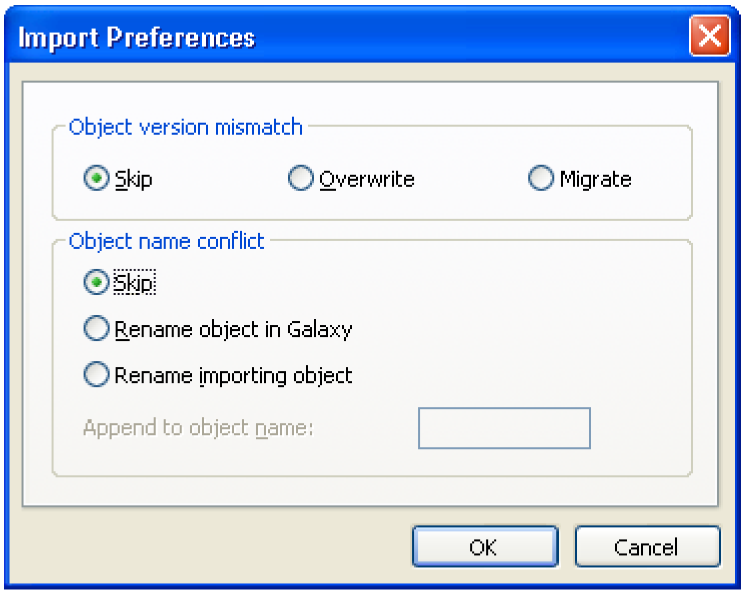
Importing Objects

To use an object, you must import it first into your Galaxy using the import feature of the IDE.

Note You can also import instances previously exported from a Galaxy. Their previous associations are retained, when possible, such as assignment, containment, derivation and area.

Objects are imported from two kinds of ArchestrA-specific package files, typically with .aaPKG and .aaPDF extensions. More than one object can be contained in a package file.

The Galaxy database does not allow two objects with the same name or more than one copy of the same version of an object. When you import an object (see “To import an object” procedure below), use the **Import Preferences** dialog box to handle version upgrades and naming conflicts.


Import Preferences

Object version mismatch: (● Skip) ○ Overwrite ○ Migrate

Object name conflict: (● Skip) ○ Rename object in Galaxy ○ Rename importing object

Append to object name:

Use the **Object Version Mismatch** group to determine whether or not to import a version upgrade of an object that already exists. A version upgrade can be the result of a MinorVersion or ConfigVersion change.

- Select **Skip** to cancel the import, leaving the existing object unchanged,
- **Overwrite** to replace the existing object with the version upgrade of that object, or
- **Migrate** to import the selected object(s) that have gone through a major version upgrade.

The latter can occur after installation of a newer version of Industrial Application Server or when you are importing a third-party object that has gone through a major upgrade. If your Galaxy has not been upgraded and you select **Migrate**, the import operation behaves like **Skip**

Important! If an object is skipped due to a version mismatch conflict, the process continues importing any objects that remain in the package file that do not have a version mismatch.

Use the **Object Name Conflict** group to determine which of three name collision resolutions are applied if a different object of the same name as the imported object is found in the Galaxy.

- Select **Skip** to not import the object when a name match exists in the Galaxy.
- Select **Rename Object in Galaxy** or **Rename Importing Object** to continue importing the new object while resolving the naming collision.
- Select **Rename Object in Galaxy** to rename the existing object by appending to its current name the string (up to four characters) you type in the **Append to Object Name** box (default value is `_old`).
- Select **Rename Importing Object** to rename the object being imported by appending to its current name the string (up to four characters) you type in the **Append to Object Name** box (default value is `_new`).

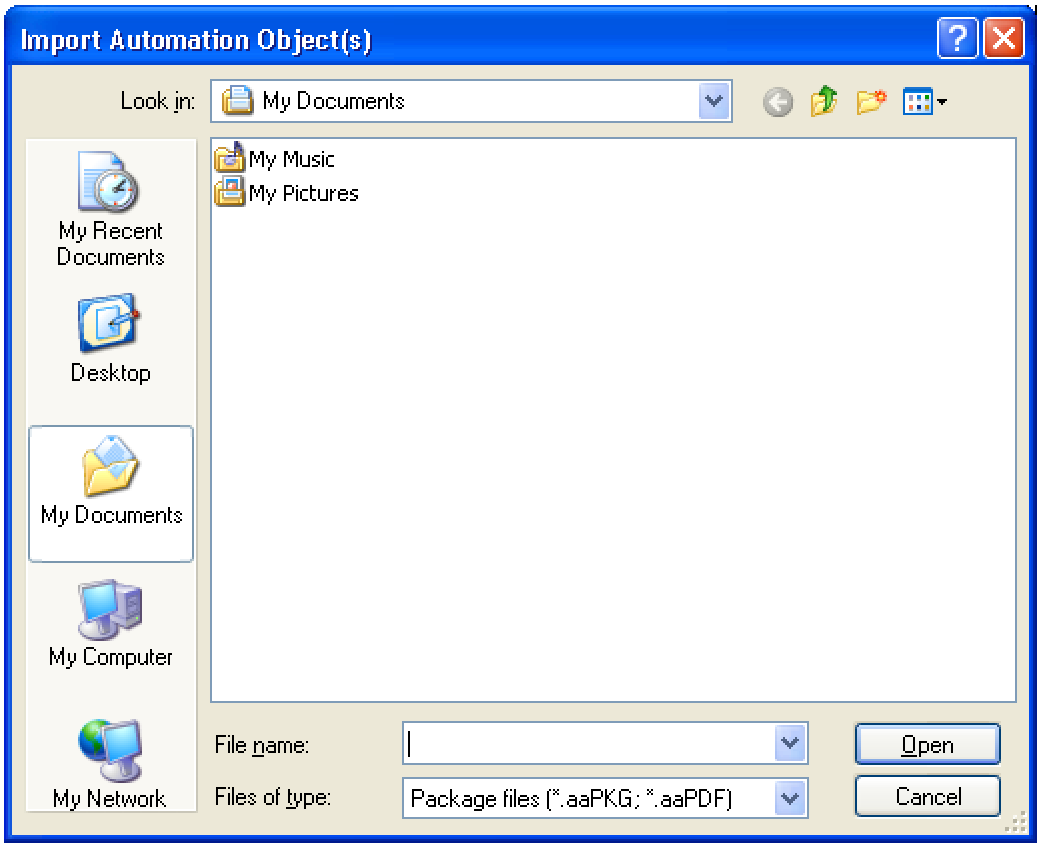
Note Object name conflict resolution applies only to templates and instances derived from different base templates (different code base).

During the import process, a progress box shows status messages indicating whether each object is imported successfully, skipped, renamed or overwritten.

Important! An object's functionality can be enhanced by attaching a script to it. Some scripts include some 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 fix this condition, import the script function library and validate the object. See Script Functions for information about importing script function libraries and Validating Objects for information about validation.

To import an object

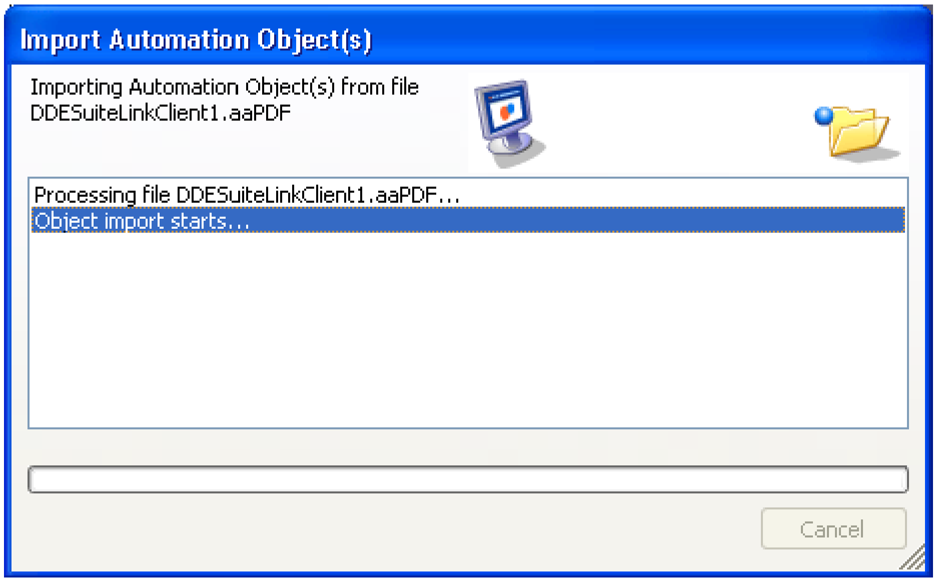
1. On the **Galaxy** menu, point to **Import** and click **Automation Object(s)**. The **Import AutomationObject(s)** dialog box appears.



File name:

Files of type: Package files (*.aaPKG; *.aaPDF)

2. Browse for an object file (`.aaPKG` or `.aaPDF` extension). You can multi-select more than one file. Click **Open**. The **Import Preferences** dialog box (see image and description previously) appears.
3. Set the behavior of the import process and click **OK** to continue. A progress dialog box appears.



Importing Automation Object(s) from file DDESuiteLinkClient1.aaPDF

```
Processing file DDESuiteLinkClient1.aaPDF...
Object import starts...
```

4. When the import process finishes, click **Close**.

Imported templates appear in the proper toolset in the **Template Toolset** as defined in the object. Imported instances appear in the **Application Views**. The following post-import rules apply:

- If the 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.

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.

Exporting Objects

Use the IDE's export function to share objects with other users or to recreate them in other Galaxies. The resulting file (`.aaPKG` extension) contains the selected objects, their associated predecessor templates and the configuration state of those objects. The export file can later be imported into the same or another Galaxy.

Subsequent exports retain the default folder as last used for the duration of the IDE session. If the designated file already exists, you are prompted to confirm overwrite.

If an object selected for export is checked out, the checked-in version of that object is exported instead.

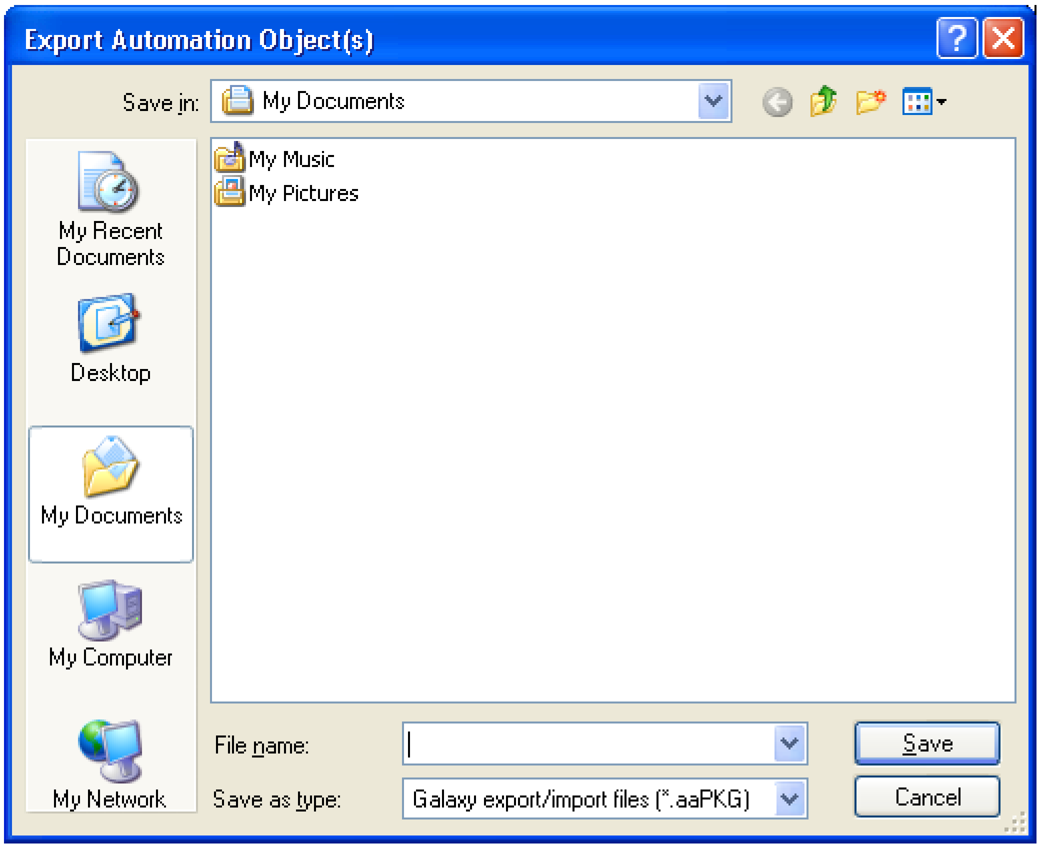
Important! An object's functionality can be enhanced by attaching a script to it. Some scripts include some functions that depend on external files called script function libraries. Scripts are included in the object import operation but you must import the script function libraries separately.

Exporting an entire Galaxy is similar to using the Galaxy Database Manager utility to back up the database. Unlike backups, during an export the change logs for the objects are not exported When you export objects, only the security information for the specific object is exported.

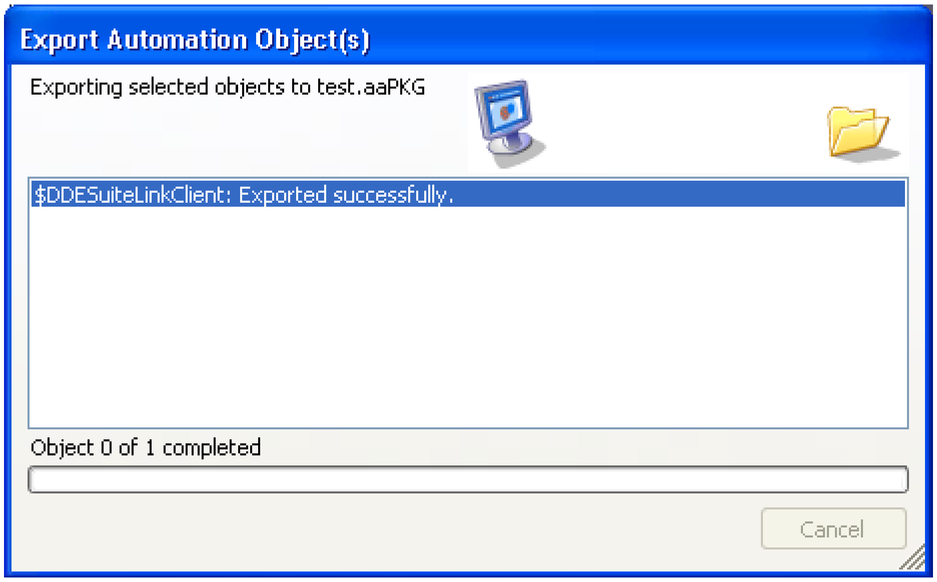
To export an object

1. Select an object in the **Template Toolbox** or **Application Views** pane.
2. Point to **Export** on the **Galaxy** menu and then click **AutomationObject(s)**. The **Export AutomationObject(s)** dialog box appears.

   Note You can export more than one object with this function by first multi-selecting objects (**Shift+Click** or **Ctrl+Click**). If you want to export all of the objects in the Galaxy, point to **Export** and then click **All AutomationObjects**.



3. In the export dialog box, browse to the path and filename (`.aaPKG` extension) of the export file and click **Save**. A progress box appears.



Exporting selected objects to test.aaPKG

$DDESuiteLinkClient: Exported successfully.

Object 0 of 1 completed

4. When the export process finishes, click **Close**. You can use the resulting `.aaPKG file` to import the selected objects into another existing Galaxy.

Important! Export maintains previously specified containment relationships. Also, if an object is currently checked out, the last checked in version of the object's configuration is exported.

Configuring Objects

Templates and instances are configured in the same way: in their configuration editors. To configure an object, select it and click **Open** on the **Galaxy** menu. The editor appears in the object editor area of the IDE's Main Window.

See Working with Object Editors for more information about configuring template editors.

Begin by checking out the object, which is done automatically when you open its editor. After you are finished configuring it, check the object in to the Galaxy database.

Checking Objects In and Out

When you want to make changes to an object, it must be checked out. Then you can modify the object and save private versions of it before releasing it to the Galaxy (checking the object in) for other users to see and use. You can also back out changes made to the object through the undo check out feature.

Note All IDEs connected to a Galaxy show current status for each object in the Galaxy. Change history for each object can be reviewed.

If you select an object and that object is already checked out by another person, the check out function is disabled.

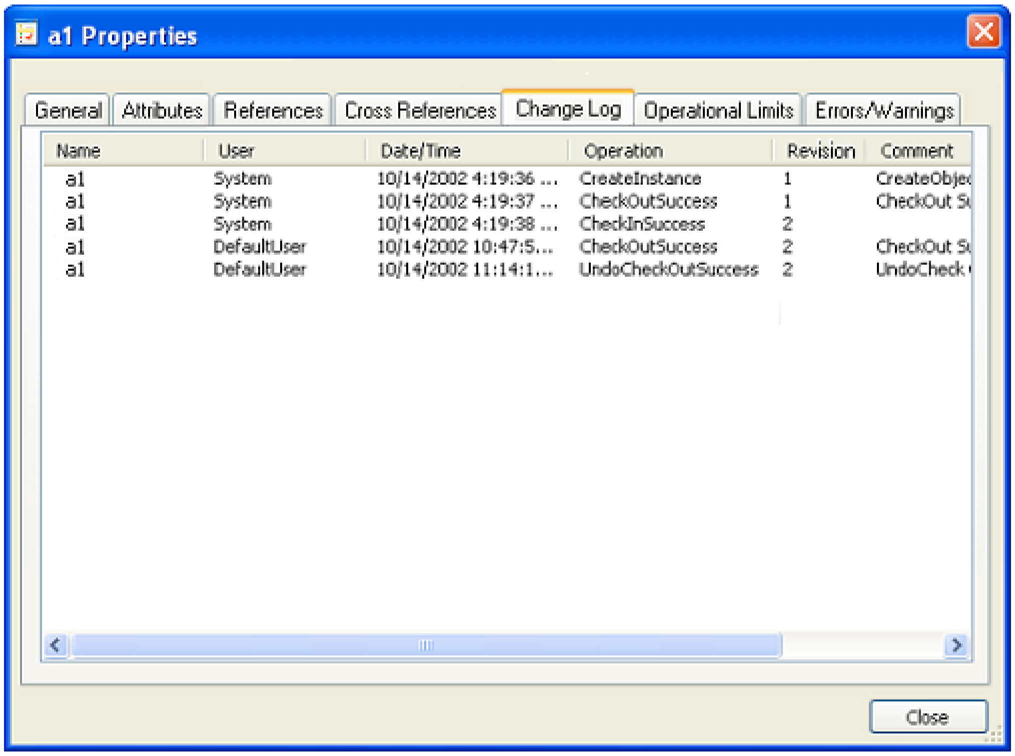
You can multi-select more than one object for checking out. If you use multi-select and any of the objects you attempt to check out are already checked out to other people, then a dialog box appears indicating their status. Also, if some of objects you attempt to check out are already checked out to you, the check out function is disabled.

The Galaxy marks the objects as checked out to you, making them unavailable for check out to other users, and it updates the object's revision history. A check mark is shown next to an object's icon in the IDE.

To check out unreserved objects

1. Select them in the **Template Toolbox** or **Application Views**.
2. On the **Object** menu, click **Check Out**. Optionally, an object is automatically checked out to you when you open its configuration editor.

   If the object is already checked out by another user, you can open its editor only in read-only mode. To determine an object's status and history, open the **Properties** dialog box.



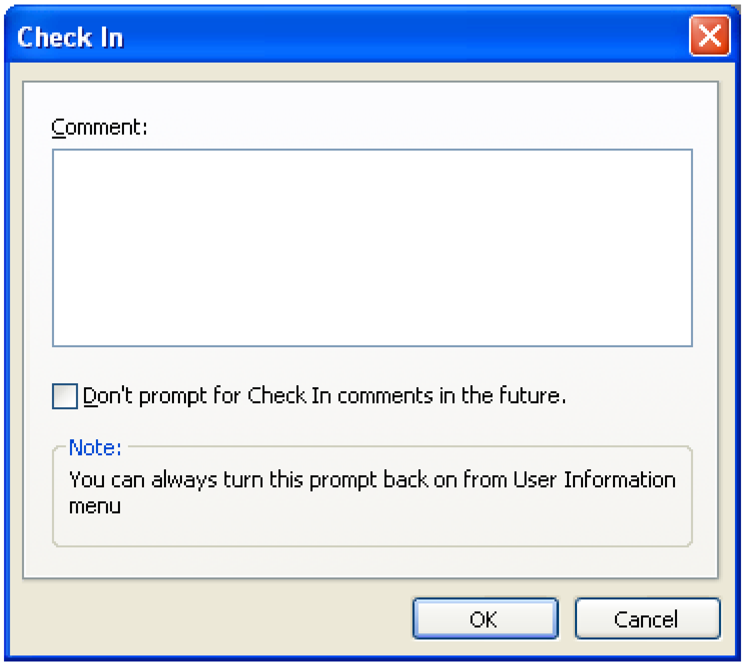
Name	User	Date/Time	Operation	Revision	Comment
a1	System	10/14/2002 4:19:36 ...	CreateInstance	1	CreateObjec
a1	System	10/14/2002 4:19:37 ...	CheckOutSuccess	1	CheckOut Su
a1	System	10/14/2002 4:19:38 ...	CheckInSuccess	2	
a1	DefaultUser	10/14/2002 10:47:5...	CheckOutSuccess	2	CheckOut Su
a1	DefaultUser	10/14/2002 11:14:1...	UndoCheckOutSuccess	2	UndoCheck

The user responsible for an operation at a specific date and time is listed on the **Change Log** page. Comments typed by a user in the **Check In** dialog box are listed under the **Comment** heading.

To check an object in to the Galaxy database

1. Select the object in an **Application View** or the **Template Toolbox** and, on the **Object** menu, click **Check In**. The **Check In** dialog box appears.

Note If you close the editor without making any changes to the object's configuration, the object is automatically checked back in to the Galaxy. See "Working with Object Editors" for more information about these functions.



Don't prompt for Check In comments in the future.

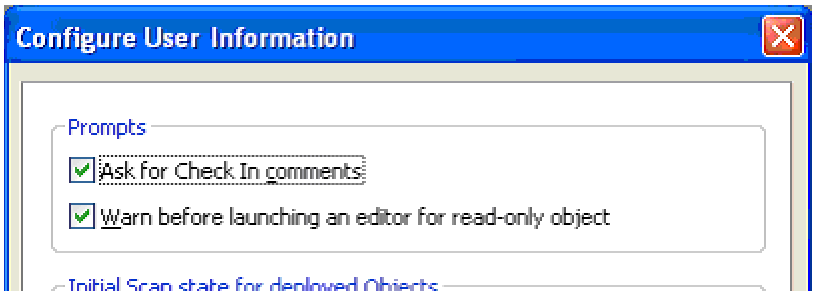
Note:
You can always turn this prompt back on from User Information menu

2. Enter a comment (optional) and click **OK** to finish checking in the object.

The Galaxy indicates whether any of the objects you are attempting to check in are checked out to other users or whether you have the object open in the editor. If an object you are attempting to check in already is checked in, check in is ignored.

The **Check In** dialog box enables you to provide comments about configuration changes made while the object was checked out. It consists of the following options:

- **Comment**: Enter your comments about configuration changes made to the object.
- **Don't Prompt for Check-In Comments in the Future**: Select this check box to turn off the comment feature when checking in objects in the future. To reinstate this feature, click **User Information** on the **Edit** menu and select **Ask for Check In Comments** in the **Configure User Information** dialog box.


Configure User Information

Prompts

✓ Ask for Check In comments

✓ Warn before launching an editor for read-only object

Initial Scan state for deployed Objects

Undo Checkout, Override Check Out

Two other IDE commands related to the concept of check out and check in include:

- **Undo Check Out**: Use this command to change an object's status from checked out to checked in. Afterwards, any user can check out and configure the object. **Undo Check Out** places a notation in the object's change log. Changes you made to the object when it was check out are backed out. An error message appears when the object's configuration editor is open.
- **Override Check Out**: Use this command to disable the checked out flag on the selected object. This command typically requires special security permissions (see "Working with Security"). Use it only in those circumstances in you are certain that object configuration is not being done by the user who originally checked out the object. If the object's editor is currently open, the override function fails.

Deleting an Object

Both templates and instances can be deleted from the Galaxy with the following exceptions. The object is:

- A deployed instance.
- A parent template (derived templates or instances are created from it).
- A container for other objects.
- Checked out by another user.

Only objects that can be deleted can be deleted from the Galaxy.

To delete an object from the Galaxy

1. Select the object from the **Template Toolbox** or **Application Views** pane. Multiple-object selection is allowed by using **Shift**+**Click** or **Ctrl**+**Click**.
2. On the **Edit** menu, click **Delete**.

Renaming an Object

An object can have up to three names depending on whether it is contained by another object. The three names include:

Name	Description
Tagname	The name of the individual object. For example, `Valve1`.
Hierarchical name	The name of the object within the context of its container object. For example, `Tank1.OutletValve`, where an object called `Tank1` contains the `OutletValve` object.
Contained name	An alias for the hierarchical name that is usually shorter in length. For example, `OutletValve`.

Besides changing 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 "Working with Containment" for more information about these naming conventions in the ArchestrA environment.

While you are renaming an object, it is automatically checked out. Until the renaming is complete and the object is checked back in, it cannot be deployed.

To rename an object's tagname

1. Select the target object.
2. On the **Edit** menu, click **Rename**. The object name is placed in edit mode.
3. Rename the object as desired and press **ENTER**.

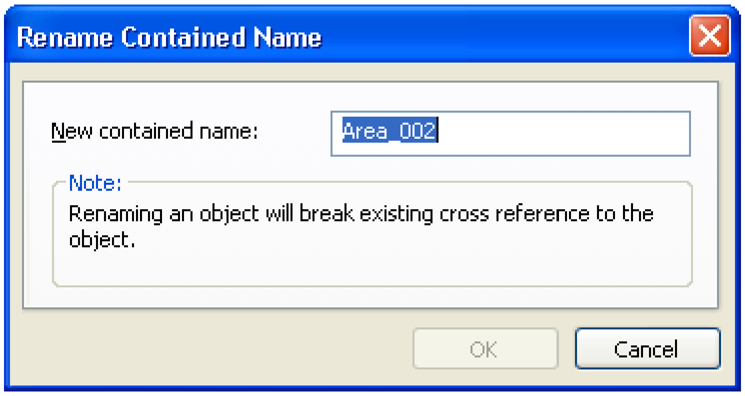
The new object name cannot be the same as another object in the Galaxy. It must also comply with naming restrictions. See "Allowed Names/Characters."

WARNING! References from other objects to the object being renamed are broken. Objects with broken references receive bad quality data at runtime.

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

To rename an object's contained name

1. Select the target object.
2. On the **Edit** menu, click **Rename Contained Name**. The **Rename Contained Name** dialog box appears.



Renaming an object will break existing cross reference to the object.

3. Rename the contained name and click **OK**.

The new contained name must be unique in the context of its container only and it must comply with naming restrictions. See "Allowed Names/Characters."

After renaming, all IDEs connected to the Galaxy reflect the new instance's contained name.

Enhancing an Object's Functionality

You can enhance the original functionality of an object through the object enhancement pages common to all object configuration editors. On the **Scripts** page, you can create, attach and set triggers for running scripts. On the **UDAs** page, you can add new attributes to the object for historizing, alarming and other functions. On the **Extensions** page, you can extend the input/output, input, output, alarm and history capabilities of the current set of attributes in an object.

All of these object enhancements can be added to the object's functionality without affecting its original capabilities. See "Enhancing an Object's Functionality" for more information about this feature.

Templates

Templates provide a unique set of functionality you can use as a building block in creating your Galaxy application. Use templates to create object instances that represent real-world device classes or groups in your application. Template names are preceded by a dollar sign ($).

Each template has a set of attributes and default values that are transferred to and, in some cases, reconfigured in those object instances. This template-instance relationship is central to the robust and reusable nature of the applications created with the IDE.

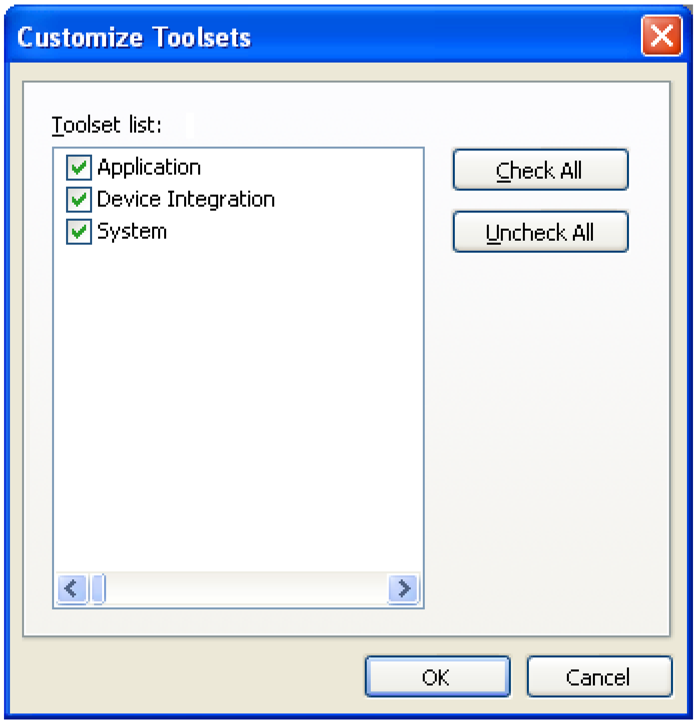
Managing Templates with Toolsets

Templates are organized in toolsets shown in the **Template Toolbox**. The IDE provides options for managing the toolbox and its content.

Show or hide the **Template Toolbox** by selecting the **Template Toolbox** icon on the toolbar. Hiding the **Template Toolbox** and the **Application Views** panes provides more work area for object editors.

You can drag templates from one toolset to another.

To manage the contents of the **Template Toolbox**, point to **Configure** on the **Galaxy** menu and click **Customize Toolsets**. The **Customize Toolsets** dialog box appears.



☑ Application
☑ Device Integration
☑ System

This dialog box shows all of the toolsets currently in the Galaxy and allows you to show and hide toolset categories from the **Template Toolbox**. The checked toolsets are currently shown in your IDE.

Note The **Template Toolbox** that appears for an individual user of the IDE can be unique. Other IDE users have different toolsets appear. If two users are showing the same toolset, though, the contents of that toolset is the same for both users.

Select unchecked toolsets and clear check boxes for others as needed. Use **Check All** and **Uncheck All** to select or clear all the available options. Click **OK** to set the contents of the **Template Toolbox** as shown.

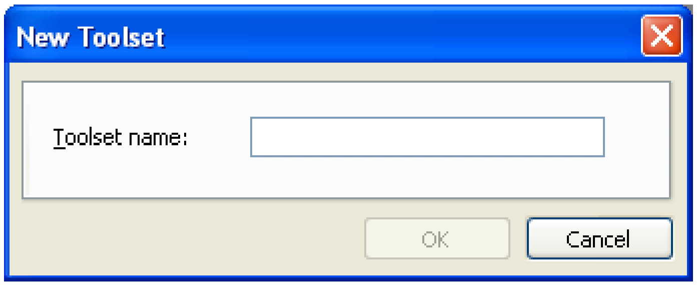
Selected toolsets and the templates within them appear in the **Template Toolbox**. Click the plus sign of a toolset to show its templates.

Important! Toolsets hidden from the **Template Toolbox** are not deleted from the Galaxy. They are only hidden from view.

Creating Toolsets

To create a new toolset in the Template Toolbox

1. On the **Galaxy** menu, point to **New** and click **Template Toolset**. The **New Toolset** dialog box appears.



2. Type a name for the new toolset.
3. Click **OK** to continue.

A new toolset appears and is in focus in the **Template Toolbox**. You can now move templates into the new toolset.

Note The Galaxy validates the toolset name, particularly whether the name already exists. If a naming conflict occurs, you are prompted to enter another toolset name.

Deleting Toolsets

To delete a toolset from the Template Toolbox

1. Select the target toolset.
2. On the **Edit** menu, click **Delete**.

   Important! You must move or delete all templates from a toolset before you can delete it. If the target toolset has templates in it, the **Delete** command is unavailable.

3. Respond appropriately to the message. Click **Yes** to delete the toolset.

Deriving a Template from Another Template

Templates can represent a wide variety of field devices. For example, the DiscreteDevice base template can represent pumps, valves and any other process device with discrete on/off functionality. When creating your Galaxy application, creating derived template groups of devices before creating instances is a good strategy.

For example, your plant processes can use several models of a pump made by a single vendor. Each model has unique characteristics that map to different attributes values of the DiscreteDevice base template. A good strategy is to create derived templates for each model from the base template. Preferably, you create a hierarchy of derived templates until you reach logical endpoints. Then create instances from each unique derived template.

To derive a template from another template

1. Select the base template in the **Template Toolbox** or **Application Views** pane.
2. Point to **New** on the **Galaxy** menu 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 as needed.

Note If the original template is a template that contains other templates, derivation also creates a derived template for each of the contained templates within the original template. The new derived templates have, by default, the same containment relationship and contained name as the original template.

The new template is an exact copy of its parent with the following possible exceptions: locking and security. See “Working with Object Editors” for more information about these functions.

Locked, Unlocked Template Attributes

A key concept of the base template-derived template-instance relationship is the attribute lock and unlock function. Typically, the object developer locks certain attributes to maintain the integrity of the object as derived templates and instances are created and configured by IDE users. Given the architecture of an object, some attributes are intended to be configured one time and then locked.

See “Working with Object Editors” for more information about using locking and unlocking while configuring template editors.

Locking an attribute in a template indicates that its value is to be logically shared with all derived objects (templates and instances). When the value changes in the template, it is propagated to the attribute in all descendents. When an attribute is locked in the template, the value can be changed in that template but not in any of the derived children.

Attribute names link attributes within templates with those in the child objects. After an attribute is locked, its name and existence are locked and cannot be changed in the children. An object cannot be modified during configuration in such a way that it introduces a name conflict with an existing attribute.

There are three logical lock types:

Lock Type	Description
Unlocked	Both templates and instances can have these. The attribute is read/write. The object has its own copy of the attribute value (it is not shared by derived objects).
Locked (in me)	Only templates can have these. The attribute value is read/write. Derived objects do not have a unique copy of this attribute, but instead share the locked one in the parent (they are Locked In Parent). Changing the value of a locked attribute in a template updates the value of that attribute in all derived objects.
Locked (in parent)	Both templates and instances can have these. The attribute is read-only. The object does not have a unique copy of this attribute, but instead references the one in the ancestor in which the attribute is locked.

Note Locking a UDA during configuration makes its value a constant. Locked UDAs cannot be written to at runtime.

An Example of Locking an Attribute

1. Derive a template from another template and call it $Valve.
2. Edit the $Valve template and set an attribute value and then lock that attribute so that derived templates and instances cannot modify the value of this attribute. Do this by clicking lock icon for the attribute.
3. Save $Valve.
4. Derive a template from $Valve and call it $BigValve.
5. Create an instance from $Valve called Valve1.

In the editor of $Valve, the attribute lock icon indicates that the attribute is locked (in me).

The attribute value in either $BigValve or Valve1 cannot be changed. The editor options for the attribute is disabled and the lock icon (if shown) indicates locked in parent. Also, the attribute lock icon in children derived from $Valve previously is now locked and disabled.

If you change the attribute value in $Valve, the change is propagated to $BigValve and Valve1.

An Example of Unlocking an Attribute

1. Assume the endpoint objects in the previous example.
2. In the $Valve template's editor, unlock the locked attribute.
3. Save $Valve.

In the editor for $Valve, the attribute lock icon indicates that 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 now indicates unlocked but the locking icon is unavailable.

Instances

Instances are the ultimate building blocks of your Galaxy application. After you create unique instances from base and derived templates, you must deploy them to begin executing their functionality.

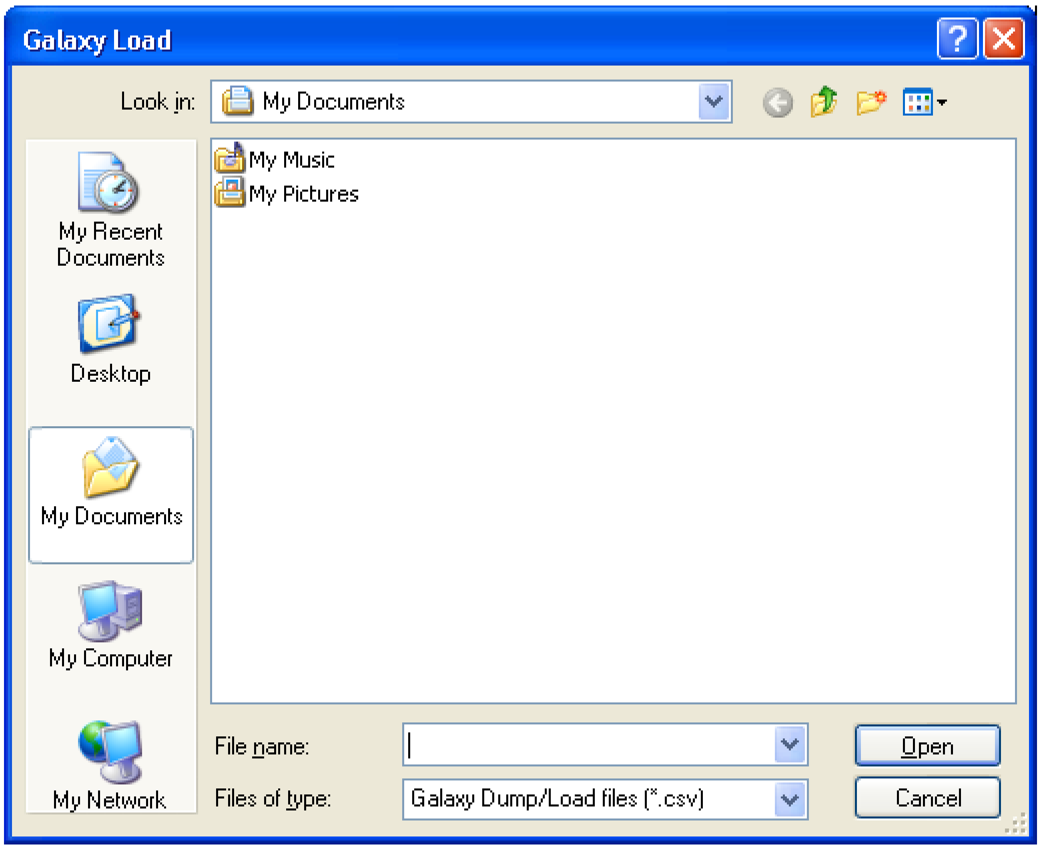
Importing a Galaxy Load File

Object instances and their configuration data in an existing Galaxy can be exported to a comma-delimited format Galaxy dump file (`.csv` extension). A `.csv` file can be edited in most text editors and spreadsheet programs. Using editing functions like copy and paste, you can create quantities of already-configured objects to import into your Galaxy.

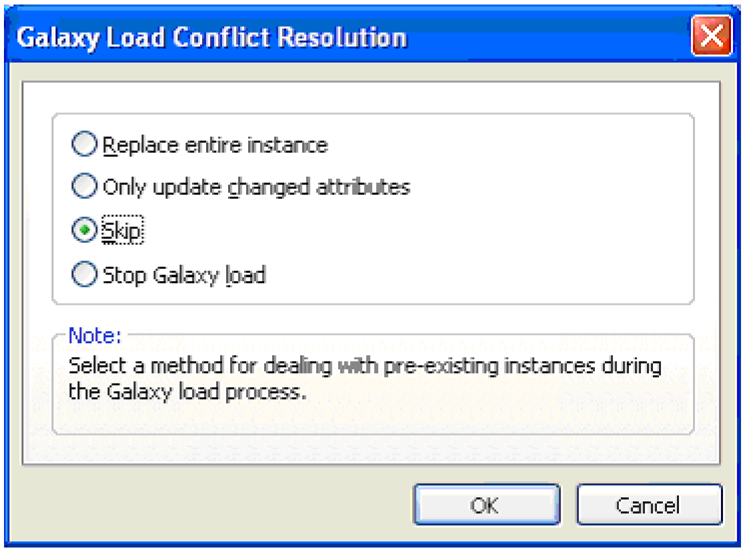
Important! The contents of the `.csv` file is determined by the original Galaxy dump. A load file contains only instances. Templates cannot be dumped and loaded. See Exporting a Galaxy Dump File for more information about the contents of the `.csv` file.

To import a `.csv` file

1. Point to **Import** on the **Galaxy** menu and then click **Galaxy Load**. The **Galaxy Load** dialog box appears.



2. In the **Galaxy Load** dialog box, browse to locate the `.csv` file that contains the objects and configuration data you want to import. Select the file and click **Open** to continue.
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. Select the preferred conflict resolution and click **OK**. A progress box appears during the Galaxy load process.


Galaxy Load Conflict Resolution
Replace entire instance
Only update changed attributes
Skip
Stop Galaxy load
OK
Cancel

[✓ Skip]

Note:
Select a method for dealing with pre-existing instances during the Galaxy load process.

The **Galaxy Load Conflict Resolution** dialog box consists of the following options:

- **Replace Entire Instance**: Select this option 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**: Select this option if an instance of an object with the same name already exists and you want to replace only the attributes of the object where the values are different.
- **Skip**: Select this option if an instance of an object with the same name already exists and you want to leave the version intact that is already in the Galaxy.
- **Stop Galaxy Load**: Select this option if an instance of an object with the same name already exists and you want to cancel the entire Galaxy Load operation.

After the load operation, all objects changed or created during the Galaxy Load process are checked in.

Important! 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 quote marks.

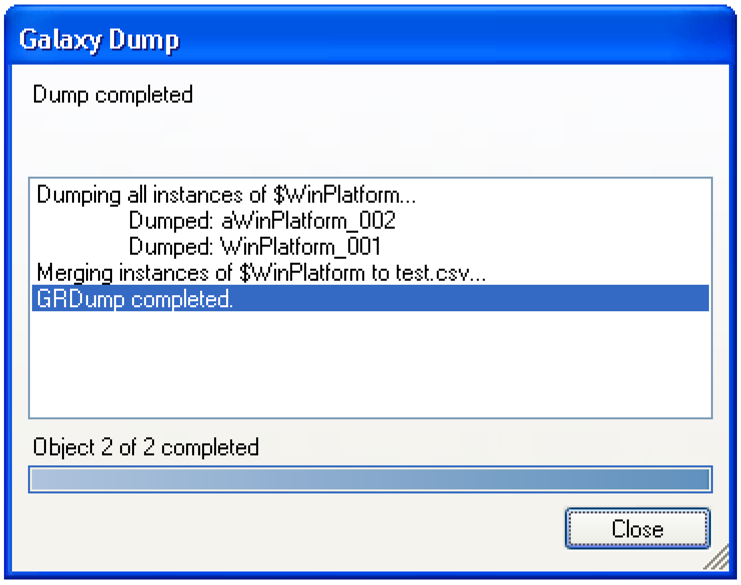
Exporting a Galaxy Dump File

Object instances and their configuration data can be exported to a comma-delimited format Galaxy dump file (`.csv` extension).

Important! Exporting only dumps instances. Templates cannot be dumped. The `.csv` file contains the configuration for the checked in versions of the selected objects as well as the checked-out objects of the user who initiates the Galaxy dump. The file contains only those attributes that are unlocked and configuration time-writeable, one column per attribute. Attributes that are locked, calculated or runtime writeable only are not saved to the file. Attributes that are not text based (for example, type QualifiedStruct) are not dumped. Object Help files are not dumped. See Galaxy Dump File (.csv) Structure later.

To export objects to a Galaxy dump file

1. Select an object in the **Application Views** pane. You can export more than one instance by first multi-selecting objects (**Shift+Click**). Also, you can dump all instances derived from a template by selecting just the template.
2. Point to **Export** on the **Galaxy** menu and then click **Galaxy Dump**. The **Galaxy Dump** dialog box appears.
3. Browse to the name and location of the `.csv` file to which you want to dump the selected instances. Click **Save** to continue. The **Galaxy Dump** progress box appears.



Dump completed

```
Dumping all instances of $WinPlatform...
            Dumped: aWinPlatform_002
            Dumped: WinPlatform_001
Merging instances of $WinPlatform to test.csv...
GRDump completed.
```

Object 2 of 2 completed

4. After the Galaxy dump process is finished, click **Close**. A `.csv` file is created containing the selected objects and configuration data.

Galaxy Dump File (.csv) Structure

Dumped objects are organized in the resulting `.csv` file based on the template each is derived from. A header row per template indicates the instance's columns' reference. Comments can be added in the resulting `.csv` file by adding a line typing a semi-colon as the first character in the comment.

Take the following behaviors into consideration when creating and using a Galaxy dump file:

- 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 of 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).

  The Host attribute column, like any other data in the Galaxy dump file, can be deleted in an editing program. This action has no effect on Platform objects in subsequent Galaxy Load operations since they take the Galaxy name as their Host. All other objects, given no Host data, are staged in the Unassigned folder.

- 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 character "\" followed by the character "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 quote marks in the `.csv` file. Type all single quotes as two single quote marks and surround the entire string with opening and closing quote marks. Make sure the string contains an even number of quote marks. When the object is loaded in a Galaxy Load operation, the extra quote marks are stripped from the string. For example, if you want to enter `3"Pipe` as a Short Description, add a second quote mark (`3"“Pipe`) and then surrounding quote marks (`“3””Pipe”`).

- 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 in a subsequent Galaxy Load operation. Galaxy Load accepts two formats: `DAYS HH:MM:SS:SSSSSS` or `HH:MM:SS:SSSSSS`. The word `DAYS` in the first format is followed by a space. But if you use the latter time format, Excel automatically changes the entry to an incompatible format. So when you are typing time entries in Excel, use the following format: `DAYS HH:MM:SS:SSSSSS` (valid examples: `0000 01:02:12.123:1 04:03:06.12:120 22:66:88:123456`).

Creating Instances from Templates

Creating instances from templates allows you to use the configurations and specifications already programmed into a template.

If assignment defaults were previously configured (see Using Object Assignment Defaults), the following rules automatically apply:

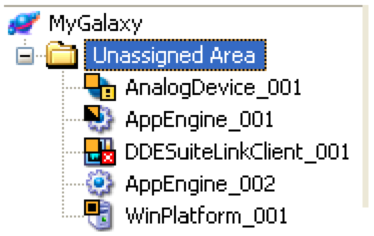
- A WinPlatform instance is created under the default Area and security group.
- An AppEngine instance is created under the default Area, security group and WinPlatform.
- An ApplicationObject is created under the default Area and security group.
- A DINetwork Object is created under the default Area and security group, and is assigned to the default AppEngine.
- A DIDevice Object is created under the default Area and security group, but is not assigned to an AppEngine by default.

To create an instance from a template

1. Select the template in the IDE's Main Window.
2. Click **New** on the **Galaxy** menu and click **Instance**. The new instance is created in the **Application Views** pane that is currently open. The instance is placed in name edit mode.
3. Rename the new instance as needed. See Allowed Names/Characters.

Instance Icon States

Examples of instances in the model are shown below.



Note that the typical object icons have changed. Two kinds of indicators accompany object icons: deployment status (for instances only) and configuration status (for templates and instances).

Deployment status indicators include:

🟧	Undeployed. See AnalogDevice_001 and DDESuiteLinkClient_001 in example above
(no indicator)	Deployed. See AppEngine_002 in example above
◪	Deployed with configuration changes. See AppEngine_001 in example above
▣	Deployed with software update required. See WinPlatform_001 in example above
◫	Applies only to redundancy scenario. AppEngine undeployed, its redundant pair not undeployed.
◫	Applies only to redundancy scenario. AppEngine deployed, its redundant pair not deployed.
◪	Applies only to redundancy scenario. AppEngine deployed, its redundant pair not deployed pending configuration updates.
▣	Applies only to redundancy scenario. AppEngine deployed, its redundant pair not deployed pending required software update.

For more information, see ArchestrA Redundancy.

Configuration status indicators include:

⚠	Configuration warning. See AnalogDevice_001 in example above
✖	Configuration error. See DDESuiteLinkClient_001 in example above
(no indicator)	Configuration good. See AppEngine_002 in example above

Validating Objects

Each object in a Galaxy has a set of possible configurations that authorizes its proper use in an application. That set of configuration possibilities is validated by the object either while you are configuring it or when you save that configuration to the Galaxy database.

Validation of an object's configuration includes checking allowable attribute value ranges, compiling its scripts, updating and binding its references, validating its extensions, updating its status, and validating other configuration parameters that are unique to the object.

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

Typically, each option on an object's 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 editor page, close the editor or save the object's configuration, a message appears about the input error, showing the allowable range.

Some configuration settings depend on associations with external components, such as script function libraries and relative references to other objects' attributes. The status of these external components can change, perhaps rendering some capability of the object inoperative.

For example, an object refers to a value of an attribute of another object, which is subsequently deleted. That scenario breaks the configuration of the remaining object. Objects are configured prior to the importing of associated script function libraries. In each case, the object has a status of either Bad or Warning. You can verify that an object's configuration is valid and reset its status to Good by manually validating it with the **Validate** command on the **Object** menu.

Manual Validation

To manually validate one or more objects, select the object(s) and click **Validate** on the shortcut menu (by right-clicking the object) or on the **Object** menu. You can select objects from the **Template Toolbox**, the **Application Views** or the **Find** dialog box.

Important! Manual validation can be done on both templates and instances, but only on those that are checked in.

Using the **Find** dialog together with the **Validate** command is an especially useful tactic. For example, you can find objects in Error state, select them all, right-click on one of them, and click **Validate** on the shortcut menu.

The **Validate** command opens the **Operations** pane in the IDE. See Operations Pane for more information.

Only one validation operation can be run at a time. But you can multi-select more than one object for each validation operation. The set of objects are validated serially.

Note You cannot cancel validation operations.

Continue using the IDE to perform other operations, if necessary, while validation is ongoing, including work on objects in the validation set. If an object is not available for validation when the command is initiated on it, validation is not performed. Also, if validation is in process on an object, other operations initiated by you on the object fail. Failure to perform validation on an object is indicated in the **Command Results** column of the **Operations** pane.

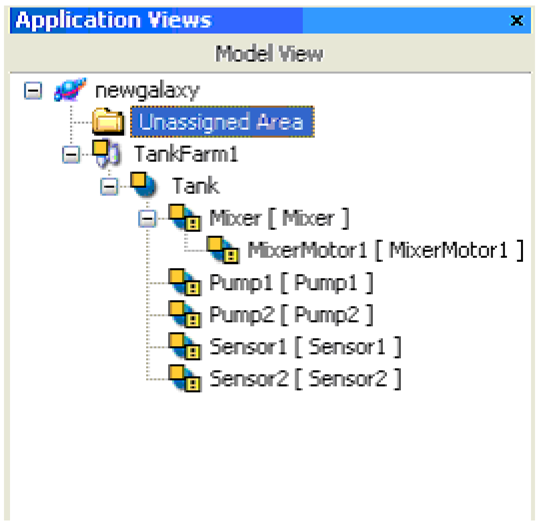
To validate all objects in the Galaxy, validate the Galaxy object.

Building Your Application

Building a Galaxy application is done on two levels, a real-world model and an object distribution model.

The real-world model, which is shown in the **Model View** of the **Application Views** pane, reflects the way your processes are organized in your plant. Typically, processes are divided into areas of activity, and those areas are comprised of devices like tanks, conveyor belts, pumps, sensors and other real-world things.

Build a representation of your real-world plant in the **Model View**, containing complex equipment in an area and smaller devices within the complex equipment. For example, a tank could be part of a process in a TankFarm area. An object representing that tank is contained in an object representing the area. Inside the real-world tank might be several smaller devices like mixers, mixer motors, pumps and sensors. Each of those smaller devices can be represented by objects that comprise the tank. The **Model View** can look like the following:



```
newgalaxy
    Unassigned Area
    TankFarm1
        Tank
            Mixer [ Mixer ]
                MixerMotor1 [ MixerMotor1 ]
            Pump1 [ Pump1 ]
            Pump2 [ Pump2 ]
            Sensor1 [ Sensor1 ]
            Sensor2 [ Sensor2 ]
```

In this example, TankFarm1 is an area that represents a recognizable process in your plant.

In the **Derivation View**, TankFarm1 is an instance of the area template, $TankFarm1. Tank and the other objects in the hierarchy are also instances of ApplicationObject templates found in the **Derivation View**.

The object distribution model, shown in the **Deployment View** of the **Application Views** pane, reflects the way you distribute objects on computers on the network. This distribution can approximate the distribution of your process devices, but it does not have to. Typically, the determining factor relates to load balancing the computers that application objects are deployed to.

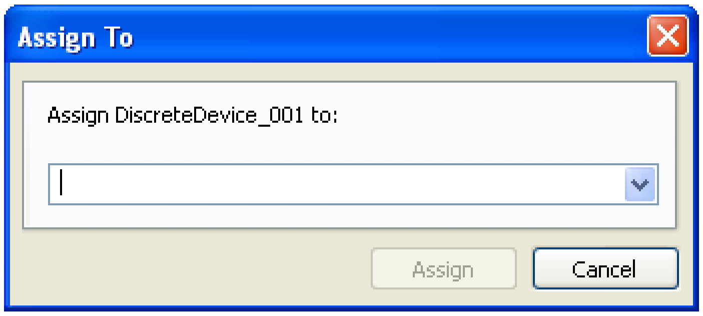
The first step in building your Galaxy application is determining which templates best represent the devices you want to emulate. Each object has Help documentation that can assist you in making that determination. After you select the proper object, each object must be configured to represent a unique field device. This is done through the object's configuration editor. See “Working with Object Editors” for more information.

Assigning Instances

Important! Certain assignment relationships are not possible. Allowed relationships depend on the type of object instances and in which **Application View** the assignment is being done. Also, an object cannot be assigned to another object if either object is already deployed.

To assign an instance to another instance

1. Select the object you want to assign.
2. On the **Object** menu, click **Assign To**. The **Assign To** dialog box appears.



3. Enter the name of the host object and click **Assign**.

Note A faster way to assign objects is to drag them. The universal “not” symbol indicates disallowed assignments.

Important! A maximum of 30,000 objects can be assigned at one time. Trying to exceed this limitation results in an error message and no objects being assigned.

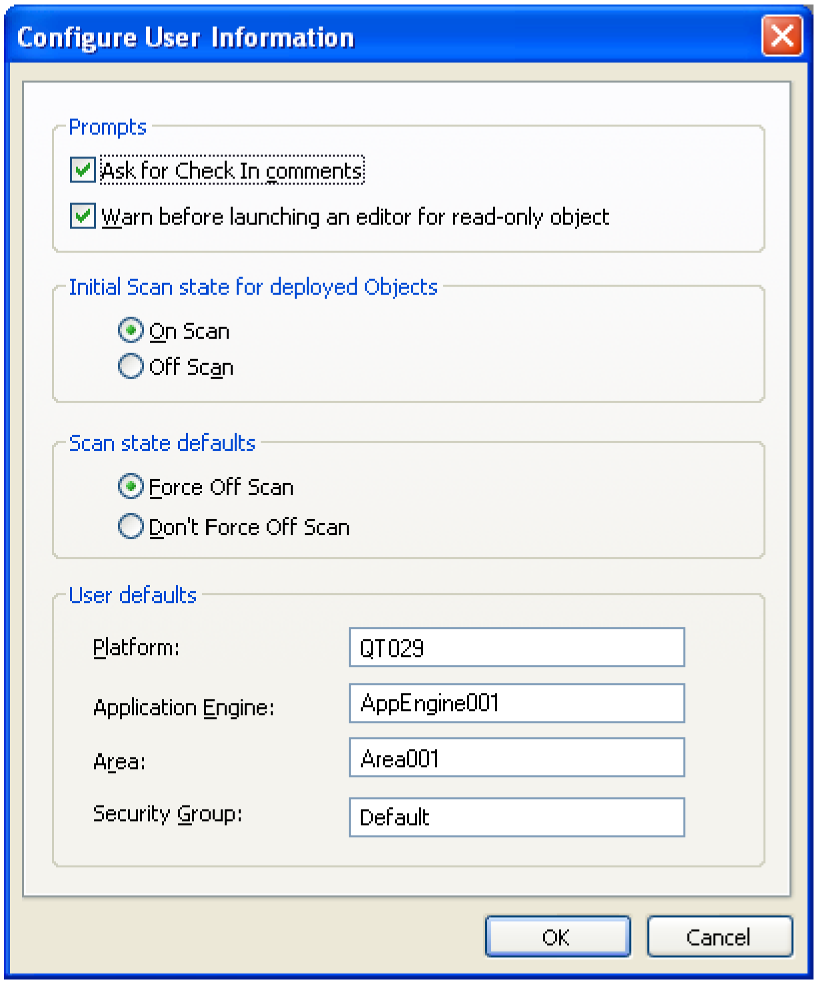
Object assignments can be done in the **Unassigned Host** folder. When the assignment is complete, all IDEs connected to the Galaxy reflect the change.

Unassigning objects is similar to assigning them. drag them onto another object or into the **Unassigned Host** folder.

Using Object Assignment Defaults

Use the **Configure User Information** dialog box to create automatic default scenarios before you begin creating instances from templates. Setting defaults simplifies instance creation, assignment and configuration.

To open the **Configure User Information** dialog box, click **User Information** on the **Edit** menu.



Configure User Information
Prompts
Initial Scan state for deployed Objects
Scan state defaults
User defaults
OK
Cancel

[✓] Ask for Check In comments
[✓] Warn before launching an editor for read-only object

[✓ On Scan] Off Scan

[✓ Force Off Scan] Don't Force Off Scan

Platform:	QT029
Application Engine:	AppEngine001
Area:	Area001
Security Group:	Default

The **Configure User Information** dialog contains the following elements:

- **Ask for Check-In Comments**: Use this check box to control whether the **Check In** dialog appears when you check in an object.
- **Warn Before Launching an Editor for Read-Only Object**: Use this check box to show a warning message when you try to open a read-only object's editor. An advantage to selecting (checking) this element is you can avoid opening a read-only object's editor without knowing that you are in read-only mode. Clearing (unchecking) this option creates the possibility that you can open a read-only object's editor without knowing that you cannot edit it.
- **Initial Scan State For Deployed Objects**: Use these option buttons to set the default scan state (on scan or off scan) of objects you deploy. You can change this setting on an individual basis in the **Deploy** dialog box.
- **Scan State Defaults**: Use these option buttons to set the default action when undeploying or redeploying instances. **Force Off Scan** causes the target object to go off scan; **Don't Force Off Scan** does not.

- **User Defaults**: Use these boxes to enter the object names of Framework objects you select to be defaults with respect to assignment relationships. Also select the default setting for associating a security group with new objects in the Galaxy.

  Note **User Defaults** functions can be done in the IDE Main Window by selecting the WinPlatform, AppEngine or Area object and then clicking **Set as Default** from the **Object** menu. Objects designated as defaults are typed in bold. Defaults are associated with the Galaxy you are connected to, and they are preserved for subsequent IDE sessions in that Galaxy.

All instances created afterwards are, by default, assigned to or configured using these configuration defaults, if the default is relevant. If no default is configured, the security group specified in the template is used.

Important! The **Security Group** user default overrides the default security group for a template as set in the **Configure Security** dialog box.

Default values for **Initial Scan States** and **Scan State Defaults** are automatically reflected in the **Currently Deployed Objects** and **Initial Scan State** groups of the **Deploy** dialog box. See "Deploying Objects" for an image of this dialog box.

Note The settings in this dialog box as well as other user interface environment settings (such as the size and position of the IDE Main Window and the docking locations of the **Template Toolbox** and **Application Views** panes) can be reset to default values by pressing **Shift** when the IDE Main Window first appears after connecting to a Galaxy and login.

Deploying Objects

Before an object can be deployed to a target computer, the following conditions must be met:

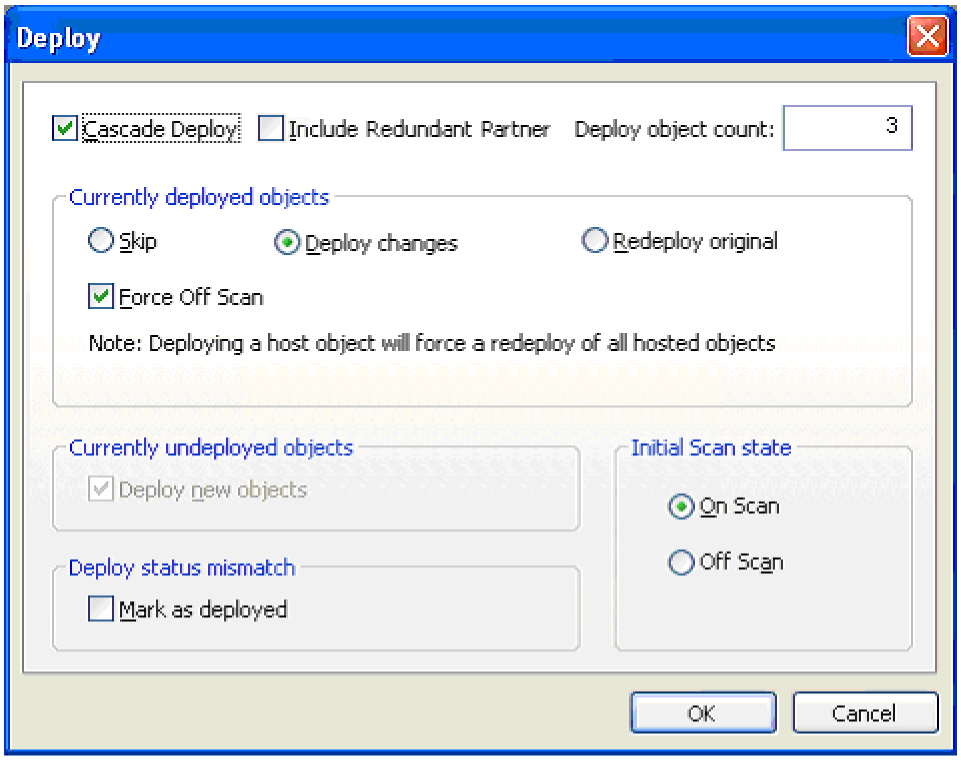
- You created, configured and checked in the object to the Galaxy.
- The object is assigned to a host.
- The object's host is already deployed. A cascade deploy operation, which deploys a hierarchy of objects, deploys all objects in the correct order. So an object's host is deployed before the object.
- Bootstrap software is installed on the target computer.
- The object being deployed is not in an Error state in the Galaxy database.

Important! 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. Refer to the DINetwork object's help file for specifics on configuration limits.

To deploy an object

1. Select the object from an **Application View** pane.

2. On the **Object** menu, click **Deploy**. The **Deploy** dialog box appears.
3. Set the parameters as needed and click **OK**. A progress box appears while deploy occurs.



[✓] Cascade Deploy [ ] Include Redundant Partner Deploy object count: 3

Currently deployed objects
Skip [✓ Deploy changes] Redeploy original
[✓] Force Off Scan
Note: Deploying a host object will force a redeploy of all hosted objects

Currently undeployed objects
[✓] Deploy new objects

Deploy status mismatch
[ ] Mark as deployed

Initial Scan state
[✓ On Scan] Off Scan

The **Deploy** dialog box consists of the following options:

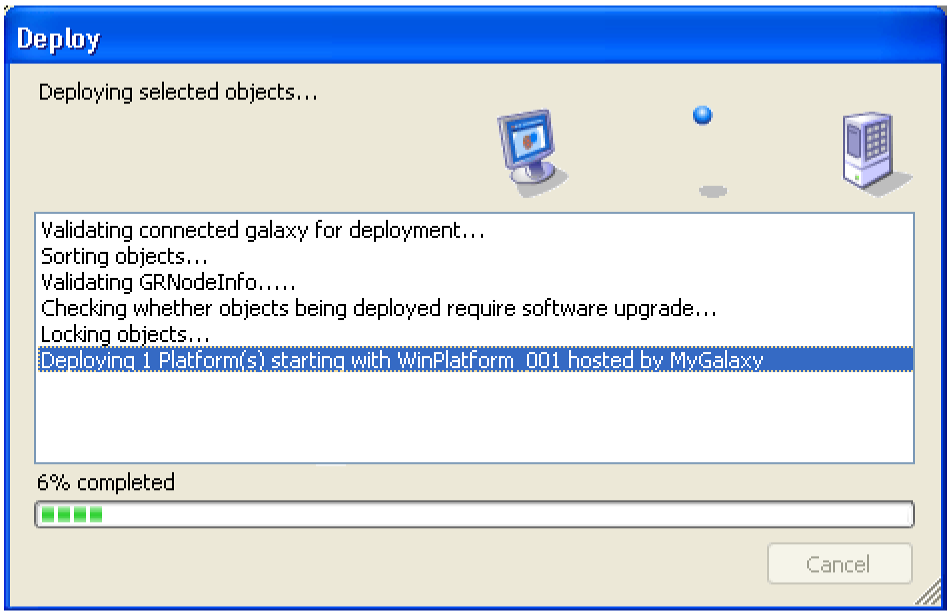
- **Cascade Deploy**: Select this check box to deploy the object selected for deployment as well as any objects it hosts. This option is selected by default if the object is a host. If you are deploying an individual host object, clear the check box.
- **Include Redundant Partner**: Select this check 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. See ArchestrA Redundancy for more information.
- **Objects to Deploy**: Shows the number of intended objects to deploy. You can select multiple objects in **Application Views** and the number of objects appears in this box. Also, you can select a single object in **Application Views** and, if **Cascade Deploy** is selected and objects are assigned to the object you are deploying, the total number of objects appears in this box.
- **Skip**: Refers to deployment of objects that are currently deployed. If one of the objects you are deploying is currently deployed, selecting **Skip** makes no changes to the already-deployed object. This option is unavailable if the object was not deployed previously.

- **Deploy Changes**: Refers to deployment of objects that are currently deployed. If one of the objects you are deploying is currently deployed, selecting **Deploy Changes** updates the object in question with new configuration data. This option is unavailable if the object was not deployed previously.
- **Redeploy Original**: Refers to deployment of objects that are currently deployed. If one of the objects you are deploying is currently deployed, selecting **Redeploy Original** deploys the same version as previously deployed. For example, use this option to redeploy an object that is corrupted on the target computer. This option is unavailable if the object was not deployed previously.
- **Force Off Scan**: Refers to deployment of objects that are currently deployed. If one of the objects you are deploying is currently deployed, selecting **Force Off Scan** causes the target object to be set to off scan before deployment occurs. This option is unavailable if the object was not deployed previously.
- **Deploy New Objects**: Refers to deployment of objects that are currently undeployed. Selecting this check box initiates a normal deployment mode. This option is unavailable if the object was deployed previously.
- **Mark as Deployed**: Refers to a mismatch in deployment status of the object being deployed. The mismatch exists when the object is previously deployed to a target node but the Galaxy indicates the object is undeployed. Selecting this option marks the object as deployed in the Galaxy. Clearing this option redeploys the object to the target node.
- **On Scan**: Sets the initial scan state to on scan for the object you are deploying. The default setting is controlled by the **User Default** settings in the **Configure User Information** dialog box (see “Using Object Assignment Defaults”). 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.

  Note Areas should always be deployed to their host AppEngines on scan. Since Areas are the primary providers to alarm clients, deploying it off scan results in alarms and events not being reported until it is placed on scan.

- **Off Scan**: Set the initial scan state to off scan for the object you are deploying. The default setting is controlled by the **User Default** settings in the **Configure User Information** dialog box (see “Using Object Assignment Defaults“). If you deploy objects off scan, you must use the ArchestrA System Management Console’s Platform Manager utility to put those objects on scan and to function properly in the runtime environment.

4. Click **OK** to deploy the object(s), and the **Deploy** progress box appears.



Deploying selected objects...

```
Validating connected galaxy for deployment...
Sorting objects...
Validating GRNodeInfo.....
Checking whether objects being deployed require software upgrade...
Locking objects...
Deploying 1 Platform(s) starting with WinPlatform_001 hosted by MyGalaxy
```

6% completed

If the object being deployed has configuration problems that were not known during configuration, the **Deploy** progress box shows messages. For example, the minimum required RAM on the target computer is larger than the actual RAM. This type of configuration parameter is not validated during configuration of the object, but is discovered during deployment.

The progress dialog box shows the affected instances and 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.

Redeploying Objects

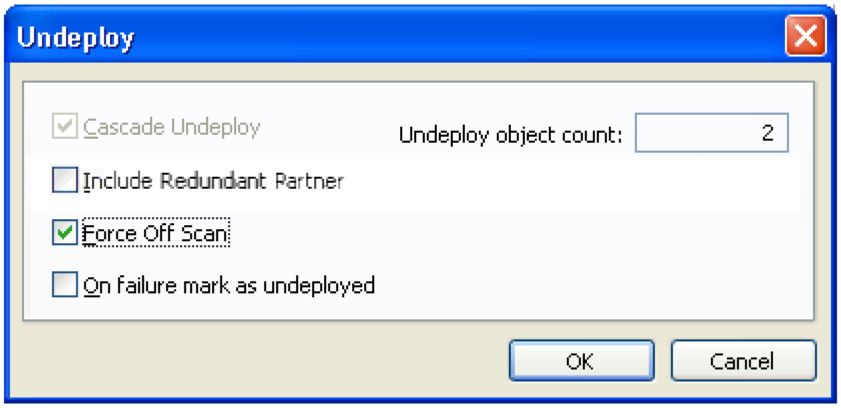
Redeploying an object is similar to deployment. To do this, click **Deploy** on the **Object** menu and following the procedure in “Deploying Objects."

If the state of the previously-deployed object is Pending Update (that is, the object has been reconfigured since deployment) and you select to deploy those changes, then the new object is marked as the last deployed version in the Galaxy.

Undeploying Objects

Note Undeploy fails if the target object has objects assigned to it unless you select **Cascade Undeploy** on the **Undeploy** dialog box.

This dialog box appears when you select the **Undeploy** command on the **Object** menu. It lets you manage the undeploy process.



[✓] Cascade Undeploy
Undeploy object count: 2
[ ] Include Redundant Partner
[✓] Force Off Scan
[ ] On failure mark as undeployed

The **Undeploy** dialog box contains the following options:

- **Cascade Undeploy**: Select this check box to undeploy the selected object as well as any objects it hosts. The availability of this option depends on which object(s) you select for undeployment.
- **Undeploy Object Count**: Shows the number of objects being undeployed. You can select multiple objects in **Application Views** and the number of objects appears in this box. Also, you can select a single object in **Application Views** and, if you selected **Cascade Undeploy** and other objects are assigned to the object you have selected, the total number of objects appears in this box.
- **Include Redundant Partner**: Select this check box 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** causes the target object to be set 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**: Use this option to mark the object as undeployed in the Galaxy when the object targeted for undeployment is not found.

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

Undeployment Post-Conditions

The following conditions 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 with the area (directly, 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 instance. However, a host might not be undeployable if one of its assigned objects cannot be undeployed.
- Assume an ApplicationObject hosted by the Active AppEngine in a redundant pair and a number of subscriptions configured in that ApplicationObject that refer 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. The time that governs how fast those items are removed is the value of the **Maximum time to maintain good quality after failure** option (Redundancy.StandbyActivateTimeout attribute) on the **Redundancy** page of the AppEngine’s editor. This behavior does not apply to the undeployment of ApplicationObjects hosted by non-redundant AppEngines.

C H A P T E R 3

Working with Object Editors

You configure an object using its editor. When you open an object’s editor, it uses all space in the IDE’s Main Window not occupied by the **Template Toolbox** and **Application Views**.

This chapter describes how to open and close object editors, and the common editor options you can use to configure the objects in your Galaxy application.

Contents

Opening/Closing an Object’s Editor

An object’s editor consists of custom pages and object extension pages that are common to all editors. The content of the custom editor pages is defined by the object. See “Enhancing an Object’s Functionality” for more information about the **Scripts**, **UDAs** and **Extensions** pages. Select the tab of each page to show its contents.

When you have an object’s editor open, the object’s name is added to the list on the **Windows** menu. When you close the editor, the name is removed from the list.

To open/close an object's 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 also checked out. You can also click **Open Read-Only** for viewing only.
3. When you are done 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.

- Select the **Keep Checked Out** check box before closing to keep the object checked out. You are prompted if configuration data was changed in the editor. Click **Yes** to save the data, **No** to close the editor and revert to previous configuration values.

Note When an object's editor is closed, configuration data is validated. If errors or warnings are identified during validation, a message appears. You can cancel the save process or save the object configuration as it is. If you cancel, the editor remains open. 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 indicates the object is undeployable.

Configuring an Object in its Editor

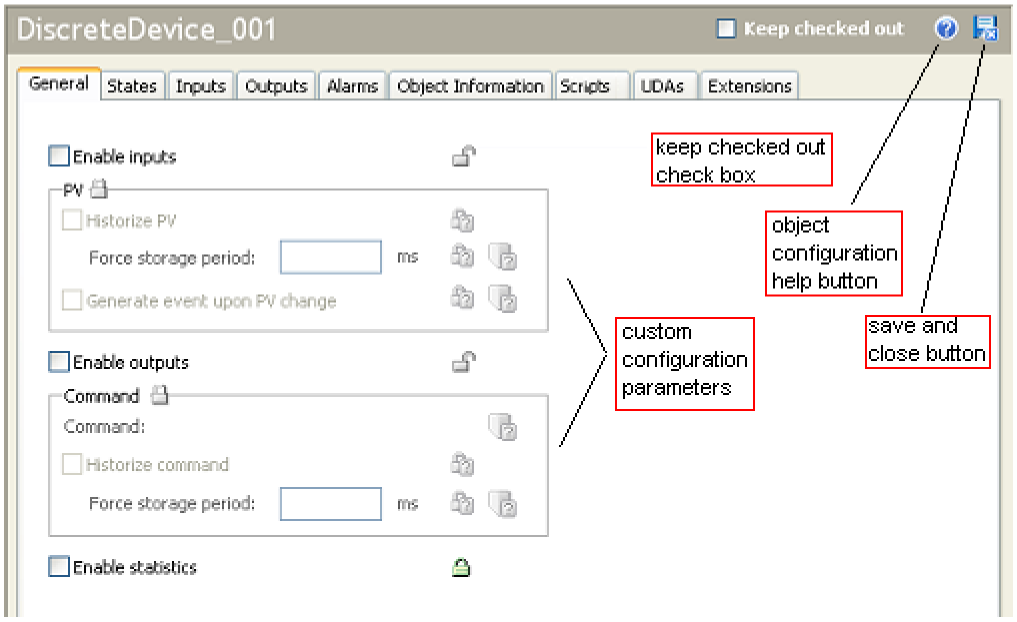
Attribute-level validation of entered values that are out of range or otherwise invalid is done when the editor option loses focus. If the value entered is not valid then the reason why the data is rejected is shown in a message.

Note Do not add leading or trailing spaces when typing any reference in an object's editor. References with such spaces fail to bind and their quality is Bad.

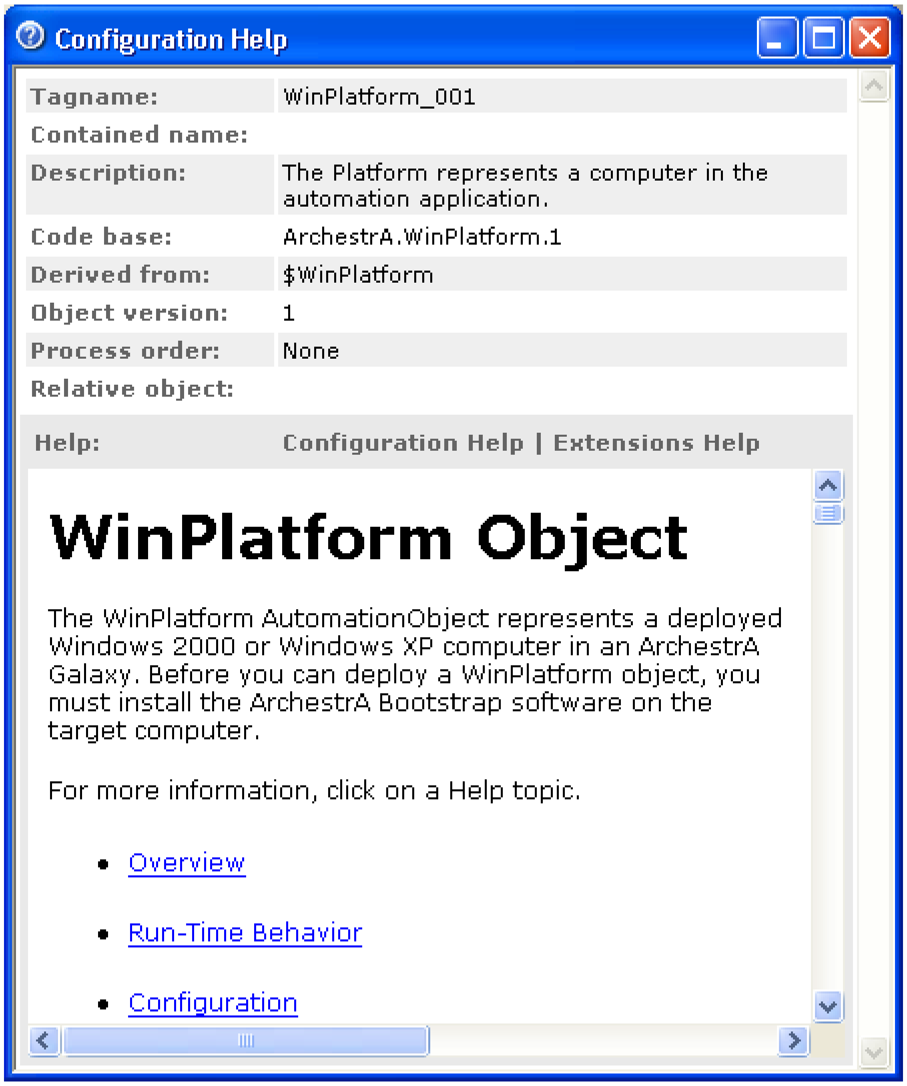
General Editor Layout

Object editors have many common elements. The editor shown in the following image is provided as an example.

When you close an object's editor, the object is also checked in to the Galaxy database. Selecting the **Keep Checked Out** check box before closing maintains the object in checked out status.



Each object provides its own documentation about usage, configuration, runtime behavior and attributes. For help in configuring the object's editor, click the question mark to open the object's configuration Help. See the example below of a typical object Help file.



Tagname:	WinPlatform_001
Contained name:	
Description:	The Platform represents a computer in the automation application.
Code base:	ArchestrA.WinPlatform.1
Derived from:	$WinPlatform
Object version:	1
Process order:	None
Relative object:	
Help:	Configuration Help | Extensions Help

WinPlatform Object

The WinPlatform AutomationObject represents a deployed Windows 2000 or Windows XP computer in an ArchestrA Galaxy. Before you can deploy a WinPlatform object, you must install the ArchestrA Bootstrap software on the target computer.

For more information, click on a Help topic.

- Overview
- Run-Time Behavior
- Configuration

The header of the Help file contains the following information:

- **Tagname**: The object’s name.
- **Contained Name**: The object’s contained name (see “Working with Containment“).
- **Description**: A short summary of the object’s purpose.
- **Code Base**: The code version of the object.
- **Derived From**: The immediate parent for the object (for example, a derived template).
- **Object Version**: The configuration version of the object.
- **Process Order**: The runtime execution order (none, before, after) relative to the **Relative Object** element.
- **Relative Object**: The object that is executed before or after in the **Process Order**.

If help for an object does not exist, a Help window appears that shows the path where the file should exist. This can happen if the object's help file was deleted from the Galaxy database. You can create a Help.HTM file and stored in the proper location, or you can import the object again.

Tooltip help is also available on the object editor. Point to any editor option and its tooltip (attribute name) appears.

Click the **Save and Close** icon to save configuration changes you made and close the editor. Closing the editor also checks in the object to the Galaxy database unless **Keep Checked Out** is checked. After the object is checked in, its configuration is accessible to other users.

Lock Icons

Operators interact with objects through the individual attributes of those objects. Any editor option (attribute) can have an associated lock control, which restricts it from further configuration. Some options are locked by the object developer because their values must remain static in all instances. Other options are left unlocked so that you can configure and lock them from additional manipulation in child objects. Each option can be associated with only one lock at a time.

If an option has a lock associated with its attribute, the lock control appears. If the option is enabled, click the lock control to switch it between locked and unlocked.

Possible lock controls include:

Lock Icon	Name	Description
🔒	Locked (in me)	Option is locked (in me) and enabled. For templates only.
🔒	Locked (in parent)	Option is locked in the parent object and disabled in the child object.
🔓	Unlocked	Option is unlocked and enabled. For unlocked attributes in templates only.
🔒	Indeterminate	This icon only refers to a specified group of options. An indeterminate state indicates different lock states for individual options in the group.
🔒?	Undefined	Option requires that another attribute be enabled before its locking function can be determined.

Security Icons

Operators interact with objects through the individual attributes of those objects. Each option (attribute) on an object's editor can have an associated security control, which determines its runtime security. Each option can be associated with only one security control at a time.

Important! An attribute's security classification is enabled when you select one of the three security authentication modes. See Working with Security for more information about authentication modes and security in general.

If an option has security associated with its attribute, the security control appears. If the option's security is configurable, click the security control to select one of seven possible states:

Security Icon	**Description**
Free Access	Allows users to change this value without restriction even if the user has no defined permissions on the object. Any user can write to this attributes to perform safety or time critical tasks that can be hampered by an untimely logon request (for example, halting a failing process).
Operate	Allows users with Operate permissions to do certain normal day-to-day tasks. These include writing to attributes like Setpoint, Output and Control Mode for a PID object, or Command for a Discrete Device object. This level of security requires the user have permission for the security group for the object.
Secured Write	Requires a user who has Operate permissions to the object to login again prior to writing to the attribute. Operators write to these attributes for normal interaction with a highly secured object.
Verified Write	Requires a user who has Operate permissions to login again and a second, different user to also login prior to writing to the attribute. Operators write to these attributes for normal interaction with a very highly secured object.
Tune	Allows end users with Tune Operational permissions to tune the attribute in the runtime 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 runtime environment. This attribute is never written to at runtime, regardless of the user's permissions.

An attribute’s security can be colored gray, which means it is either secured 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 of the 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.

Object Editor Behavior

Use the **TAB** key to change focus from one option to the next in an object editor, generally from top left to bottom right and including any option you can interact with. The **TAB** function does not include locked options. After you begin editing text in a memo-style text box, the **TAB** key inserts a tab character for alignment purposes.

The **HOME** and **END** keys reposition the cursor to the start and end, respectively, of the option field. Arrow keys reposition the cursor in text boxes and move through previous and following values of a list.

When an option loses focus (you press **ENTER** or **TAB**, or you click anywhere outside of the option) the data is processed. Validation, if any, occurs at this time.

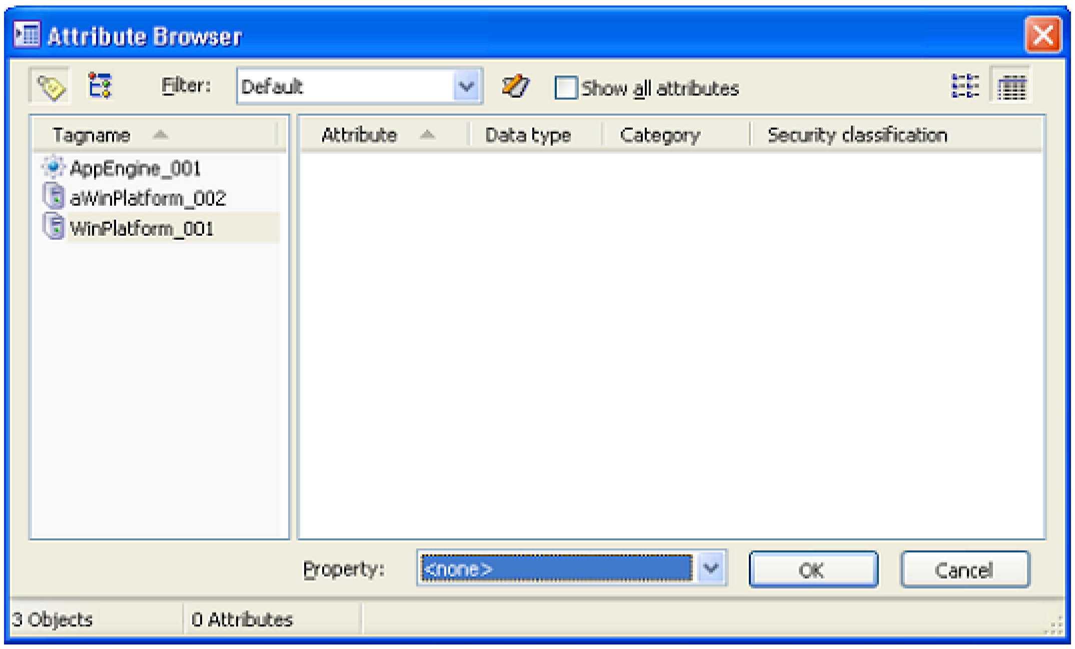
Press the Escape (ESC) key to cancel any editing of an option. The option reverts to its previous value.

When an option’s check box is in focus, you can switch the check box from selected to cleared by pressing the spacebar.

Referencing Objects Using the Attribute Browser

The **Attribute Browser** is an ArchestrA tool you can use to quickly find an object attribute (or attribute property) and then add that reference to another object's configuration.

Specific options on an object's editor (of the type Reference) require you to enter references to other objects. To easily determine an attribute you want to reference, select the option and click the **Attribute Browser** button. The following dialog box appears.



Filter: Default

Tagname
AppEngine_001
aWinPlatform_002
WinPlatform_001

Attribute	Data type	Category	Security classification

Property: <none>

3 Objects 0 Attributes

See "Enhancing an Object's Functionality" for more information about using the **Attribute Browser** and references to objects.

Attribute Browser User Interface

The Attribute Browser consists of four main functional areas: the top row of options, the left pane, the right pane and the property selector.

Top Row of Options

To switch the content of the left pane (**Tagname** in the previous image) between a Tagname and Hierarchical Name list of objects, select either the Use Tagname or Use Hierarchical Name buttons .

To create a filter to limit the list of objects, click the Edit Filter icon . Filters can be based on object name or common attributes. After you create a filter, it appears for future use in the **Filter** list. The Default filter provides an unfiltered list of objects and attributes; it cannot be edited. See "Creating a Filter" for more information.

By default, the browser shows only those attributes defined as frequently accessed. If you are viewing the attributes of an object for the first, the right pane could be blank. Check the **Show All Attributes** box to show all of the object's attributes.

The Left Pane

The objects shown in the left pane include all of the logged in user's checked-out objects plus the checked-in versions of all other objects. Use the Use Tagname or Use Hierarchical Name buttons to change the content of this pane.

Important! The Attribute Browser shows only the Primary AppEngine and its attributes of a redundant pair. The Backup AppEngine is not shown. See ArchestrA Redundancy for more information.

The Right Pane

The right pane of the **Attribute Browser** shows the attributes of the object selected in the left pane. Each attribute is listed under the following column headings:

• Attribute	The name of the attribute accompanied by an icon that represents the Data Type group to which it belongs.
• Data Type	The data type of the attribute, such as String, Integer, Boolean, Reference, Time, InternationalizedString and CustomEnum.
• Category	Indicates whether the attribute is readable/writeable during configuration and runtime.
• Security Classification	The attribute's runtime security, such as Tune, Operate, Free Access, Secured Write, Verified Write, View Only. Security controls access to each object attribute and the data it represents. If an attribute has no security, this column is blank. For more on security classifications, see Security Icons.

Property Selector

After you select an attribute, you can select a specific property to reference in the object. Depending on the attribute selected, possible properties include:

<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 entire dimension of the attribute if it is an array.

Locked	determines whether the attribute is currently locked. Valid values are Unlocked LockedInMe LockedInParent.
Quality	the quality of the attribute as defined in the OPC Draft 3.0 quality definition. ArchestrA 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 ArchestrA application in the runtime environment. Relevant only for attributes that are user writeable in the runtime environment.
Type	the data type of the attribute: Integer Boolean Float Double String Internationalized String Time ElapsedTime ReferenceType CategorizedStatusType DataTypeEnum SecurityClassificationEnum DataQualityType CustomEnum CustomStruct.
Value	the primary value of the attribute. In some cases, 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: .00 (least significant bit) .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 .31 (most significant bit)

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 runtime environment.

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

Using the Attribute Browser

If the option in the object's 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.

Also, if you selected text in the script editor, that text is used by the **Attribute Browser** as the initial reference string and the browser finds the nearest attribute reference to the selected text.

When you are done selecting the attribute/property, click **OK** to place the reference into the object's editor. 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.

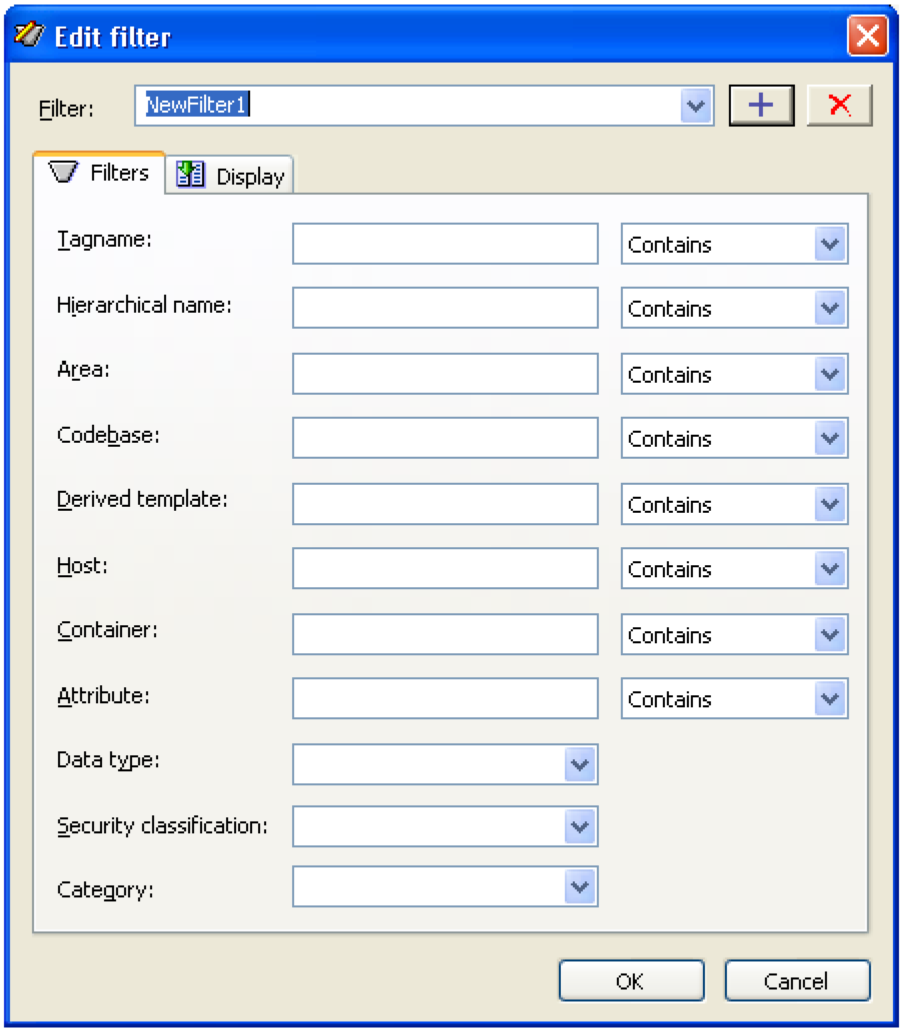
Important! The Attribute Browser shows the object list based on the last used state (Tagname or Hierarchical name). Selecting a partially entered reference in an object's editor that does not match the last used state of the browser does not automatically change the browser's state. If the last used state of the browser was Tagname and the selected editor reference is a Hierarchical name, the browser still opens in Tagname mode.

Creating a Filter

Depending on the number of objects deployed in your Galaxy application, the amount of data shown in the Attribute Browser can be large. To make this information more manageable, reduce the data shown by creating a filter.

To begin creating a filter, click the **Edit Filter** icon . Filters can be based on object names, assignment, containment or derivation; attribute names, type, category and security classifications; and the object and attribute fields shown in the Attribute Browser.

The **Edit Filter** dialog box initially shows the Default filter, listing all objects and attributes. To create a new filter, click the plus sign . A new filter name is provided in the **Filter** box in rename mode. Rename as desired.



Filter:	NewFilter1	
Tagname:		Contains
Hierarchical name:		Contains
Area:		Contains
Codebase:		Contains
Derived template:		Contains
Host:		Contains
Container:		Contains
Attribute:		Contains
Data type:		
Security classification:		
Category:		

The **Edit Filter** dialog box shows two pages in which you can set different kinds of filters. On the **Filters** page, use the options to reduce the number of objects and attributes listed. On the **Display** page, use the options to reduce the number of fields associated with each object and attribute.

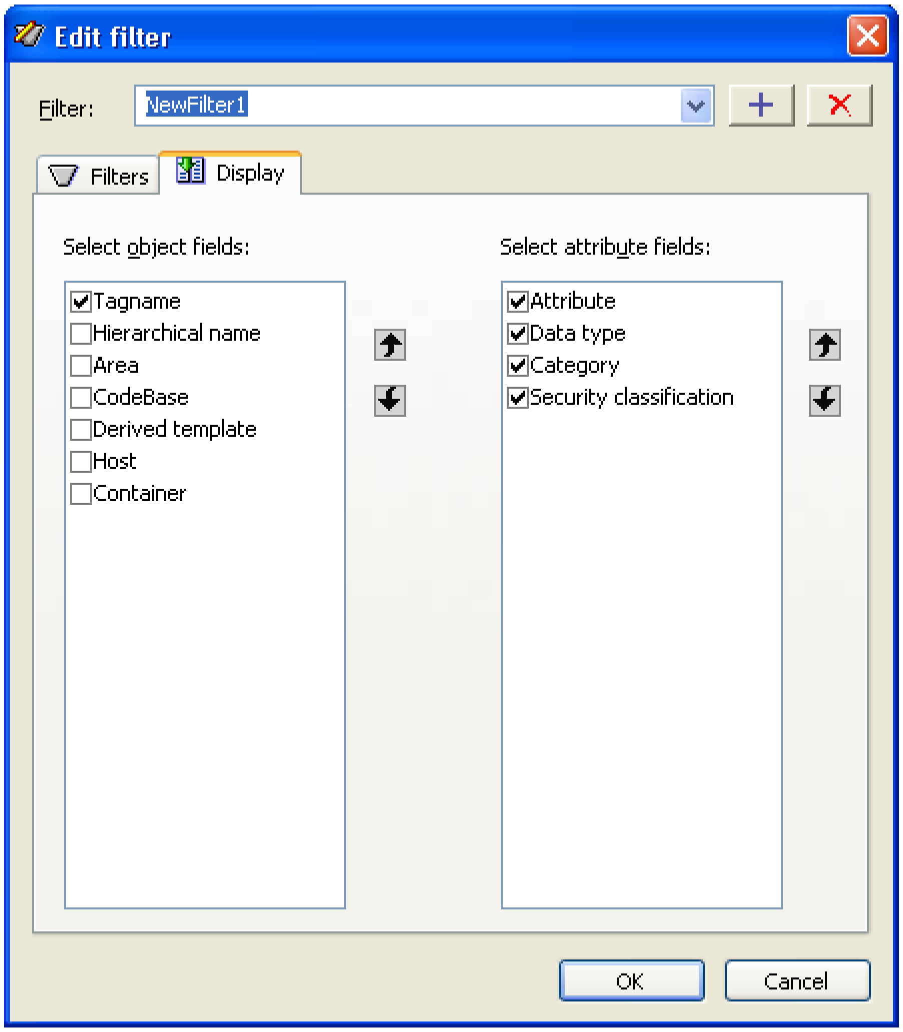
The **Filters** page shows eight options with an associated filter (default setting is **Contains**). Possible filters include:

- **Contains**
- **Exact match**
- **Starts with**
- **Ends with**

This page also shows three other filters, including:

Data Type	Security Classification	Category
Boolean	Configure	Calculated
CustomEnum	Free access	CalculatedRetentive
CustomStruct	Operate	Constant
DataType	Secured write	PackageOnly
Double	Tune	PackageOnly_Lockable
ElapsedTime	Verified write	SystemInternal
Float	View only	SystemInternal_Browsable
Integer		SystemSetsOnly
InternationalizedString		SystemWriteable
Quality		Writeable_C_Lockable
Reference		Writeable_S
SecurityClassification		Writeable_U
Status		Writeable_UC
String		Writeable_UC_Lockable
Time		Writeable_US
		Writeable_USC
		Writeable_USC_Lockable

Select the filter(s) best suit the data you want to show in the Attribute Browser and click **OK**.



Filter: NewFilter1

Select object fields:

- [x] Tagname
- [ ] Hierarchical name
- [ ] Area
- [ ] CodeBase
- [ ] Derived template
- [ ] Host
- [ ] Container

Select attribute fields:

- [x] Attribute
- [x] Data type
- [x] Category
- [x] Security classification

The **Display** page shows two sets of filtering options, one set for objects in the left pane of the Attribute Browser (begins with the **Tagname** column) and one for attributes in the right pane (begins with the **Attribute** column). Select (check) those options that you want to see in the Attribute Browser. Deselect (uncheck) those options you want to hide. Select an option in either list and use the up and down arrows to rearrange the order shown in the Attribute Browser.

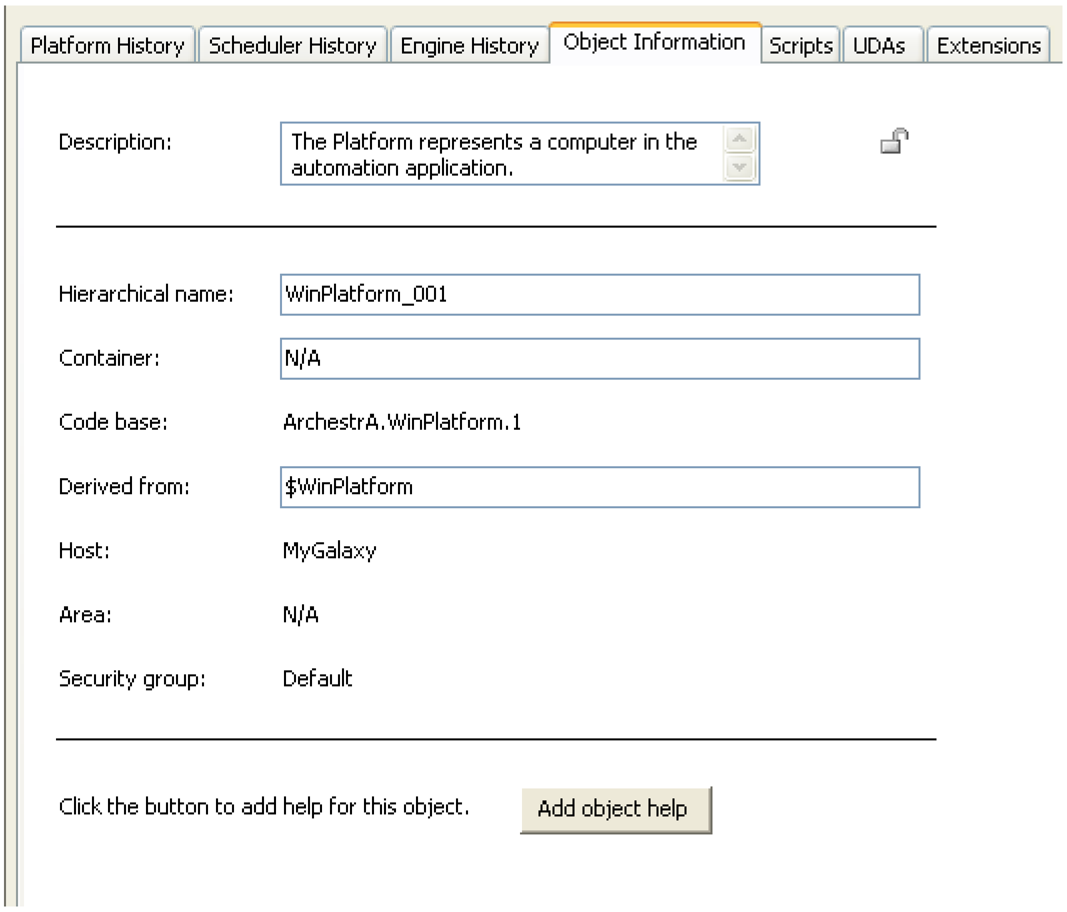
Note **Tagname** and **Attribute** are always selected for viewing and always first in order. You cannot change these settings.

Select the filter(s) that best suit the data you want to show in the Attribute Browser and click **OK**. After you create a filter, its name appears for future use in the **Filter** list on the Attribute Browser.

If you don't want any filter applied to the list of objects and attributes, select the **Default** filter.

Object Information Page

This page is common to all object configuration editors is the Object Information page.



Description: The Platform represents a computer in the automation application.

Hierarchical name: WinPlatform_001

Container: N/A

Code base: ArchestrA.WinPlatform.1

Derived from: $WinPlatform

Host: MyGalaxy

Area: N/A

Security group: Default

Click the button to add help for this object.

The **Object Information** page consists of the following elements:

- **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 another object that contains the 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).
- **Area**: An object that represents a logical grouping to which this object belongs.
- **Security Group**: The security group the object is associated with. See “Working with Security“ for more information about security groups.

- **Add Object Help**: Opens a copy of the Help documentation for the template this object is derived from. You can edit this topic. The purpose is to create Help about the object you are currently configuring for downstream users. This Help that appears when you select an object in a view and then click **Object Help** on the **Help** or shortcut menu.

Caution! Microsoft Word is not supported as an editor for creating downstream object help. If clicking **Add Object Help** opens Word on your computer, you must change the program associated with editing HTM files on the **File Types** page of Windows Explorer's **Folder Options** dialog box.

An additional group appears on the **Object Information** page of some objects: the **Execution Order** group. In this group, you can set a specific execution order for the object you are configuring, based on another object. You can set the object you are configuring to execute just before or just after another object executes.

The **Execution Order** group consists of a **Process Order** setting (**Before** or **After**) and a **Relative Object** reference, the latter being the name of the object whose execution timing is the benchmark for the execution of the object you are configuring.

The **Relative Object** reference requires an @ sign preceding the object name. If you type the object name, you must include the @; if you use the **Attribute Browser** to locate the object, the @ sign is added for you.

Important! To add images to the Help file, you must place those images in the proper folder on the Galaxy Repository computer and use a relative path to those images in the HTML file. The path to each object’s Help folder is unique and dependent on the path selected when you installed the Galaxy Repository. The path to an object’s Help is `\\<Installation Path>\Framework\FileRepository\<YourGalaxyName>\Objects\<TheObjectName>\Help\1033`. The default `<Installation Path>` is `\\<Program Files\ArchestrA\`. You can place images in the `\1033` folder or create an images folder under it.

C H A P T E R 4

Working with History

Wonderware's IndustrialSQL is the historian for Industrial Application Server. All ArchestrA objects can be configured to send attribute data for storage in an InSQL database.

This chapter describes how to configure objects to store historical data in InSQL, the Industrial Application Server historian.

For more information on historizing data, refer to your *Factory Suite A^2 Deployment Guide.*

Contents

InSQL the Historian

Installing InSQL has the following conditions:

- Must be installed on a computer outside the Galaxy but on the local network.
- Cannot have any ArchestrA components installed on the same computer.
- A single InSQL installation can receive historical data from a single Galaxy.

InSQL supports quality definitions that Industrial Application Server follows (OPC Draft 3.0). It creates additional VTQ records that modify quality when certain scenarios arise like network disconnects suggesting down times. Also, InSQL stores two additional data quality fields that are not OPC compliant (quality and quality detail) for each record received from Industrial Application Server components.

Important! Industrial Application Server communicates with the InSQL node through an interface called MDAS (Manual Data Acquisition Service). MDAS uses DCOM (Distributed COM) which requires that TCP/UDP port 135 is available on the InSQL node. If that port is not available, Industrial Application Server fails to configure and store history to InSQL. One possible reason that port 135 is not available is a router between the Industrial Application Server node and the InSQL node that is blocking that port. Another reason might be that DCOM is disabled on either node. For correct functionality, ensure that DCOM is enabled and not blocked when configuring Industrial Application Server.

Configuring WinPlatforms, AppEngines for History

To configure WinPlatforms and AppEngines for history

1. Check out the object.
2. Open the object's editor.
3. On the **Engine** page of the object's editor, select **Enable Storage to Historian**.
4. In the **Historian** box, enter the InSQL node name.
5. Enter the **History Store Forward Directory** location. If needed, the folder is created when the object is deployed. If you leave this option blank, a default location is used. This option is configurable only on WinPlatforms.
6. Enter the **Store Forward Deletion Threshold**.
7. Save and close the object editor.
8. Check in the object to the Galaxy.
9. Deploy the object to its target computer in an on scan state.

See "Working with Objects" and "Working with Object Editors" for more information about each step in the procedure above. Also refer to the Help documentation for WinPlatform and AppEngine for specifics about the custom editor pages of these object.

Important! If an AppEngine is deployed before InSQL is started, history is not stored until a successful registration with InSQL is achieved. When an AutomationObject is deployed on scan, some attributes' data points are not stored initially until they are registered and committed to InSQL.

Configuring Objects to Store History

Objects have attributes. Some attributes can be historized when values change. Data quality is also recorded along with attribute values. The historical records of this data can then be retrieved, trended and used for other purposes later.

For attribute data to be stored in the historian, a host AppEngine must be configured to send all history data to an InSQL node and deployed in the Galaxy.

Important! A change in quality always causes a new record to be stored, regardless of whether the value changed.

During runtime execution, if the object updates an attribute configured for historization, data is historized as follows:

- If the attribute type is numerical (double, float or integer) and the attribute's new value represents a change from the previously-historized data value by more than the value deadband; or the attribute's quality changed state (for example, from Good to Bad). If no previously-historized value exists, then the first value (for example, after startup) is always historized.
- If the attribute type is qualitative (enumeration, string or boolean) and the attribute's value or quality changed. If no previously-historized value exists, then the first value is always historized.
- If the current time equals or exceeds the time the last Forced Store occurred for the attribute (as determined by the Force Storage Period). If no last Force Store occurred since startup, a new store occurs immediately.

  Note The Value Deadband mechanism, if enabled, resets itself based on this new value stored.

- The new attribute value, timestamp and quality are sent to InSQL for storage.
- InSQL logs the data to disk.

Important! If the InSQL node shuts down, data storage continues locally. When the InSQL node recovers, the data is forwarded at a low priority. If an AppEngine loses connectivity with the InSQL node, InSQL reports bad data quality to clients. When you undeploy an object with attributes configured for historization, InSQL stores the final data points with Bad quality.

To configure an object to store history

1. Check out the object.
2. Open the object's editor.
3. For each attribute you want to historize, select the **History** check box and provide a **Force Storage Period** value.

   Force Storage Period is the time interval, in seconds, in which the attribute value must be stored, regardless of the value deadband setting. In addition to the **Value Deadband** setting, the value is stored continuously at this interval. A value of zero (0) disables this feature.

   Important! If successive and identical value/quality pairs are received by InSQL, they are not actually stored to disk.

4. For non-array, numerical attributes that you want to historize (for example, integer, floats and doubles), configure the **Value Deadband**, **TrendHi** and **TrendLo** options.

   Value Deadband is 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 is the default. Zero means that some change is required prior to storing the value.

   Trend Hi specifies the initial maximum trend value for clients.

   Trend Lo specifies the initial minimum trend value for clients.

5. Save and close the object editor.
6. Check in the object to the Galaxy.
7. Deploy the object in an on scan state to a host AppEngine that is configured for History (see “Configuring WinPlatforms, AppEngines for History."

See “Working with Objects” and “Working with Object Editors” for more information about each step in the procedure above. Also refer to the Help documentation for each object for specifics about the custom editor pages of that object.

Historizable Data Types for Attributes

Attributes of the following data types support historization:

- Float (numerical)
- Double (numerical) - maps to an InSQL float. 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, so truncation can occur. If so, quality is set to Uncertain.
- CustomEnum (non-numerical) - maps to an InSQL integer
- ElapsedTime (numerical) - maps to an InSQL float and converted to seconds

Note Entire arrays or portions of arrays are not supported.

Enum type attributes are historized as integer ordinal values. NaN values for float and double data types are converted to null values.

All numerical attributes are sent to InSQL in the engineering units exposed by the attribute. InSQL does not scale the value.

History Extension During Configuration

The Extensions page that is common to all object editors allows you to historize attribute values that were not originally set up for history. See Enhancing an Object’s Functionality for more information about adding this kind of history to your application.

InSQL and Object Redeployment, Undeployment

When an AutomationObject configured for history is redeployed, changes to the attribute configuration of the object makes the historian to reconfigure storage. For example, if the engineering units string for the tag change from “Deg F” to “Deg C” when the object is redeployed, the InSQL configuration database reflects the change.

When an AutomationObject configured for history is undeployed, all history remains in the InSQL historian. The history data can be examined in the future even if the AutomationObject is no longer deployed.

C H A P T E R 5

Working with Events and Alarms

Industrial Application Server users can automate the detection, notification, historization and viewing of either application (process) events and alarms or system and software events and alarms. Events and alarms are runtime occurrences that are sent to InTouch and historized using the InTouch Alarm Logger.

This chapter describes:

- the difference between events and alarms
- how to configure alarm providers
- how alarms and events are generated, reported and handled by clients

Contents

Events and Alarms

Events and alarms are different.

- An event is an occurrence of a condition at a certain point in time. ArchestrA can detect events, store them historically and report them to various clients.
- An alarm is the occurrence of a condition that is considered abnormal (generally bad) and requires immediate attention from a user. Alarms are a special type of event that have state and must be acknowledged by a user. ArchestrA handles the real-time reporting of alarms in a special manner and provides special clients for their viewing.

Examples of events include:

- A plant process has started (for example, a new batch starts).
- The operator changed a plant operator parameter (for example, a setpoint on a temperature controller).
- The system engineer changed the runtime system configuration (for example, deployment of a new AutomationObject).
- The system engineer started or stopped a system component (for example, stopping an AppEngine).
- A WinPlatform come back online after a failure or shutdown.
- A user logged into the system.
- Detection of a severe software problem (for example, a failed Application Object component).

Examples of alarms include:

- A process measurement exceeded a predefined limit (for example, a high temperature alarm).
- A process device is not in the desired state (for example, a pump that should be running has stopped).
- The system hardware is not operating within desired limits (for example, the CPU performance on a WinPlatform exceeds 99% for an extended time).

The following items are not considered alarms or events:

- Configuration actions within the Galaxy Repository (for example, import or check out of an object).
- Installation of Bootstrap on a computer.
- A message sent to the ArchestrA Logger. Sometimes, certain software events log a message in addition to generating an event. Logger messages are not events.
- Viewing a window in InTouch.

ApplicationObjects include built-in alarming and event reporting capabilities. These capabilities must be configured in the IDE to perform the alarm/event function. After an Area object is deployed, alarms generated by Automation Objects belonging to the Area can be monitored and controlled by alarm clients.

InTouch as the Alarm and Event Client

InTouch runtime clients subscribe for event reports from the Galaxy. For Industrial Application Server alarming to function, the following is required:

- A WinPlatform is deployed to and is running on the InTouch client node. Depending on your configuration, you can specify it as an InTouch Alarm Provider. For more information about when you want to do this, refer to the *Industrial Application Server Deployment Guide*.
- One or more Area objects are deployed and running.

- The InTouch operator can view alarms, acknowledge alarms, disable alarms and enable alarms from InTouch.
- The InTouch alarm client is placed on a window and configured as an alarm consumer of the Galaxy.
- The InTouch client (WindowViewer) is running.
- The source AutomationObject is on scan.
- The source AutomationObject’s Area is on scan.

  Note The Area does not have to be on scan if the AutomationObject of interest is a WinPlatform, AppEngine or DI Object. In that case, only the object itself needs to be on scan.

- Alarming is enabled for the target AutomationObject.
- An Industrial Application Server alarm can occur in an AutomationObject assigned to an Area or in a WinPlatform, AppEngine or Device Integration Object. Both sets of objects must be subscribed to by the InTouch Alarm Client through the InTouch Distributed Alarm system.
- Industrial Application Server reports the alarm to the InTouch Distributed Alarm System, which reports it to InTouch.
- In the InTouch alarm client display, the alarm information for the new unacknowledged alarm, including all required fields, appears. The new alarm is in the unacknowledged state.
- The new alarm is recorded in InTouch Event History.
- The InTouch security setting is configured for ArchestrA security mode.
- The user is logged into InTouch, and is authorized to acknowledge alarms for the AutomationObject that is in the alarm state.
- The user opens the InTouch alarm view, selects the unacknowledged alarm and acknowledges it, specifying an optional comment field.
- The Industrial Application Server validates that the user has sufficient security privileges to acknowledge the alarm.
- Other InTouch alarm clients show that the alarm is acknowledged.
- The alarm acknowledge event is recorded in InTouch Event History.
- When the alarm is acknowledged and returns to normal state, it is cleared from the InTouch alarm display.

Caution! If you do not have sufficient privileges to acknowledge alarms, you can still acknowledge the alarm, specifying an optional comment. But the Galaxy rejects the acknowledgement 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.

Disabling an AutomationObject’s Alarms

A user with the correct permissions can disable an AutomationObject's alarms through an InTouch window. This is done by requesting that the enable/disable state of the AlarmModeCmd attribute of any one of the following objects be disabled:

- The AutomationObject.
- The AutomationObject's container (if any).
- The AutomationObject's Area.

An event, including the user's name, is generated indicating the object's alarms are disabled. Active alarms for the disabled object disappear from the InTouch Alarm display.

When you disable alarms in an AutomationObject, all alarms for the AutomationObject are disabled. When you disable alarms in the object’s Area, alarms for all contained Areas and AutomationObjects belonging to those Areas are also disabled. When you disable alarms in a container AutomationObject, alarms for all contained AutomationObjects are also disabled.

Enabling an AutomationObject’s Alarms

To enable an AutomationObject’s alarms, you must ensure that the AlarmModeCmd and AlarmInhibit attributes are enabled for the AutomationObject, its container and its Area. An event, including the user’s name, is generated indicating the object’s alarms are enabled.

Configuring an AutomationObject for Alarms

Alarming capabilities are a part of object templates, but they are not implemented until the object is configured in the IDE.

Configuring an AutomationObject to be an alarm provider includes:

- Deciding whether alarm notification is desired for each possible alarm condition in the object. For example, a command timeout alarm for a valve if the output command fails to cause the valve to move.
- Editing the object and configuring an attribute that specifically commands alarm notification.
- Configuring the alarm configuration attributes. Most typically, the fields that require configuration are Category, Priority and Description.
- Configuring any limit fields for triggering alarm detection (for example, the feedback timeout time limit).

To configure a WinPlatform to be an InTouch alarm provider

1. Check out the WinPlatform object.
2. Open the object’s editor.

3. On the **General** page of the object's editor, select InTouch **Alarm Provider**.
4. Enter specific Areas in the **Alarm Areas** box, or leave the box blank to include all Areas.
5. Save and close the object's editor.
6. Check in the object to the Galaxy.
7. Deploy the object to its target computer in an on scan state.

User-Defined Alarms

IDE users can add alarms detection and reporting capabilities to AutomationObjects that were not originally developed to detect alarms. This capability is provided through a combination of scripts and alarm extensions. See "Enhancing an Object's Functionality" for more information about this capability.

Objects Generating Alarms/Events

After AutomationObject instances that are configured for alarm and event detection are deployed and put on scan, they begin checking for alarm and event conditions. When an alarm or event arises, the condition is reported to alarm/event distributors, which are other AutomationObjects running on the same AppEngine.

Alarm/event distributors include:

- Area AutomationObjects: All Domain AutomationObjects and Area objects report detected alarms through the Area, which distributes them to alarm and event clients.
- WinPlatform AutomationObjects: Report their own alarms and events.
- AppEngine AutomationObjects: Report their own alarms and events.
- Device IntegrationObjects: Report their own alarms and events.

WinPlatforms, AppEngines and Device Integration objects do not report their alarms and events to Area AutomationObjects even though they belong to Areas. This allows alarm clients to receive alarm notifications without any dependencies on Area AutomationObjects. 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.

Area AutomationObject

The Area AutomationObject plays a key role in alarm and event distribution. All AutomationObjects belong to an Area. Areas can contain sub-Areas. Alarm and event clients are configured to subscribe to a set of Areas. Thus 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.

Alarm, Event Subscription

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, and if WinPlatforms, AppEngines or Device Integration objects belong to an Area, those objects are also directly subscribed to.

When a notification distributor receives a alarm/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/event subscription requests do not include filters (for example, all alarms greater than a specific priority value). Consequently, 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.

Alarm enable/disable

Alarms (not events) can be enabled or disabled in the runtime environment. This action can be taken at the Area level, at a container object level or at an individual object level.

Note Individual alarms in a single object cannot be enabled/disabled selectively.

Runtime alarm actions include:

- Enabled: All alarms for the object are reported to clients and historized normally.
- Disabled: No alarms for the object are detected. The alarm is effectively return-to-normal until the alarm is reenabled.

Important! Silenced mode is reserved for future use. It currently functions in the same way as Disabled.

The table below defines the resulting Enable/Disable state given the state of other objects in a hierarchy. The most restrictive state controls.

Object's Area Alarm Mode	Container Alarm Mode (if any)	Object Commanded Alarm Mode	Resulting State for Object
Enabled	Enabled	Enabled	Enabled
Disabled	Any	Any	Disabled
Any	Disabled	Any	Disabled
Any	Any	Disabled	Disabled

Alarm Provider Behavior in a Network Outage

A WinPlatform, when configured and acting as an InTouch Alarm Provider in the runtime environment, sends an alarm through the InTouch Distributed Alarm System to InTouch clients when it has lost communication with an Area that it subscribes to. This condition typically occurs during a network outage with computers hosting those Areas.

In such a network outage condition, 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 has been 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.

Distinctions Between Alarms and Events in Industrial Application Server and InTouch

InTouch and Industrial Application Server both implement alarms and events. There are similarities and differences between the two models as described in the table below.

Item	InTouch	Industrial Application Server
Alarm configured/ detected by	Within a Tag	Within an AutomationObject
Alarm Classes (client column)	Only certain Classes of alarms are supported/ 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. Mapped from Category.

Item	InTouch	Industrial Application Server
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 ".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)	Tag comment = short description.	AutomationObject 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.
State	UNACK, ACK, RTN	Alarm state provides equivalent concept and can be mapped.
Value, CheckValue	Only static values sent with alarm message.	Static values and dynamic references are provided.
Ack	All alarms sent to client and require acknowledgement regardless of priority.	All alarms sent to client and require acknowledgement regardless of priority.
History	Alarm state changes are logged to event history and displayed on historical client.	Alarm state changes are logged to event history and displayed on historical client.

C H A P T E R 6

Enhancing an Object’s Functionality

Each object imported into your Galaxy has a predetermined set of operations and functions it is capable of executing. That behavior, which cannot be modified, is em bedded in the object by the object's developer. The object's capabilities can be enhanced in the IDE through the editor's object enhancement pages.

This enhancement capability allows you to add additional functions to the object while not changing the object's original behavior. These enhanced capabilities are called object extensions, which can be added to derived templates and object instances. Base templates cannot be extended.

There are three types of object extensions:

- Scripts. You can add to an object’s functionality on the **Scripts** page of the object’s editor.
- User defined attributes. You can add to an object’s functionality on the **UDAs** page of the editor.
- Attribute extensions. You can extend the functionality of an object’s already-existing attributes on the **Extensions** page of the editor.

This chapter describes how to use scripting, UDAs and attribute extensions to enhance the functionality of an existing object in your Industrial Application Server application.

For more information on enhancing an object’s functionality, refer to your *Factory Suite A^2 Deployment Guide.*

Contents

Configuration Editor Extension Pages

To access object extension functions in the IDE, open any object's configuration editor. The last three pages, **Scripts**, **UDAs** and **Extensions**, show the object extensibility part of the object's editor.

Select the desired page and see the specific instructions about each page below.

Extension Inheritance Characteristics

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

- Renaming an extension in the template to which it was originally added causes it to be renamed in all other objects derived from the template. This change occurs when the template is checked in.
- Removing an extension from the template to which it was originally added causes it to be removed from all objects derived from the template. This change occurs when the template is checked in.
- Adding an extension to a derived template that has objects derived from it causes it to be added to the derived objects.
- Adding extensions to derived objects that are duplicates of parent object extensions (the same name and type) are not allowed.
- Adding an extension that causes a name collision with an existing attribute extension is not allowed.
- Checking in a template with a new extension that causes a name collision with an existing attribute in a derived object succeeds. The template's definition of the extension overrides that of the derived object.

Scripting

A script lets you execute commands and logical operations based on specified criteria being met. For example, a key being pressed, a window being opened or an attribute's value changing. By using scripts, you can create a wide variety of customized and automated system functions.

A script adds behavior that executes when the object that contains the script is deployed and either:

- is on scan in the runtime environment or
- changes scan or startup/shutdown state

The script typically is executed based on attributes of the object that contains it, but it can be executed by another script based on changing values of attributes of more than one object.

Note Attribute data from any object located off the AppEngine on which a startup script runs is not available when that script first executes. Typically, this is true for several scan cycles. Before executing critical functionality, you can include Quality testing scripting to determine if this data is available.

Scripts do not support public variables that behave like attributes of the extended object. But scripts and user defined attributes (UDAs) can be combined to create complex customized behavior that extends beyond the object’s basic functionality. See User Defined Attributes for more information.

The **Scripts** page provides a location for you to write custom scripts that execute in the runtime environment of the extended object. The execution of the script is conditionally triggered.

You can add script extensions to any derived template or instance in the Galaxy.

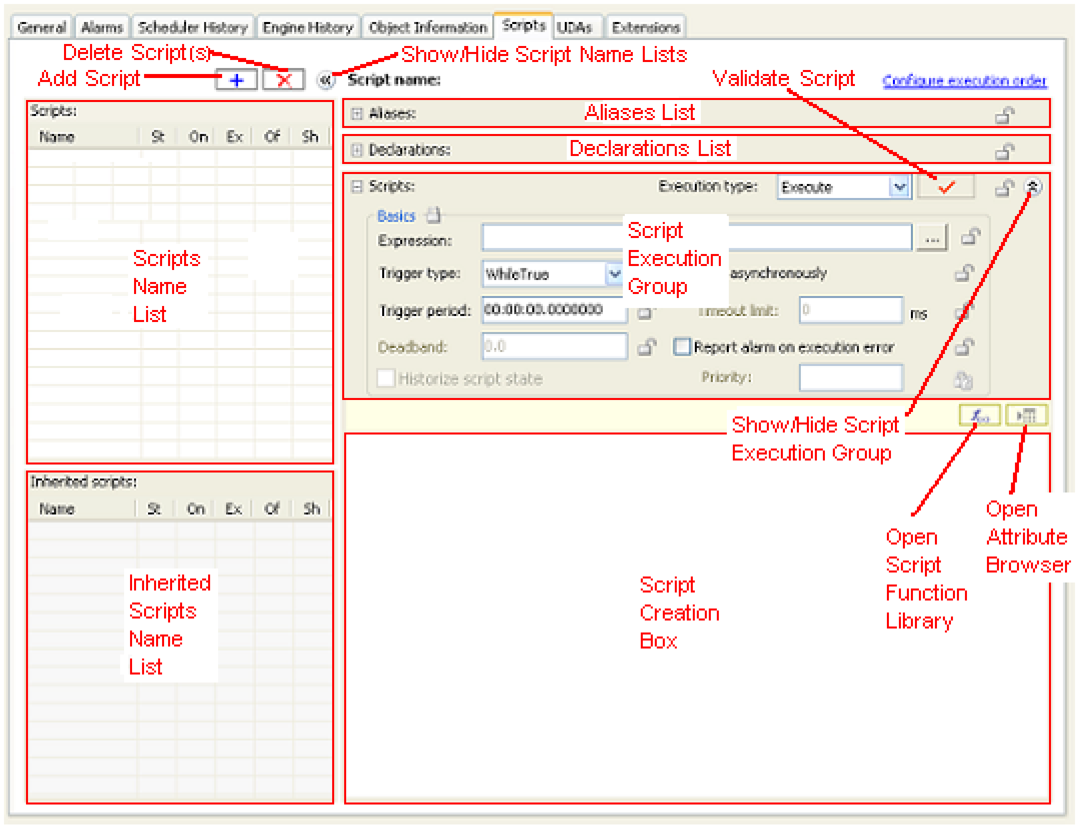
Industrial Application Server scripting supports the following kinds of functionality for the runtime behavior of AutomationObjects:

- Supervisory control algorithms
- Batch control algorithms
- Complex alarm detection
- Database storage and retrieval
- Report generation and publication
- .NET System namespace

See QuickScript .NET Scripting Language for specific script functions, operands and sample code.

The Scripts Page

The **Scripts** page has six main functional areas and a set of function buttons. These are described below.



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

- **Scripts Name** 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.
- **Inherited Scripts Name** List: Shows all scripts associated with the object's parent. The object automatically is associated with those scripts. The columns indicate which kind of trigger the script uses: Startup, On Scan, Execute, Off Scan and Shutdown.
- **Aliases** List: Shows aliases that apply to the script you are working on. Aliases are shortened, more logically descriptive names for longer ArchestrA reference strings that can be used in the script.
- **Declarations** List: Provides a place to add variable declaration statements, such as `DIM MyArray[1] as FLOAT;`. These declarations are persisted and apply only to the script in which they are declared.
- **Script Execution Group**: Provides a location in which you set the triggering conditions that cause the script to run in the runtime environment. See the "To create and associate a script with an object" procedure below for descriptions of triggers and when they are executed.
- Script Creation Box: Shows the script you are writing.
- **Configure Execution Order**: Use to set the execution order of multiple scripts (inherited and local) associated with this object.

Important! The existence and execution order of scripts associated with an object are inherently locked at each stage of development - 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 executed in any order before and after the template's block of scripts. The derived template's execution order is then treated as a block in any downstream derived templates or instances.

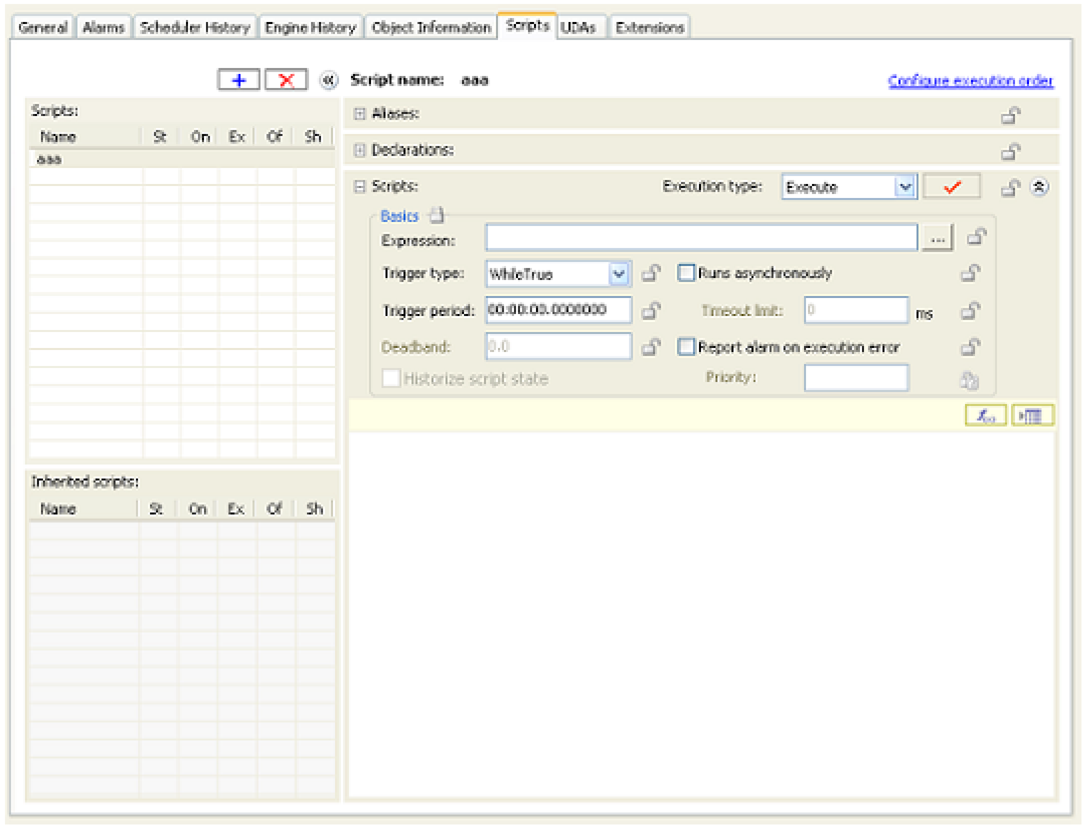
- **Historize Script State**: Select to send the state of the script to InSQL, the ArchestrA historian.

The buttons include:

- Add Script: Click to add a new script to the object.
- Delete Script(s): Click to delete the script(s) selected in the Scripts Name list.
- Show/Hide Script Name Lists: Click to switch between hiding and showing the Script Name and Inherited Scripts Name lists. This provides more room for the remainder of the user interface.
- Validate Script: Click to validate the expressions already written into the script.
- Show/Hide Script Execution Group: Click to switch between hiding and showing the script execution group. This provides more room for the script creation box.
- Display Script Function Browser: Click to open the **Script Function Browser** where you can select a set of functions to include in your script.
- Open Attribute Browser: Click to open the **Attribute Browser** where you can select particular object attributes and properties to include in your script.

To create and associate a script with an object

1. On the **Scripts** page of the object's editor, click the plus (+) sign. A script is added to the **Scripts Name** list. The name entry point is shown in edit mode. Select a script name and click the delete button (red X) to remove it from the list. You can multi-select script names with **SHIFT+CLICK**.
2. Type a name for the script and press **ENTER**. The **Scripts** page might appear as follows, assuming a script name of "aaa".



Script name: aaa

Scripts:

Name	St	On	Ex	Of	Sh
aaa					

Scripts: Execution type: Execute

Expression:

Trigger type: WhileTrue

Runs asynchronously

Trigger period: 00:00:00.0000000

Timeout limit: 0 ms

Deadband: 0.0

Report alarm on execution error

Historize script state

Priority:

Inherited scripts:

Name	St	On	Ex	Of	Sh

3. Select a trigger for executing the script in the runtime environment. **Execution Type** triggers include: Startup, On Scan, Execute, Off Scan and Shutdown. If you select any trigger except Execute, the **Basics** group is made unavailable. That means that the script is triggered whenever the object starts up, goes on scan, goes off scan or shuts down. If you select Execute, the **Basics** group is available.

Note 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.

The message exchange system cannot access Off Engine data during an on scan script trigger.

4. 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 runtime environment, the script is executed. See the table below for more information.

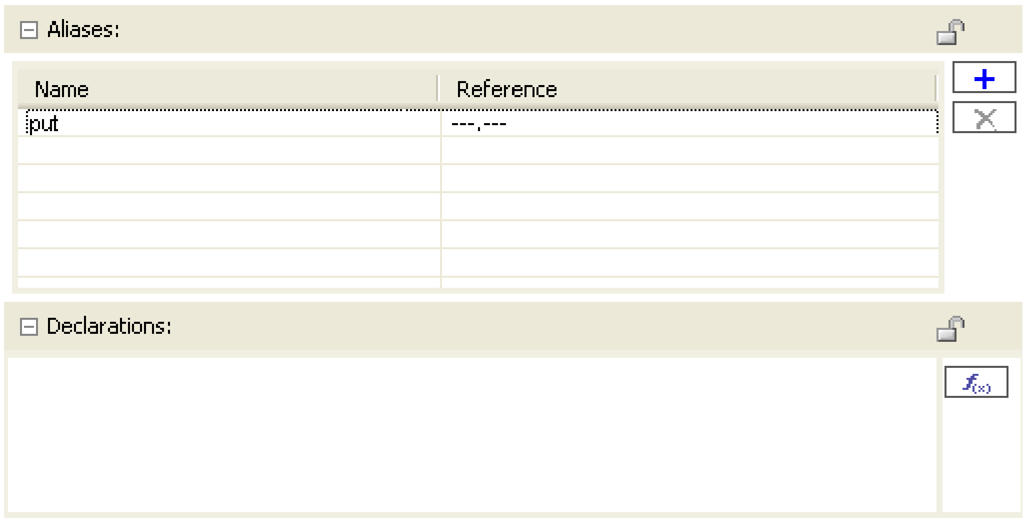
Note The **Trigger Period** format is as follows:
Hours:Minutes:Seconds:Milliseconds
For example, 3 hours, 5 minutes and 10.5 seconds is:
03:05:10:5000000

Trigger Type	Description
Periodic	Script is executed based on a time interval specified in the **Trigger Period** box. A time interval of zero (0) executes the script during every scan. This trigger does not require an expression.
While True	When the object containing the script is going OnScan, a While True script evaluates its expression at the next scheduled scan period of the AppEngine, executes if true and then periodically thereafter at the trigger interval. Script is executed as long as the **Expression** value evaluates to be true. A Trigger Period is required; zero (0) causes the expression to be evaluated at the AppEngine scan period and non-zero means the expression is evaluated at the specified time interval.
On True	When the object containing the script is going OnScan, an On True script evaluates its expression at the next scheduled scan period. Script is executed at the transition between the expression going from false to true.
On False	When the object containing the script is going OnScan, an On False script evaluates its expression at the next scheduled scan period. Script is executed at the transition between the expression going from true to false.
Data Change	Script is executed when the value or quality of the expression changes. The expression evaluates to a single, non-arrayed numerical value of the following types: integer, real, time, elapsedtime, string, double, Boolean, custom enumeration and quality. Deadband can be specified for all types, the deadband type always being specified as a double. Deadband units for time and elapsedtime types are milliseconds. Deadband is always ignored for strings since 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. After the object is put onscan, Data Change-triggered scripts are executed on the AppEngine's next scan period and then on each subsequent scan period in which the value or quality changes.
While False	When the object containing the script is going OnScan, a While False script evaluates its expression at the next scheduled scan period, executes if false and then periodically thereafter at the trigger interval. Script is executed as long as the **Expression** value evaluates to be false. A Trigger Period is required; zero (0) causes the expression to be evaluated at the AppEngine scan period and non-zero means the expression is evaluated at the specified time interval.

Note See The UDAs Page for information about using UDAs in scripts.

Note 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.

5. Set the **Runs Asynchronously** and associated **Timeout Limit** parameters as needed (see “Synchronous/Asynchronous Scripts” later for more information). 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. Also, select **Historize Script State** to store the state of the script in you application’s historian.
6. Type variable declarations about the script you are writing in the **Declarations** list.



Name	Reference
put	---.---

7. Set up aliases for object reference strings in the **Aliases** list. This helps 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 just link external attributes to the alias names. In the **Aliases** list, click the plus (+) sign to add a new alias. An alias is added to the list; the name entry point is shown in edit mode. Double-click the **Reference** entry (which puts it into edit mode), and enter a reference string for the alias. Optionally, click the button that appears at the end of the Reference block to open the Attribute Browser for easy selection of an object’s attributes. Select an alias and click the delete button (red X) to remove it from the list. **Shift+click** to select multiple aliases.
8. 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, respectively, in your script.
9. Use the validate script button to determine whether your script syntax is valid.

10. Click **Configure Execution Order** and use the dialog box when you have more than one script associated with a single object. This ordering function does not apply to asynchronous scripts. If a script is added to an object instance derived from a template that contains scripts, the new script automatically defaults to running after the derived scripts.
11. When you are done creating your script and setting its execution triggering parameters, save and close the object's editor. After the object is deployed and on scan in the runtime environment, the script executes as designed.

The following characteristics apply to the scripting environment:

- Change the name of a script at any time by renaming it in the object's editor.
- Script text has no length limitations.
- Selecting a script function from the **Script Function Library** dialog box adds it and its calling syntax to the script text.
- You can save a script with syntax errors, but the object cannot be deployed until the errors are resolved.
- A script semantics check is also done, helping you avoid syntactically correct but semantically incorrect combinations such as two statements declaring the same variable (variables can only be declared once in a single block).
- In the runtime environment, a script execution error causes the script's current execution span to end. Script execution is reattempted on the next AppEngine scan after the error was encountered.

Synchronous/Asynchronous Scripts

Synchronous scripting mode is the default for running scripts in the runtime environment. This mode represents serial execution of scripts while an object is running on scan.

Asynchronous scripting mode involves 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.

Note Asynchronous script timeout occurs on the next scan of the AppEngine, even if the script's timeout setting is faster than the AppEngine's scan rate.

Alarm Type Errors

Execution errors are any errors detected by the script execution engine that make the script execution invalid and causes it to stop.

Note Alarm notification only applies for Execute scripts, not for those that execute at Startup, Shutdown, On Scan or Off Scan transitions.

The following alarming errors are supported:

- Timeout: To prevent the possibility that a script can cause an overrun of the AppEngine scan cycle (infinite loop). All synchronous scripts are subject to the timeout period set in the configuration editor of the AppEngine that contains the script or the AppEngine hosting the object that contains the script. Asynchronous script timeout limit is set on the **Scripts** page of the object containing the script, but the timeout value is accurate only within the resolution of the hosting AppEngine's scan period.
- Overflow: Applies only to float, double and integer data types.
- Division by zero: Applies only to integer data type.
- `.NET` call execution: A script encounters an exception during a `.NET` function call and an error is sent to the Logger. Also, the execution alarm boolean condition is raised.

Script Functions

Industrial Application Server architecture supports the use of script functions, pre-compiled functions that can be used in scripts. The preferred languages for written script functions are `.NET` languages (for example, `VB.NET` or `C#`), although they can be written in other languages and imported as WDF files (legacy InTouch) and COM objects (C++ and VB).

Script functions are grouped into libraries, categorized by function and available to be called from any script or script function. They do not maintain states between calls (exception: if a variable is declared, section state is maintained between execute calls), they can receive in and out parameters when called and can return a value if required.

By default, six binary libraries of script functions are available for use in script development:

- Math functions
- String functions
- System functions
- Miscellaneous functions
- Types functions
- WWDDE functions

To insert a script function into a script, place the cursor where needed, click the **Display Script Function Browser** button , select the function in the **Script Function Browser** dialog box, and click **OK**. The function is inserted in the script. Text that requires your input is highlighted.

Important! You can also create new binary libraries through Visual Studio.NET. COM libraries developed with Visual Studio 6 or earlier can be imported as well but, unlike .NET components, COM libraries are not automatically deployed with the associated object. You must install and register them manually by navigating to their folder location at a DOS prompt, and then typing: `Regsvr32 <.dll filename>`. Optionally, you can export them as `.aaSLIB` files (a cab file), manually change the XML file in the cab file to designate the library as a `COM` object requiring registration and then reimport the `.aaSLIB` file. The resulting components are deployed and registered. See the procedures below for importing, exporting and modifying the contents of an `.aaSLIB` file.

Note QuickScript .NET language requires parentheses around script function arguments. See QuickScript .NET Scripting Language for more information.

Guidelines when Developing and Using Script Function Libraries

Keep the following in mind when developing and using script function libraries:

- Nested public structures and classes are not supported in imported script function libraries.
- Script function libraries must follow recommended naming guidelines. See Allowed Names/Characters for naming guidelines. To prevent name clashes across different library-providing vendors, the vendor's name should prefix the library name (for example: VendorCompany.ComponentName.dll.).
- The concept of developing additional binary libraries is similar to developing new InTouch script functions in C, using the InTouch Extensibility Toolkit.
- Scripts execute in the same process as the object to which they are attached. As a result, binary script functions can disrupt the function of the AppEngine if poorly developed.

When you import the library, if the new name of the library is identical to the name of a library that was previously imported into the Galaxy, you are prompted to replace the library or rename the library being imported. Also, if you replace an existing script function library and redeploy the objects using it, you must restart the objects' AppEngine for the new script function library to take effect.

Caution! If you import a script library that has the name of an existing script library, you may see problems when you deploy. The deployed objects may not be able to "see" the scripts in the library. Make sure you follow the steps below every time.

Important! The AutomationObject export command does not automatically export script functions used by exported objects if those script functions were supplied through an imported library. If you import those objects into a second Galaxy, you must export the script function library from the original Galaxy and import it into the second Galaxy. Only then do the scripts associated with those objects run properly.

To develop and import the library

1. Develop the library in a development tool such as Visual Studio .NET.
2. Import the library into the IDE. Do the following:
   - Point to **Import** on the **Galaxy** menu and then click **Script Function Library**.
   - In the **Import Script Function Library** dialog box, browse to locate the library file (`.aaSLIB`, `.dll`, `.wdf`, `.tlb` and `.olb` extension). Select the file and click **Open**.
   - Click **OK** in the final message box. The Galaxy imports the designated library makes its functions available to all IDEs connected to the Galaxy through the script function browser on the **Scripts** page of the object's editor.
3. Use a function from the imported library in a script on a test object.
4. Deploy the test object to a non-production (test) AppEngine.
5. Debug the script and test.

Note If you reimport your script library, you must do the following steps. If you do not do the following steps, your new script library may not update after you import.

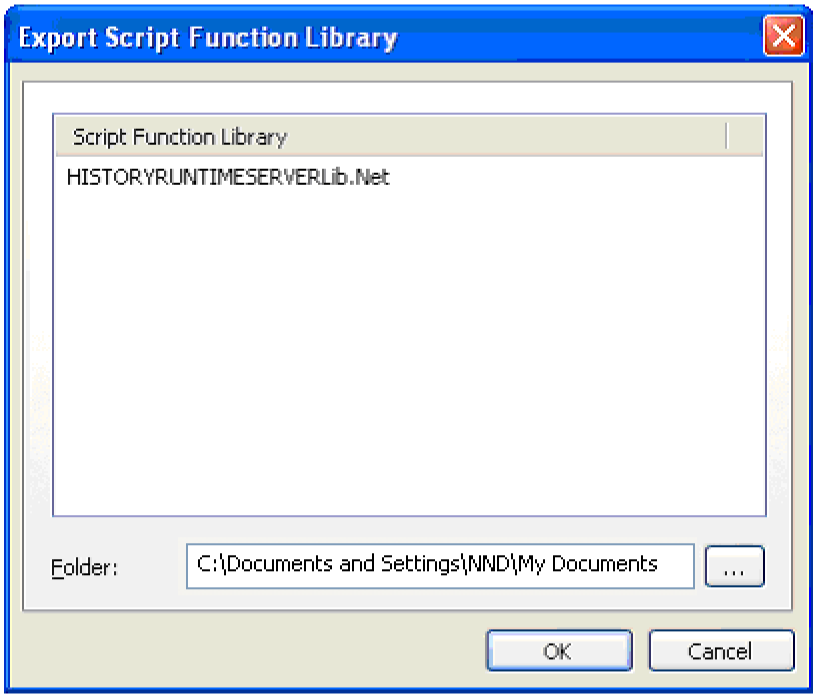
6. When you are done developing, importing, and testing the library, validate the objects in your Galaxy. All the objects that use the new script library are marked for Software Upgrade Required.
7. Undeploy the objects that are marked for Software Upgrade Required.
8. Shutdown the AppEngine that has the objects marked for Software Upgrade Required. This forces the .NET assembly to unload from memory.

Note Step 8 is a critical step. You may get errors in the deployed objects later if you do not shut down the AppEngine now.

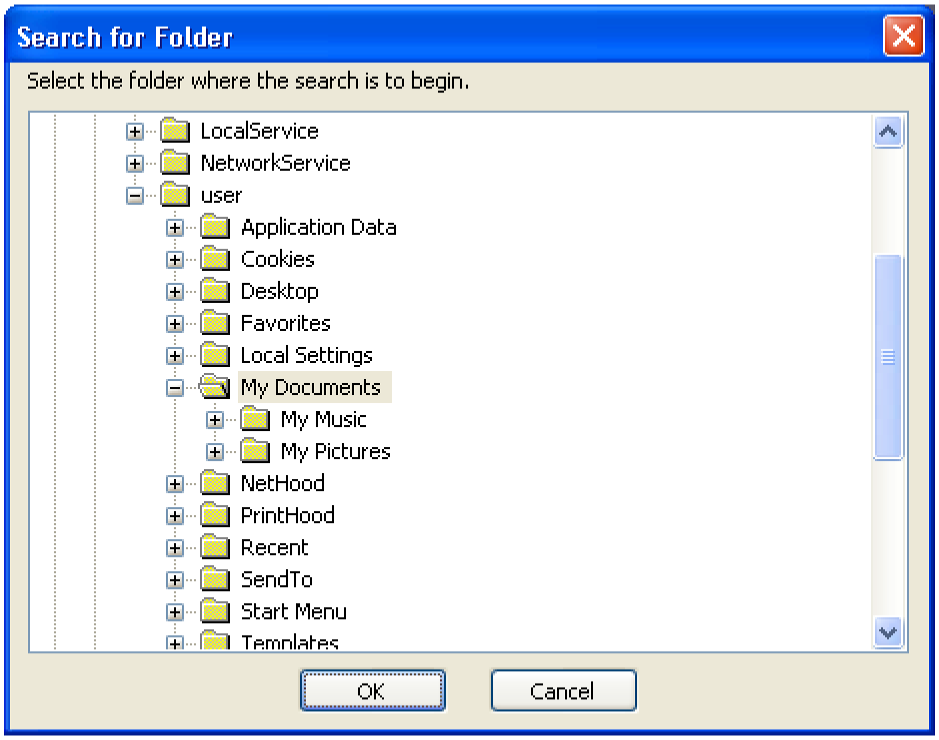
9. Restart the AppEngine.
10. Redeploy the objects marked Software Upgrade Required from the previous steps. You may want to retest at this point.

To export a script function library

1. Point to **Export** on the **Galaxy** menu and then click **Script Function Library**. The **Export Script Function Library** dialog box appears:


Export Script Function Library
Script Function Library
HISTORYRUNTIMESERVERLib.Net
Folder:
C:\Documents and Settings\NND\My Documents
...
OK
Cancel

2. Select the library from the list. If the desired library does not appear, enter the appropriate folder path or click the browse button to go to the desired path. The **Search for Folder** dialog box appears.



3. Click **OK** in the Search for Folder dialog box and then again in the Export Script Function Library dialog box. A standard Microsoft Save As dialog box appears.
4. Enter a path and filename for the exported library and click **Save**. The library is saved to a `.aaSLIB` file.

To manually modify the XML file in an `.aaSLIB` file to designate a script function library as a COM object requiring registration

1. In Windows Explorer, rename the ???.aaSLIB file to ???.aaSLIB.cab where ??? is the name of the script library being modified.
2. Create a working folder on your computer.
3. Double-click the `.cab` file. The files included in the `.cab` file appear.
4. Select all files, right-click and then click **Extract**.
5. Browse to the working folder and click **Extract**.
6. In Windows Explorer, browse to the working folder and open the __aaCabinetFileList`.xml` file in Internet Explorer.

Note The root node in this .xml file is <aaCabinetFileList>. Under this root are <FileItem> xml nodes. Each node provides a mapping via the FileCode xml attribute and the FileName xml attribute.

7. Find the FileItem with the FileName = *.Net.xml and note the FileCode.
8. Open in Notepad the .tmp file with the same name as the FileCode you noted in the previous step.
9. Under the <dependentFiles> xml node, locate the <file> xml node containing the desired COM object requiring registration. Modify the <registrationType> element from “eNormal” to “eComDLL”. (Do **not** change the <registrationType> for the ???.Net.dll node.)
10. Save the `.xml` file.
11. Open a Command Prompt window and navigate to the working folder.
12. At the command prompt, type: “`Cabarc N ???.aaSLIB *.*`” where `???` is the same filename as in step 1.

Note You can download the `.cab` file utility, `cabarc.exe`, from the Microsoft website. The utility must be copied into your `\system32` folder.

The resulting `.aaSLIB` file can be imported into the Galaxy. The script function library is automatically deployed and registered with objects that use the library.

Important! If a `COM` library developed with `Visual Studio 6` or earlier has file dependencies, ensure that you include those files in the .aaslib file and include a `<file> ... </file>` entry in the .xml file under `<dependentFiles>` for each dependent file. Otherwise, objects that use the script function library are not deploy properly.

Important! To successfully reimport the `.aaSLIB` file, you must first update the version data in the script function library `.dll`. Otherwise, the `.dll` is not properly registered as a COM object and is not redeployed with the object.

Locking Scripts

The following rules apply to locking scripts in templates:

- The name of a script and its existence is implicitly always locked, which means:
  - The existence of the script cannot be deleted in derived objects.
  - The name of the script cannot be changed in derived objects.
  - Renaming the script in the template causes the name to change in all derived objects.
  - A script can be deleted in the template after derived objects have been created. The script then disappears from the derived objects.
  - A script can be added to the template after derived objects have been created. The script then appears in the derived objects.
  - You can add scripts to derived objects without limitation.
- The script text can be locked or unlocked in a template. There is script text for Declarations, Execute, Startup, Shutdown, On Scan and Off Scan. Each cannot be separately locked in the script editor. Instead, a single group lock is supplied that locks or unlocks all at once.
  - Once locked, 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 that the entire list of alias names is locked, including the number of items in the list. New alias names cannot be added 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 a script is added to a template, all properties of the script are editable.
- When a script is added to an instance, all properties of the script are editable except for the lock properties (a lock is never editable in an instance).

Important! Typically an expression uses attribute references. If you want to lock the expression and the associated script in a template, then you should 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 that template share the copy of the value of that locked attribute. A change to that value is only allowed in the template that locked it, and that 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 (for example, modifying the script text) propagate (and appear) in all derived templates and instances. If instances are deployed, they are marked pending update status. Once 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 since 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 private copy and the value of that unlocked attribute can be changed. 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.

User Defined Attributes

The purpose of the User-Defined Attribute (**UDAs**) page is to allow a user to do the following:

- 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 comprise it.
- Set alarms and historization for the new attribute (done on the **Extensions** page).

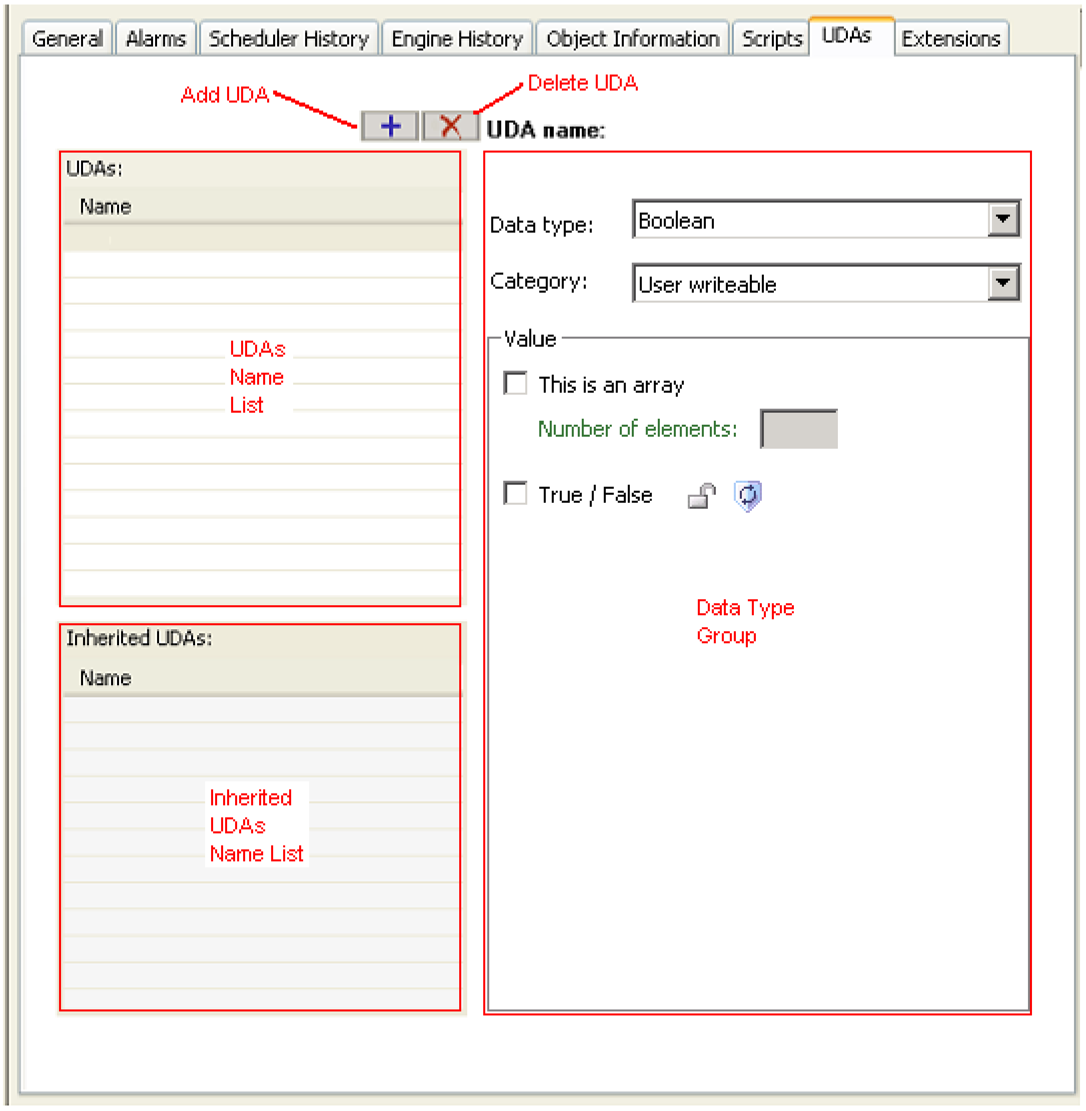
- Define security and references to other objects (done on the **Extensions** page).

Note After you add an attribute to an instance, it appears in the Attribute Browser list for use with the scripting and attribute extension functions.

A UDA can be added to a template or instance. When added to a template, the UDA, its data type and category are automatically locked in its instances. If other parameters (initial value, lock and security classification) are locked in the template, the UDA cannot be changed in any instances derived from the template. If those other parameters are unlocked in the template, initial value, lock and security are editable and lockable in derived templates. When unlocked in either the base or derived template, the value is editable in instances.

The UDAs Page

The **UDAs** page consists of three main functional areas and a set of function buttons. These are described below.



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

- **UDAs Name** List: Lists all UDAs currently associated with the object.
- **Inherited UDAs Name** List: Lists all UDAs associated with the object's parent. The object automatically is associated with these UDAs.
- **Data Type** Group: Provides options for configuring the selected UDA.

The buttons include:

- Add UDA: Click to add a new UDA to the object.
- Delete UDA: Click to delete the UDA selected in the UDAs Name list.

In the **Data Type** group, allowed data types are **Boolean**, **Integer**, **Float**, **Double**, **String**, **Time**, **ElapsedTime** and **InternationalizedString**. Allowed categories are **Calculated**, **Calculated Retentive**, **Object Writeable** and **User Writeable**. Writeable attributes can be locked. If you select **Calculated** for an attribute, only scripts running on the same object can write to the attribute.

You can develop an array for each data type except **InternationalizedString**. Select **This is an Array** and set the dimensions in the **Number of Elements** box. As you increment the **Number of Elements** number, additional layers are added to the **Value** box.

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 implement a True value (leave the check box unselected to implement a False value). For an arrayed Boolean, select the default value and type either true or false.

When using UDAs in scripting, note the following:

- When using **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 OnScan and Execute scripts (that is, scripts with those types of **Execution Type** triggers), but not Startup scripts.
- **Calculated Retentive** UDAs must be initialized in Startup scripts, and can be initialized in OnScan and Execute scripts. The main purpose of a **Calculated Retentive** UDA is to retain the attribute's current value after a computer reboot, redundancy-related failover, or similar occurrence in which valid checkpoint data is present. Therefore, your Startup script should contain a statement testing the Boolean value of the attribute, `StartingFromCheckpoint`, on the object's AppEngine. If the value is TRUE, you should not initialize the UDA; if the value is FALSE, you should initialize the UDA.

See The Scripts Page for more details about **Execution Type** triggers and related scripting information.

To create and associate a UDA with an object

1. On the **UDAs** page of the object's editor, click the plus (+) sign. A UDA is added to the **UDAs Name** list; the name entry point is shown in edit mode. Type the desired UDA name. To delete a UDA from the list, select it and then click the delete button (red X).

Note A UDA name that starts with an underscore (_) as the first character of the name is treated as 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 user defined attribute, it is available, like all attributes, when extending the object further on the **Extensions** page. If you extend a user defined attribute and then either delete or rename the user defined attribute, all object extensions added to the object on the **Extensions** page are lost.

2. In the **Data Type** group, select the correct **Data Type** for the new attribute. The remainder of the **Data Type** Group shows content dynamically depending on your selection in the **Data Type** list.

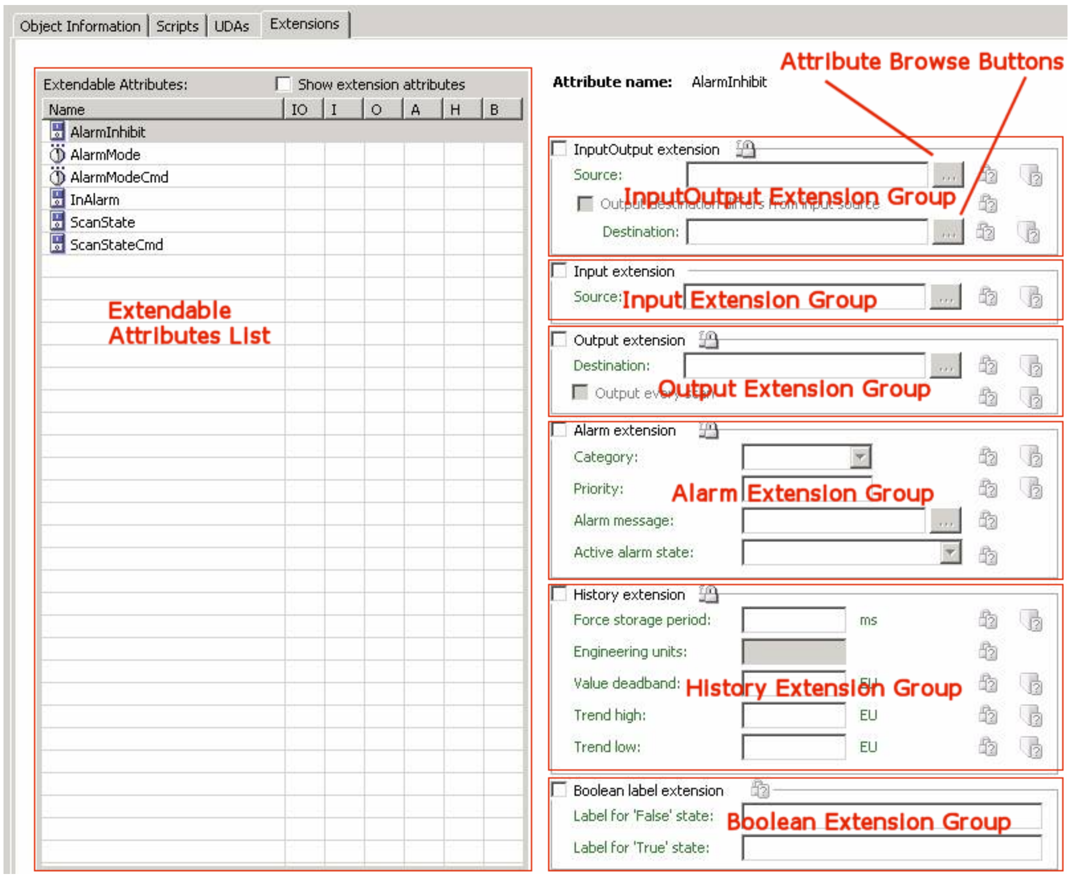
3. Set the remaining parameters in the **Data Type** group as desired.
4. Lock the value if desired. The lock symbol is available only when you are extending a template. Otherwise, it indicates the lock condition of the value in the parent object. If the value is an array, the lock condition applies to the array dimension also.
5. Set the security classification for the attribute. See “Security Icons“ for more information.
6. Save and close the object editor to include the new attributes in the configured object.

Attribute Extensions

The purpose of the **Extensions** page is to allow you to configure an already-existing attribute for input, output, alarm and history functionality not embedded in the original object.

The Extensions Page

The **Extensions** page consists of five main functional areas. These are described below.



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

- Extendable Attributes List: Lists all attributes currently associated with the object. The list can include those added through the UDA object extension function. Select the **Show Extension Attributes** check box to include attributes added on the **UDAs** page.
- InputOutput Extension Group: Use to 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. If you have a large IO count, use InputOutputExtension instead of using Input Extension and Output Extension separately. The behavior of Boolean attributes/UDAs that are an Output extension or InputOutput extension extended can handle Momentary changes such as false-true-false transitions within a scan.
- Input Extension Group: Use to configure an attribute to be a reference source for another object. The extended attribute acquires the update value of the **Source** attribute.
- Output Extension Group: Use to configure an attribute to be writeable to an external object’s attribute. 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 Output extension or InputOutput extension extended can handle Momentary changes such as false-true-false transitions within a scan.
- Alarm Extension Group: Use to create an alarm condition when a Boolean attribute’s value is set to TRUE.

  In the Alarm Message field, 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 Industrial Application Server alarm view and can be used in in InTouch.

  If you specify False and True alarm text in the Boolean Label Extension area, these values appear in the Alarm Active State list. These values can also can be used in InTouch.
- History Extension Group: Use to historize the value of an attribute that does not already have history capabilities. You can historize boolean (Discrete) values, as well as many others. These values are stored in InSQL, the ArchestrA historian.
- Boolean Label Extension: Use to specify the text strings for the False state and the True state. These text strings appear in the Active Alarm State list for you to select. If you are using InTouch, you can see these text strings.

To create and associate an extension with an object

1. On the **Extensions** page of the object’s 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 check box of the kind of extension you want to apply to the selected attribute. The associated parameters for each kind of extension become available.

3. For **InputOutput Extension**, enter a Source attribute by either typing in the reference string or clicking the attribute browser button at right. Use the **Attribute Browser** dialog box to search for the reference string in an object. Then if **Destination** is different from **Source**, click **Output Destination Differs from Input Source**, and enter a **Destination** attribute by either typing in the reference string or clicking the attribute browser button. An X is placed in the **InputOutput** column of the selected attribute.

Caution! If you clear the **Output Destination Differs from Input Source** check box, the content of the **Destination** box automatically becomes "---". In the run-time environment, "---" is interpreted as the same reference as the **Source** value entered during configuration time. In the run-time, you can change the **Source** reference. Therefore, during configuration, do not lock the **Destination** parameter if you clear the **Output Destination Differs from Input Source** check box.

4. For **Input Extension**, enter a **Source** attribute by either typing in the reference string or clicking the attribute browser button at right. Use the **Attribute Browser** dialog box to search for the desired reference string in an object. An X is placed in the **Input** column of the selected attribute.
5. For **Output Extension**, enter a **Destination** attribute by either typing in the reference string or clicking the attribute browser button at right. Use the **Attribute Browser** dialog box to search for the desired reference string in an object. 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 is placed in the **Output** column of the selected attribute.
6. 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**. Type a **Priority** level for the alarm (default is 500). Also, select between using the Object Description for Alarm Message or typing in another alarm message in the **Message** box. An X is placed in the **Alarm** column of the selected attribute.
7. 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 is placed in the **History** column of the selected attribute.

- 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.

8. 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.
9. Set the security classification for the attribute if available. See "Security Icons" for more information.
10. Save and close the object editor to include the new attribute extensions in the configured object.

InputOutput Extension

The purpose of the InputOutput extension is to allow an attribute in an object (template or instance) to be configured so that its value is both read from and written to an external reference. The primary role of the InputOutput extension is to monitor the value/quality of an input and to send outputs upon change.

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

Multiple InputOutput extensions can be added to an AutomationObject. You cannot, though, add an InputOutput extension to an attribute that already has either an input or output extension.

Note Lockable attributes can be extended with an InputOutput extension, but they only function correctly at runtime if the extended attribute is unlocked.

When the object is OnScan, the value and quality of the InputOutput-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 due to communication errors or due to datatype conversion failure.

A key behavior while the extended object is OnScan is the following: 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 and plays an important role in driving outputs.

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

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 first type of script uses the current value coming from the input source location and performs logic or calculations on it. This input-side script should refer directly to the extended attribute in its expressions. For example, if the extended attribute is “me.uda1”, the script should refer directly to “me.uda1” for data change conditions and for expressions within the script.

The second type of script can be written to manipulate an output or validate a new requested output value. This output-side script must be written to refer to the “WriteValue” attribute that extends the extended attribute: “me.uda1.WriteValue”. So, to validate a new requested value to the uda1, for example, a data change condition expression should be written on “me.uda1.WriteValue”. In addition, if the script wants to do clamping or validation, it can manipulate the “me.uda1.WriteValue” directly to clamp the output value. For example:

```
If (me.uda1.WriteValue > 100.0 ) then
  Me.uda1.WriteValue = 100.0;
Endif;
```

The data change expression for such a script should be “me.uda1.WriteValue” since this is the value changes when a new value is about to be written to the field. The script has an opportunity to intercept this value just prior to 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 Output Functionality.

Input Extension

Only writeable attributes of an objects can be extended with an input extension. Arrays are not supported.

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 that are extended by an input extension are not protected by their security classification. The only enforced security controls whether an IDE user is allowed to edit the object (to extend it).

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.

Output Extension

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

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 configuration error and type mismatch state.

An output extension can be added to a template or 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.

For more information, see Output Functionality.

Note The quality property of an attribute that has been enhanced with an Output Extension has the following characteristics: Only when quality is Good or Uncertain can a value be output. 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 has not been 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.

Alarm Extension

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

Attribute arrays cannot be extended.

History Extension

Any attribute that exists at runtime and is not already historizable can be configured with the history extension

Writeable and ReadOnly attributes of the following data types can be extended 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) and 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 0 is valid and is the default and means that some change is required prior to storing the value. 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.

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

Output Functionality

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.

The following occurs 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 in the same way 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 had not been 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.

The exception to the above-described functionality is for Boolean data types used in User sets (sets from InTouch or FactorySuite Gateway). This functionality accounts for an unknown user input rate (for example, repeated button pushes) with a consistent object scan rate for outputs, and therefore creates reproducible results. In this case, a combination of folding as described above plus maintenance of a queue of one element deep to better meet 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.

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

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

Important! 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, increasing the potential for additional network traffic. The behavior described in this note does not apply to an InputOutput extension.

QuickScript .NET Scripting Language

This section describes typical script construction and the set of script functions that are included by default in the Industrial Application Server development environment. Script functions are grouped in libraries, including common ones used in typical script development.

QuickScript .NET syntax supports the following .NET constructions:

- Declaring .NET type variables.
- Calling .NET type public constructors (with and without parameters) .
- Calling .NET type public instance methods (with and without parameters).
- Calling .NET type public static methods (with and without parameters).
- Calling .NET type public indexers (with one or more parameters).
- Using .NET enumerations.

The following QuickScript .NET keywords are reserved for use by the scripting language syntax:

Quickscript Keywords		
and	exit	null
as	false	object
attribute	float	or
boolean	for	real

Quickscript Keywords		
dim	if	shl
discrete	in	shr
double	indirect	step
each	integer	string
elapsedtime	message	then
else	mod	time
elseif	new	to
endif	next	true
endwhile	not	while

The New keyword

The following is an example of using the New keyword.

```
'Add to Declarations of script
dim zipcodes as System.Data.PropertyCollection;
Dim iZipCode;
----------------------------------
'Add to Startup method of script
zipcodes = new System.Data.PropertyCollection;
zipcodes.Add("Irvine", 92618);
zipcodes.Add("Mission Viejo", 92692);
------------------------------
'Add to Execute method of script
iZipCode = zipcodes.Item("Irvine");
IF iZipCode == 92692 Then
   LogMessage("FAIL");
Elseif iZipCode == 92618 Then
   LogMessage("Pass");
Endif;
```

Indirect Keyword

The Indirect keyword supports dynamically binding a variable to an arbitrary reference string for get/set usage.

The syntax for this keyword, including the use of the method, BindTo(s), is show in the example below:

```
dim x as indirect;
...
x.BindTo(s); ' where s is any expression that returns a
   string
x = 1234;
if WriteStatus(x) == MxStatusOk then
```

```
 ' ... do something
endif;
```

Important! The Attribute() keyword does not accept values as a parameter. To use dynamic references to an attribute value, use Indirect instead.

The following words are also reserved for future use in QuickScript .NET :

Future Quickscript Reserved Words			
abstract	error	me	switch
base	event	namespace	this
begin	explicit	nothing	throw
break	extern	on	trigger
call	finally	out	try
case	function	private	typeof
catch	goto	protected	until
class	implicit	public	using
continue	import	raiseevent	virtual
default	imports	readonly	void
delegate	include	ref	when
do	inherits	return	with
end	interface	select	
endsub	internal	sizeof	
endswitch	is	static	
enum	like	sub	

Note QuickScript .NET keywords and variable name identifiers are case independent. Case is preserved in editor sessions.

Common Scripting Environment

This section describes common styles, syntax, commands and behaviors of the scripting environment.

Both single and mult-line comments are supported. Single-line comments start with a “’” in the source code line but require no ending “’” in the line. Multi-line comments start with a “{“and end with a “}” and can span multiple lines.

White space rules apply for space and indention. Indent with the `TAB` key. Individual statements are distinguished by a semicolon marking the end of the statement.

Script Editing Styles and Syntax

Two styles of script-writing are supported: simple and complex. Simple scripts allow assignments, comparisons, simple math functions, and similar actions. Complex scripts provide the ability to perform logical operations in the form of IF-THEN-ELSE type control structures. Simple scripts are described below. See QuickScript .NET Control Structures for more information about complex control structures.

Required Syntax for Expressions and Scripts

The syntax used in scripts is similar to the algebraic syntax of a calculator. Most expressions are assignment statements in the following form:

```
a = (b - c) / (2 + x) * xyz;
```

This statement places the value of the expression to the right of the equal sign (=) in the variable location named “a.” A single entity must appear to the left of the assignment operator =. The operands in an expression can be constants or variables. Every expression must end with a semicolon (;).

Entities can be concatenated by using the plus (+) operator. For example, if a data change script such as the one below is created, each time the value of “Number” changes, the indirect entity “Setpoint” changes accordingly:

```
Number=1;
Setpoint = “Setpoint” + Text(Number, “#”);
```

Where: The result is “`Setpoint1`.”

Simple Scripts

Simple scripts provide the ability to implement logic such as assignments, math and functions. An example of this type of scripting is:

```
React_temp = 150;
ResultTag = (Sample1 + Sample2)/2;
{this is a comment}
```

QuickScript .NET Operator(s)

QuickScript .NET operator(s) that require a single operand:

Operators	Short Description
~	Complement
-	Negation
NOT	Logical NOT

QuickScript .NET operator(s) that require two operands:

Operators	**Short Description**
*	Multiplication
/	Division
+	Addition and concatenation
-	Subtraction
=	Assignment
MOD	Modulo
SHL	Left shift
SHR	Right shift
&	Bitwise AND
^	Exclusive OR
|	Inclusive OR
**	Power
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equivalency (is equivalent to); not supported for entire array compares. Arrays must be compared one element at a time using ==.
<>	Not equal to
AND	Logical AND
OR	Logical OR

Precedence of operators:

Precedence	**Operator**
1 (highest)	( )
2	- (negation), NOT, ~
3	**
4	*, /, MOD
5	+, - (subtraction)
6	SHL, SHR
7	<, >, <=, >=
8	==, <>
9	&
10	^
11	|
12	=

Precedence	Operator
13	AND
14 (lowest)	OR

Arguments for the previously listed operators can be numbers or attribute values. Putting parentheses around the arguments is optional, and the operator names are not case-sensitive.

Parentheses ( )

Use parentheses to ensure the correct order of evaluation for the operator(s). They can also make a complex expression easier to read. Operator(s) in parentheses are evaluated first (preempting the other rules of precedence that apply in the absence of parentheses). If the precedence is in question or needs to be overridden, use parentheses.

In the example below parentheses are used to force B and C to be added together before multiplying by D:

```
( B + C ) * D;
```

If you are using QuickScript .NET indexers, you can use either () or [].

Negation (-)

Negation is an operator that acts on a single component. It converts a positive integer or real number into a negative number.

Complement (~)

This operator yields the one's complement of a 32-bit integer. It converts each zero-bit to a one-bit and each one-bit to a zero-bit. The one's complement operator is an operator that acts on a single component, and it accepts an integer operand.

Power (**)

This binary operator returns the result of a number (the base) raised to the power of a second number (the power). The base and the power can be any real or integer numbers, subject to the following restrictions:

A zero base and a negative power are invalid.

Example: "0 ** - 2" and "0 ** -2.5"

A negative base and a fractional power are invalid.

Example: "-2 ** 2.5" and "-2 ** -2.5"

Invalid operands yield a zero result. Moreover, the result of the operation should not be so large or so small that it cannot be represented as a real number. Example:

```
1 ** 1 = 1.0

3 ** 2 = 9.0

10 ** 5 = 100,000.0
```

Multiplication (*), Division (/), Addition (+), Subtraction (-)

These binary operators perform basic mathematical operations. The plus (+) is also used to concatenate Memory or Message types. For example, if a data change script such as the one below is created, each time the value of “Number” changes, “Setpoint” changes accordingly:

```
Number=1;
Setpoint.Name = “Setpoint” + Text(Number, “#” );
```

Where: The result would be “`Setpoint1`.”

Modulo (MOD)

MOD is a binary operator that divides an integer quantity to its left by an integer quantity to its right. The remainder of this division is the result of the MOD operation. Example:

```
97 MOD 8 yields 1
63 MOD 5 yields 3
```

Shift Left (SHL), Shift Right (SHR)

SHL and SHR are binary operators that use only integer operands. The binary content of the 32-bit word referenced by the quantity to the left of the operator is shifted (right or left) by the number of bit positions specified in the quantity to the right of the operator. Bits shifted out of the word are lost. Bit positions vacated by the shift are zero-filled. The shift is an unsigned shift.

Bitwise AND (&)

This is a bitwise binary operator which compares 32-bit integer words with each other, bit for bit. A common use of this operator is to mask a set of bits. The operation in this example “masks out” (sets to zero) the upper 24 bits of the 32-bit word. For example:

```
result = name & 0xff;
```

Exclusive OR (^) and Inclusive OR (|)

The ORs are bitwise logical operators which compare 32-bit integer words to each other, bit for bit. The Exclusive OR looks for opposite bit status's in corresponding locations. If the corresponding bits are the same, a zero is the result. If the corresponding bits differ, a one is the result. Example:

```
0 ^ 0 yields 0
0 ^ 1 yields 1
1 ^ 0 yields 1
1 ^ 1 yields 0
```

The Inclusive OR examines the corresponding bits for a one condition. If either bit is a one, the result is a one. Only when both corresponding bits are zeros is the result a zero. For example:

```
0 | 0 yields 0
0 | 1 yields 1
1 | 0 yields 1
1 | 1 yields 1
```

Assignment (=)

Assignment is a binary operator which accepts integer or real operands. Each statement can contain only one assignment operator. Only one name can be on the left side of the assignment operator. Read the equal sign (=) of the assignment operator as “is assigned to” or “is set to.”

Note Do not confuse the equal sign with the equivalency sign (==) used in IF-THEN-ELSE clauses and relational contacts.

Comparisons (<, >, <=, >=, ==, <>)

These operators are used in IF-THEN-ELSE statements to execute various instructions based on the state of an expression.

AND, OR, and NOT

These operators only work on discrete attributes. If these operators are used on integers or reals, they are converted as follows:

Real to Discrete: If real is 0.0, discrete is 0, otherwise discrete is 1.

Integer to Discrete: If integer is 0, discrete is 0, otherwise discrete is 1.

Thus, if the statement is: “`Disc1 = Real1 AND Real2;`” and `Real1` is 23.7 and `Real2` is 0.0, `Disc1` has 0 assigned to it, since `Real1` is converted to 1 and `Real2` is converted to 0.

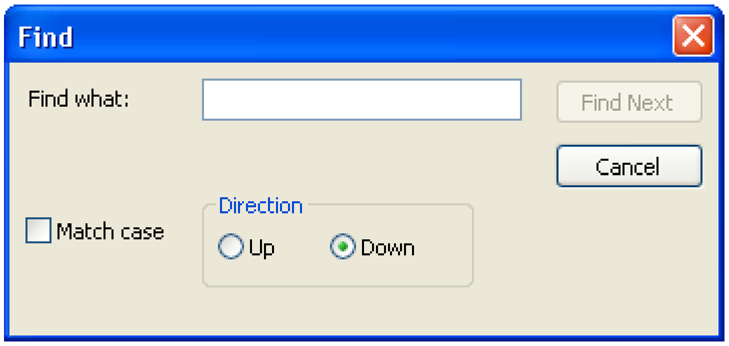
Important! When assigning the floating-point result of a mathematical operation to an integer, Industrial Application Server rounds the value to the nearest integer instead of truncating it. This means that an operation like `IntAttr = 32/60` results in `IntAttr` having a value of 1, not 0. If truncation is desired instead, use the `Trunc()` function.

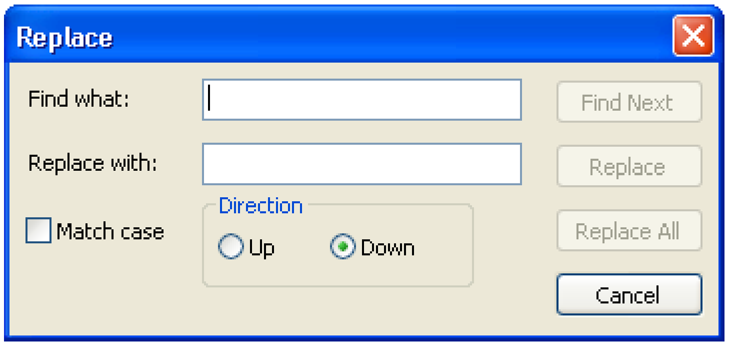
Script Creation Box Shortcut Menu

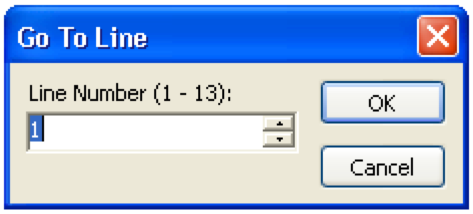
Use the shortcut menu to manipulate text in the Script Creation Box. Right-click anywhere in this script-writing area to see the following commands:

Command	Description
Cut	Removes selected text.
Copy	Writes the selected text to the Windows Clipboard.
Paste	Writes the contents of the Windows Clipboard to the cursor position.
Delete	Permanently removes selected text. The text is not available for pasting from Windows Clipboard.
Select All	Selects all text in the Script Creation Box.
Zoom In	Increases the size of the text in the Script Creation Box.
Zoom Out	Decreases the size of the text in the Script Creation Box.
Find	Locates specific text. This command opens the **Find** dialog box shown below.
Replace	Locates specific text and replace it with new text. This command opens the **Replace** dialog box shown below.

Command	Description
Find Next	Locates the next instance of the same text sought with the **Find** command.
Go To	Moves the cursor to a specific line of code in your script. This command opens the **Go To Line** dialog box shown below.







If you want to print a script, use the **Select All**, **Copy** and **Paste** commands on the shortcut menu to copy to Microsoft Notepad, and then print from that utility.

Script Functions

This section describes the set of script functions included by default in the Industrial Application Server development environment. Functions are listed alphabetically with

- a functional description

- the category in which each is shown in the IDE user interface
- its proper syntax with descriptions of parameters
- examples

An additional category of script functions shown in the IDE user interface are the Types functions, which are not described in this documentation. These functions include .NET functions provided by the Microsoft .NET Framework and any .NET functions you or a third-party developed with Microsoft Visual Studio .NET.

For descriptions of each function provided by the Microsoft .NET Framework, refer to the Microsoft Developers Network website (http://msdn.microsoft.com/). For information about other functions in this category, refer to third-party documentation.

Abs()

Returns the absolute value (unsigned equivalent) of a specified number.

Category

Math

Syntax

```
Result = Abs( Number );
```

Parameters

Number
  Any number or numeric attribute.

Example(s)

```
Abs(14); ' returns 14
```

```
Abs(-7.5); ' returns 7.5
```

ActivateApp()

Activates another currently running Windows application.

Category

Miscellaneous

Syntax

```
ActivateApp( TaskName );
```

Parameter

TaskName
  The task this function activates.

Remarks

TaskName is the exact text string, including spaces, that appears on the Task Bar or in the Task Manager (accessed by right-clicking the Taskbar in Windows and then clicking Task Manager or Ctrl+Alt+Delete).

Example(s)

```
ActivateApp("Calculator");
```

ArcCos()

Returns an angle between 0 and 180 degrees whose cosine is equal to the number specified.

Category

Math

Syntax

```
Result = ArcCos( Number );
```

Parameter

Number
  Any number or numeric attribute with a value between -1 and 1 (inclusive).

Example(s)

```
ArcCos(1); ' returns 0
```

```
ArcCos(-1); ' returns 180
```

See Also

Cos(), Sin(), Tan(), ArcSin(), ArcTan()

ArcSin()

Returns an angle between -90 and 90 degrees whose sine is equal to the number specified.

Category

Math

Syntax

```
Result = ArcSin( Number );
```

Parameter

Number
  Any number or numeric attribute with a value between -1 and 1 (inclusive).

Example(s)

```
ArcSin(1); ' returns 90
```

```
ArcSin(-1); ' returns -90
```

See Also

Cos(), Sin(), Tan(), ArcCos(), ArcTan()

ArcTan()

Returns an angle between -90 and 90 degrees whose tangent is equal to the number specified.

Category

Math

Syntax

```
Result = ArcTan( Number );
```

Parameter

Number
Any number or numeric attribute.

Example(s)

```
ArcTan(1); ' returns 45
```

```
ArcTan(0); ' returns 0
```

See Also

Cos(), Sin(), Tan(), ArcCos(), ArcSin()

Cos()

Returns the cosine of an angle given in degrees.

Category

Math

Syntax

```
Result = Cos( Number );
```

Parameter

Number
Any number or numeric attribute.

Example(s)

```
Cos(90); ' returns 0
```

```
Cos(0); ' returns 1
```

This example shows how to use the function in a math equation:

```
Wave = 50 * Cos(6 * Now().Second);
```

See Also

Sin(), Tan(), ArcCos(), ArcSin(), ArcTan()

CreateObject()

Creates an ActiveX (COM) object.

Category

System

Syntax

```
ObjectResult = CreateObject( ProgID );
```

Parameter

ProgID
The program ID (as a string) of the object to be created.

Example

```
CreateObject(“ADODB.Connection”);
```

DText()

Returns one of two possible strings, depending on the value of the *Discrete* parameter.

Category

String

Syntax

```
StringResult = DText( Discrete, OnMsg, OffMsg );
```

Parameter(s)

Discrete
A Boolean value or Boolean attribute.

OnMsg
The message that is shown when the value of *Discrete* equals **true**.

OffMsg
The message shown when *Discrete* equals **false**.

Example(s)

```
StringResult = DText(me.temp > 150, "Too hot", "Just
    right");
```

Exp()

Returns the result of *e* raised to a power.

Category

Math

Syntax

```
Result = Exp( Number );
```

Parameter

Number
  Any number or numeric attribute.

Example(s)

```
Exp(1); ' returns 2.718...
```

Int()

Returns the next integer less than or equal to a specified number.

Category

Math

Syntax

```
IntegerResult = Int( Number );
```

Parameter

Number
  Any number or numeric attribute.

Remarks

When handling negative real (float) numbers, this function returns the integer farthest from zero.

Example(s)

```
Int(4.7); ' returns 4
Int(-4.7); ' returns -5
```

IsBad()

Returns a Boolean value indicating if the quality of the specified attribute is Bad.

Category

Miscellaneous

Syntax

```
BooleanResult = IsBad( Attribute1, Attribute2, … );
```

Parameter

Attribute1, Attribute2, etc.
: Name of the attribute(s) for which you want to determine Bad quality. You can include a variable-length list of attributes.

Return Value

If any of the specified attributes has Bad quality, then **true** is returned. Otherwise, **false** is returned.

Example(s)

```
IsBad(TIC101.PV);
```

```
IsBad(TIC101.PV, PIC102.PV);
```

See Also

IsGood(), IsInitializing(), IsUncertain(), IsUsable()

IsGood()

Returns a Boolean value indicating if the quality of the specified attribute is Good.

Category

Miscellaneous

Syntax

```
BooleanResult = IsGood( Attribute1, Attribute2, … );
```

Parameter

Attribute1, Attribute2, etc.
: Name of the attribute(s) for which you want to determine Good quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have Good quality, then **true** is returned. Otherwise, **false** is returned.

Example(s)

```
IsGood(TIC101.PV);
```

```
IsGood(TIC101.PV, PIC102.PV);
```

See Also

IsBad(), IsInitializing(), IsUncertain(), IsUsable()

IsInitializing()

Returns a Boolean value indicating if the quality of the specified attribute is Initializing.

Category

Miscellaneous

Syntax

```
BooleanResult = IsInitializing( Attribute1, Attribute2, …
    );
```

Parameter

Attribute1, Attribute2, etc.
Name of the attribute(s) for which you want to determine Initializing quality. You can include a variable-length list of attributes.

Return Value

If any of the specified attributes has Initializing quality, then **true** is returned. Otherwise, **false** is returned.

Example(s)

```
IsInitializing(TIC101.PV);
```

```
IsInitializing(TIC101.PV, PIC102.PV);
```

See Also

IsBad(), IsGood(), IsUncertain(), IsUsable()

IsUncertain()

Returns a Boolean value indicating if the quality of the specified attribute is Uncertain.

Category

Miscellaneous

Syntax

```
BooleanResult = IsUncertain( Attribute1, Attribute2, … );
```

Parameter

Attribute1, Attribute2, etc.
Name of the attribute(s) for which you want to determine Uncertain quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have Uncertain quality, then **true** is returned. Otherwise, **false** is returned.

Example(s)

```
IsUncertain(TIC101.PV);
```

```
IsUncertain(TIC101.PV, PIC102.PV);
```

See Also

IsBad(), IsGood(), IsInitializing(), IsUsable()

IsUsable()

Returns a Boolean value indicating if the specified attribute is usable for calculations.

Category

Miscellaneous

Syntax

```
BooleanResult = IsUsable( Attribute1, Attribute2, … );
```

Parameter

Attribute1, Attribute2, etc.
Name of the attribute(s) for which you want to determine unusable quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have either Good or Uncertain quality, then **true** is returned. Otherwise, **false** is returned.

Remarks

To qualify as usable, the attribute must have Good or Uncertain quality. In addition, each float or double attribute cannot be a NaN (not a number).

Example(s)

```
IsUsable(TIC101.PV);
```

```
IsUsable(TIC101.PV, PIC102.PV);
```

See Also

IsBad(), IsGood(), IsInitializing(), IsUncertain()

Log()

Returns the natural log (base e) of a number.

Category

Math

Syntax

```
RealResult = Log( Number );
```

Parameter

Number
Any number or numeric attribute.

Remarks

Natural log of 0 is undefined.

Example(s)

```
Log(100); ' returns 4.605...
```

```
Log(1); ' returns 0
```

See Also

LogN(), Log10()

LogN()

Returns the values of the logarithm of x to base n.

Category

Math

Syntax

```
Result = LogN( Number, Base );
```

Parameter

Number
Any number or numeric attribute.

Base
Integer to set log base. You could also specify an integer attribute.

Remarks

Base 1 is undefined.

Example(s)

```
LogN(8, 3); ' returns 1.89279
```

```
LogN(3, 7); ' returns 0.564
```

See Also

Log(), Log10()

Log10()

Returns the base 10 log of a number.

Category

Math

Syntax

```
Result = Log10( Number );
```

Parameter

Number
Any number or numeric attribute.

Example(s)

```
Log10(100); ' returns 2
```

See Also

Log(), LogN()

LogMessage()

Writes a user-defined message to the Logger.

Category

Miscellaneous

Syntax

```
LogMessage( msg );
```

Parameter

msg
The message to write to the Logger. Actual string or a string attribute.

Remarks

This is a very powerful function for troubleshooting scripting. By strategically placing **LogMessage()** functions in your scripts, you can determine the order of QuickScript execution, performance of scripts, and identify the value of attributes both before they are changed and after they have been affected by the QuickScript.

Each message posted to the Logger is stamped with the exact date and time. The message always begins with the component "Tagname.ScriptName" so you can tell what object and what script within the object logged the message.

Example(s)

```
LogMessage("Report Script is Running");
```

The above statement would write the following to the Logger:

```
94/01/14 15:21:14 ScriptRuntime Message:Report Script is
   Running.
```

```
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

```
MyTag+MyTag + 10;
```

```
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

Now()

Returns the current time.

Category

System

Syntax

```
TimeValue = Now();
```

Remarks

The return value can be formatted using .NET functions or the **StringFromTime()** function.

Pi()

Returns the value of Pi.

Category

Math

Syntax

```
RealResult = Pi();
```

Example(s)

```
Pi(); ' returns 3.1415926
```

Round()

Rounds a real number to a specified precision and returns the result.

Category

Math

Syntax

```
RealResult = Round( Number, Precision );
```

Parameter

Number
Any number or numeric attribute.

Precision
Sets the precision to which the number is rounded. This value can be any number or a numeric attribute.

Example(s)

```
Round(4.3, 1); ' returns 4
Round(4.3, .01); ' returns 4.30
Round(4.5, 1); ' returns 5
Round(-4.5, 1); ' returns -4
Round(106, 5); ' returns 105
Round(43.7, .5); ' returns 43.5
```

See Also

Trunc()

SendKeys()

Sends keys to an application. To the receiving application, the keys appear to be entered from the keyboard. Use this capability to enter data into the application or to give commands to the application. Most keyboard keys can be used in a **SendKeys()** statement. Each key is represented by one or more characters, such as A for the letter A or {ENTER} for the Enter key.

Category

Miscellaneous

Syntax

```
SendKeys( KeySequence );
```

Parameter

KeySequence
Any key sequence or a string attribute.

Remarks

To specify more than one key, concatenate the codes for each character. For example, to specify the dollar sign ($) key followed by a (b), enter $b. The following lists the valid send keys for unique keyboard keys:

Key	Code	Key	Code
BACKSPACE	{BACKSPACE}or {BS}	HOME	{HOME}
BREAK	{BREAK}	INSERT	{INSERT}
CAPSLOCK	{CAPSLOCK}	LEFT	{LEFT}
DELETE	{DELETE} or {DEL}	NUMLOCK	{NUMLOCK}
DOWN	{DOWN}	PAGE DOWN	{PGDN}
END	{END}	PAGE UP	{PGUP}
ENTER	{ENTER} or ~ (tilde)	PRTSC	{PRTSC}
ESCAPE	{ESCAPE} or {ESC}	RIGHT	{RIGHT}
F1	{F1}*	TAB	{TAB}
		UP	{UP}

* All functions keys are entered the same.

Special keys (SHIFT, CTRL and ALT) have their own key codes:

Key	Code
SHIFT	+ (plus)
CTRL	^ (caret)
ALT	% (percent)

Enhancements to the Microsoft Hardware Abstraction Layer in Windows can keep this function from operating on some computers.

Example(s)

If two special keys are to be used together, a second set of parentheses is required. The following statement holds down the CTRL key while pressing the ALT key followed by p:

```
SendKeys ("^(%(p))");
```

Commands can be preceded by the **ActivateApp()** command to direct the keystrokes to the proper application.

The following statement gives the computer focus to Excel and sends the key combination CTRL+P, which can run a previously defined print macro assigned to run with the key combination CTRL+P:

```
ActivateApp("Microsoft Excel");

SendKeys("^(p)");
```

SetBad()

Sets the quality of an attribute to Bad.

Category

Miscellaneous

Syntax

```
SetBad( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Bad.

Remarks

The specified attribute must be within the object to which the script is attached.

Example(s)

```
SetBad(me.PV);
```

See Also

SetGood(), SetInitializing(), SetUncertain()

SetGood()

Sets the quality of an attribute to Good.

Category

Miscellaneous

Syntax

```
SetGood( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Good.

Remarks

The specified attribute must be within the object to which the script is attached.

Example(s)

```
SetGood(me.PV);
```

See Also

SetBad(), SetInitializing(), SetUncertain()

SetInitializing()

Sets the quality of an attribute to Initializing.

Category

Miscellaneous

Syntax

```
SetInitializing( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Initializing.

Remarks

The specified attribute must be within the object to which the script is attached.

Example(s)

```
SetInitializing(me.PV);
```

See Also

SetBad(), SetGood(), SetUncertain()

SetUncertain()

Sets the quality of an attribute to Uncertain.

Category

Miscellaneous

Syntax

```
SetUncertain( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Uncertain.

Remarks

The specified attribute must be within the object to which the script is attached.

Example(s)

```
SetUncertain(me.PV);
```

See Also

SetBad(), SetGood(), SetInitializing()

Sgn()

Determines the sign of a value (whether it is positive, zero, or negative) and returns the result.

Category

Math

Syntax

```
IntegerResult = Sgn( Number );
```

Parameter

Number
Any number or numeric attribute.

Return Value

If the input number is positive, the result is 1. Negative numbers return a -1, and 0 returns a 0.

Example(s)

```
Sgn(425); ' returns 1
Sgn(0); ' returns 0
Sgn(-37.3); ' returns -1
```

Sin()

Returns the sine of an angle given in degrees.

Category

Math

Syntax

```
Result = Sin( Number );
```

Parameter

Number
Angle in degrees. Any number or numeric attribute.

Example(s)

```
Sin(90); ' returns 1
```

```
Sin(0); ' returns 0
```

This example shows how to use the function in a math expression:

```
wave = 100 * Sin (6 * Now().Second);
```

See Also

Cos(), Tan(), ArcCos(), ArcSin(), ArcTan()

Sqrt()

Returns the square root of a number.

Category

Math

Syntax

```
RealResult = Sqrt( Number );
```

Parameter

Number
Any number or numeric attribute.

Example(s)

This example takes the value of me.PV and returns the square root as the value of x:

```
x=Sqrt(me.PV);
```

StringASCII()

Returns the ASCII value of the first character in a specified string.

Category

String

Syntax

```
IntegerResult = StringASCII( Char );
```

Parameter

Char
Alphanumeric character or string or string attribute.

Remarks

When this function is processed, only the single character is tested or affected. If the string provided to StringASCII contains more than one character, only the first character of the string is tested.

Example(s)

```
StringASCII("A"); ' returns 65
```

```
StringASCII("A Mixer is Running"); ' returns 65
```

```
StringASCII("a mixer is running"); ' returns 97
```

See Also

StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringChar()

Returns the character corresponding to a specified ASCII code.

Category

String

Syntax

```
StringResult = StringChar( ASCII );
```

Parameter

ASCII
ASCII code or an integer attribute.

Remarks

One use of this function is to add ASCII characters not normally represented on the keyboard to a string attribute.

Example(s)

In this example, a [Carriage Return (13)] and [Line Feed (10)] are added to the end of StringAttribute and passed to ControlString. Inserting characters out of the normal 32-126 range of displayable ASCII characters can be very useful for creating control codes for external devices such as printers or modems.

```
ControlString=StringAttribute+StringChar(13)+StringChar(10
    );
```

Another common use of this function is for SQL commands. The where expression sometimes requires double quotes around string values, so StringChar(34) is used.

See Also

StringASCII(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringFromIntg()

Converts an integer value into its string representation in another base and returns the result.

Category

String

Syntax

```
SringResult = StringFromIntg( Number, numberBase );
```

Parameter

Number
Number to convert. Any number or an integer attribute.

numberBase
Base to use in conversion. Any number or an integer attribute.

Example(s)

```
StringFromIntg(26, 2); ' returns "11010"
StringFromIntg(26, 8); ' returns "32"
StringFromIntg(26, 16); ' returns "1A"
```

See Also

StringASCII(), StringChar(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringFromReal()

Converts a real value into its string representation, either as a floating-point number or in exponential notation, and returns the result.

Category

String

Syntax

```
StringResult = StringFromReal( Number, Precision, Type );
```

Parameter

Number
: Is converted to the *Precision* and *Type* specified. Any number or a float attribute.

Precision
: Specifies how many decimal places is shown. Any number or an integer attribute.

Type
: A string value that determines the display method. Possible values are:

Type	Description
"f"	Display in floating-point notation.
"e"	Display in exponential notation with a lower-case "e."
"E"	Display in exponential notation with an upper-case "E."

Example(s)

```
StringFromReal(263.355, 2,"f"); ' returns "263.36"
StringFromReal(263.355, 2,"e"); ' returns "2.63e2"
StringFromReal(263.55, 3,"E"); ' returns "2.636E2"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringFromTime()

Converts a time value (in seconds since January 1, 1970) into a particular string representation and returns the result.

Category

String

Syntax

```
StringResult = StringFromTime( SecsSince1-1-70, StringType
    );
```

Parameter

SecsSince1-1-70
: Is converted to the *StringType* specified.

StringType
Determines the display method. Possible values are:

Type	Description
1	Shows the date in the same format set from the Windows Control Panel.
2	Shows the time in the same format set from the Windows Control Panel.
3	Shows a 24-character string indicating both the date and time: "Wed Jan 02 02:03:55 1993"
4	Shows the short form for the day of the week: "Wed"
5	Shows the long form for the day of the week: "Wednesday"

Remarks

The time value should be UTC equivalent (number of seconds since January 1, 1970 GMT). The value returned reflects the local time.

Example(s)

```
StringFromTime(86400, 1) ' returns "1/2/70"

StringFromTime(86400, 2) ' returns "12:00:00 AM"

StringFromTime(86400, 3) ' returns "Fri Jan 02 00:00:00
   1970"

StringFromTime(86400, 4) ' returns "Fri"

StringFromTime(86400, 5) ' returns "Friday"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringInString()

Returns the position in a string of text where a specified string first occurs.

Category

String

Syntax

```
IntegerResult = StringInString( Text, SearchFor, StartPos,
   CaseSens );
```

Parameter

Text
The string that is searched. Actual string or a string attribute.

SearchFor
: The string to be searched for. Actual string or a string attribute.

StartPos
: Used to determine where in the text the search begis. Any number or an integer attribute.

CaseSens
: Determines whether the search is case-sensitive. 0 = Not case-sensitive; 1 = Case-sensitive. Any number or an integer attribute.

Remarks

If multiple occurrences of *SearchFor*are found, the location of the first one is returned.

Example(s)

```
StringInString("The mixer is running", "mix", 1, 0) '
    returns 5
```

```
StringInString("Today is Thursday", "day", 1, 0) ' returns
    3
```

```
StringInString("Today is Thursday", "day", 10, 0) ' returns
    15
```

```
StringInString("Today is Veteran's Day", "Day", 1, 1) '
    returns 20
```

```
StringInString("Today is Veteran's Day", "Night", 1, 1) '
    returns 0
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringLeft()

Returns the specified number of characters in a string value, starting with the leftmost character of text.

Category

String

Syntax

```
StringResult = StringLeft( Text, Chars );
```

Parameter

Text
: Actual string or a string attribute.

Chars
: Number of characters to return or an integer attribute.

Remarks

If *Chars* is set to 0, the entire string is returned.

Example(s)

```
StringLeft("The Control Pump is On", 3) ' returns "The"

StringLeft("Pump 01 is On", 4) ' returns "Pump"

StringLeft("Pump 01 is On", 96) ' returns "Pump 01 is On"

StringLeft("The Control Pump is On", 0) ' returns "The
   Control Pump is On"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringLen()

Returns the length, in characters, of text.

Category

String

Syntax

```
IntegerResult = StringLen( Text );
```

Parameter

Text
   Actual string or a string attribute.

Remarks

All the characters in the string attribute are counted, including those not normally shown on the screen.

Example(s)

```
StringLen("Twelve percent") ' returns 14

StringLen("12%") ' returns 3

StringLen("The end." + StringChar(13)) ' returns 9
```

The carriage return character is ASCII 13.

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringLower()

Converts all of the upper-case characters in text string to lower case and returns the result.

Category

String

Syntax

```
StringResult = StringLower( Text );
```

Parameter

Text
String to be converted to lower case. Actual string or a string attribute.

Remarks

Lower-case characters, symbols, numbers, and other special characters is not affected.

Example(s)

```
StringLower("TURBINE") ' returns "turbine"
```

```
StringLower("22.2 Is The Value") ' returns "22.2 is the
   value"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringMid()

Extracts a specific number of characters from a starting point within a section of text and returns the result.

Category

String

Syntax

```
StringResult = StringMid( Text, StartChar, Chars );
```

Parameter

Text
Actual string or a string attribute

StartChar
The position of the first character to extract. Any number or an integer attribute.

Chars
: The total number of characters to return. Any number or an integer attribute.

Remarks

This function is slightly different than the **StringLeft()** function and **StringRight()** function in that it allows you to specify both the start and end of the string that is to be extracted.

Example(s)

```
StringMid("The Furnace is Overheating",5,7,) ' returns
   "Furnace"
```

```
StringMid("The Furnace is Overheating",13,3) ' returns "is
   "
```

```
StringMid("The Furnace is Overheating",16,50) ' returns
   "Overheating"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringReplace()

Replaces or changes specific parts of a provided string and returns the result.

Category

String

Syntax

```
StringResult = StringReplace( Text, SearchFor,
   ReplaceWith, CaseSens, NumToReplace, MatchWholeWords );
```

Parameter

Text
: The string to change. Actual string or a string attribute.

SearchFor
: The string to search for and replace. Actual string or a string attribute.

ReplaceWith
: The replacement string. Actual string or a string attribute.

CaseSens
: Determines whether the search is case-sensitive. (0=no and 1=yes) Any number or an integer attribute.

NumToReplace
: Determines the number of occurrences to replace. Any number or an integer attribute. To indicate all occurrances, set this value to -1.

MatchWholeWords

Determines whether the function limits its replacement to whole words. (0=no and 1=yes) Any number or an integer attribute.

If *MatchWholeWords* is turned on (set to 1) and the *SearchFor* is set to "and", the "and" in "handle" would not be replaced. If the *MatchWholeWords* is turned off (set to 0), it would be replaced.

Remarks

Use this function to take a string attribute and replace characters, words, or phrases.

The **StringReplace()** function does not recognize special characters, such as @#$%&*(). It reads them as delimiters. For example, if the function **StringReplace()** (abc#,abc#,1234,0,1,1) is processed, there is no replacement. The # sign is read as a delimiter instead of a character.

Example(s)

```
StringReplace("In From Within","In","Out",0,1,0) ' returns
   "Out From Within"  (replaces only the first one)

StringReplace("In From Within","In","Out",0,-1,0) '
   returns "Out From without"  (replaces all occurrences)

StringReplace("In From Within","In","Out",1,-1,0) '
   returns "Out From Within"  (replaces all that match
   case)

StringReplace("In From Within","In","Out",0,-1,1) '
   returns "Out From Within"  (replaces all that are whole
   words)
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringRight()

Returns the specified number of characters starting at the rightmost character of text.

Category

String

Syntax

```
StringResult = StringRight( Text, Chars );
```

Parameter

Text

Actual string or a string attribute.

Chars

The number of characters to return or an integer attribute.

Remarks

If *Chars* is set to 0, the entire string is returned.

Example(s)

```
StringRight("The Pump is On", 2) ' returns "On"
```

```
StringRight("The Pump is On", 5) ' returns "is On"
```

```
StringRight("The Pump is On", 87) ' returns "The Pump is
   On"
```

```
StringRight("The Pump is On", 0) ' returns "The Pump is On"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringSpace()

Generates a string of spaces either within a string attribute or within an expression and returns the result.

Category

String

Syntax

```
StringResult = StringSpace( NumSpaces );
```

Parameter

NumSpaces
Number of spaces to return. Any number or an integer attribute.

Example(s)

All spaces are represented by the "×" character:

```
StringSpace(4) ' returns "××××"
```

```
"Pump" + StringSpace(1) + "Station" ' returns
   "Pump×Station"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringTest()

Tests the first character of text to determine whether it is of a certain type and returns the result.

Category

String

Syntax

```
DiscreteResult = StringTest( Text, TestType );
```

Parameter

Text
String that function acts on. Actual string or a string attribute.

TestType
Determines the type of test. Possible values are:

Type	Description
1	Alphanumeric character ('A'-'Z', 'a-z' and '0-9')
2	Numeric character ('0'- 9')
3	Alphabetic character ('A-Z' and 'a-z')
4	Upper-case character ('A'-'Z')
5	Lower-case character ('a'-'z')
6	Punctuation character (0x21-0x2F)
7	ASCII characters (0x00 - 0x7F)
8	Hexadecimal characters ('A'-'F' or 'a'-'f' or '0'-'9')
9	Printable character (0x20-0x7E)
10	Control character (0x00-0x1F or 0x7F)
11	White Space characters (0x09-0x0D or 0x20)

Remarks

StringTest() function returns **true** to *DiscreteResult* if the first character in *Text* is of the type specified by *TestType*. Otherwise, **false** is returned. If the **StringTest()** function contains more than one character, only the first character of the attribute is tested.

Example(s)

```
StringTest("ACB123",1) ' returns 1
```

```
StringTest("ABC123",5) ' returns 0
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringToIntg()

Converts the numeric value of a string to an integer value and returns the result.

Category

String

Syntax

```
IntegerResult = StringToIntg( Text );
```

Parameter

Text
String that function acts on. Actual string or a string attribute.

Remarks

When this statement is evaluated, the system reads the first character of the string for a numeric value. If the first character is other than a number (blank spaces are ignored), the string's value is equated to zero (0). If the first character is found to be a number, the system continues to read the subsequent characters until a non-numeric value is detected.

Example(s)

```
StringToIntg("ABCD"); ' returns 0

StringToIntg("22.2 is the Value"); ' returns 22  (since
   integers are whole numbers)

StringToIntg("The Value is 22"); ' returns 0
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToReal(), StringTrim(), StringUpper(), Text()

StringToReal()

Converts the numeric value of a string to a real (floating point) value and returns the result.

Category

String

Syntax

```
RealResult = StringToReal( Text );
```

Parameter

Text
String that function acts on. Actual string or a string attribute.

Remarks

When this statement is evaluated, the system reads the first character of the string for a numeric value. If the first character is other than a number (blank spaces are ignored), the string's value is equated to zero (0). If the first character is found to be a number, the system continues to read the subsequent characters until a non-numeric value is encountered.

Example(s)

```
StringToReal("ABCD"); ' returns 0
```

```
StringToReal("22.261 is the value"); ' returns 22.261
```

```
StringToReal("The Value is 2"); ' returns 0
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringTrim(), StringUpper(), Text()

StringTrim()

Removes unwanted spaces from text and returns the result.

Category

String

Syntax

```
StringResult = StringTrim( Text, TrimType );
```

Parameter

Text
String that is trimmed of spaces. Actual string or a string attribute.

TrimType
Determines how the string is trimmed. Possible values are:

Type	Description
1	Remove leading spaces (left of the first non-space character)
2	Remove trailing spaces (right of the last non-space character)
3	Remove all spaces except for single spaces between words

Remarks

The text is searched for white-spaces (ASCII 0x09-0x0D or 0x20) that are to be removed. *TrimType* determines the method used by the function:

Example(s)

All spaces are represented by the "×" character.

```
StringTrim("×××××This×is×a××test×××××", 1) ' returns
    "This×is×a××test×××××"
```

```
StringTrim("×××××This×is×a××test×××××", 2) ' returns
    "×××××This×is×a××test"
```

```
StringTrim("×××××This×is×a××test×××××", 3) ' returns
    "This×is×a×test"
```

The **StringReplace()** function can be used to remove ALL spaces from a specified a string attribute. Simply replace all the space characters with a "null."

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringUpper(), Text()

StringUpper()

Converts all the lower-case characters in text to upper case and returns the result.

Category

String

Syntax

```
StringResult = StringUpper( Text );
```

Parameter

Text
String to be converted to upper case. Actual string or a string attribute.

Remarks

Upper-case characters, symbols, numbers, and other special characters is not affected.

Example(s)

```
StringUpper("abcd"); ' returns "ABCD"
```

```
StringUpper("22.2 is the value"); ' returns "22.2 IS THE
    VALUE"
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), Text()

Tan()

Returns the tangent of an angle given in degrees.

Category

Math

Syntax

```
Result = Tan( Number );
```

Parameter

Number
The angle in degrees. Any number or numeric attribute.

Example(s)

```
Tan(45); ' returns 1
```

```
Tan(0); ' returns 0
```

This example shows how to use the function in a math expression:

```
Wave = 10 + 50 * Tan(6 * Now().Second);
```

See Also

Cos(), Sin(), ArcCos(), ArcSin(), ArcTan()

Text()

Converts a number to text based on a specified format.

Category

String

Syntax

```
StringResult = Text( Number, Format );
```

Parameter

Number
Any number or numeric attribute.

Format
Format to use in conversion. Actual string or a string attribute.

Example(s)

```
Text(66,"#.00"); ' returns 66.00
```

```
Text(22.269,"#.00"); ' returns 22.27
```

```
Text(9.999,"#.00"); ' returns 10.00
```

The following example shows how to use this function within another function:

```
LogMessage("The current value of FreezerRoomTemp is:" +
    Text (FreezerRoomTemp, "#.#"));
```

In the following example, MessageTag is set to "One=1 Two=2".

```
MessageTag = "One + " + Text(1,"#") + StringChar(32) +
    "Two +" + Text(2,"#");
```

See Also

StringFromIntg(), StringToIntg(), StringFromReal(), StringToReal()

Trunc()

Truncates a real (floating point) number by simply eliminating the portion to the right of the decimal point and returns the result.

Category

Math

Syntax

`NumericResult` `= Trunc(` *`Number`* `);`

Parameter

Number
Any number or numeric attribute.

Remarks

This function accomplishes the same result as placing the contents of a float type attribute into an integer type attribute.

Example(s)

```
Trunc(4.3); ' returns 4
```

```
Trunc(-4.3); ' returns -4
```

See Also

Round()

WriteStatus()

Returns the enumerated write status of the last write to the specified attribute.

Category

Miscellaneous

Syntax

`Result` `= WriteStatus(` *`Attribute`* `);`

Parameter

Attribute
The attribute for which you want to return write status.

Return Value

The return statuses are:

- MxStatusOk
- MxStatusPending

- MxStatusWarning
- MxStatusCommunicationError
- MxStatusConfigurationError
- MxStatusOperationalError
- MxStatusSecurityError
- MxStatusSoftwareError
- MxStatusOtherError

Remarks

If the attribute has never been written to, this function returns MxStatusOk. This function always returns MxStatusOk for attributes that do not support a calculated (non-Good) quality.

Example(s)

```
WriteStatus(TIC101.SP);
```

WWControl()

Restores, minimizes, maximizes, or closes an application.

Category

Miscellaneous

Syntax

```
WWControl( AppTitle, ControlType );
```

Parameter

AppTitle
: The name of the application title to be controlled. Actual string or a string attribute.

ControlType
: Determines how the application is controlled. Possible values are shown in the following table. These actions are identical to clicking on their corresponding selections in the application's Control Menu. Actual string or a string attribute.

Type	Description
"Restore"	Activates and shows the application's window.
"Minimize"	Activates a window and shows it as an icon.
"Maximize"	Activates and shows the application's window.
"Close"	Closes an application.

Example(s)

```
WWControl("Calculator","Restore");
```

See Also

ActivateApp()

WWExecute()

Executes a command (using the DDE protocol) to a specified application and topic and returns the status.

Category

WWDDE

Syntax

```
Status = WWExecute( Application, Topic, Command );
```

Parameter

Application
The application to which you want to send an execute command. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Command
The command to send. Actual string or a string attribute.

Return Value

Status is an Integer attribute to which 1, -1, or 0 is written. The **WWExecute()** function returns 1 if the application is running, the topic exists, and the command was sent successfully. It returns 0 when the application is busy, and -1 when there is an error.

Remarks

The *Command* string is sent to the particular application and topic specified.

Important! The following applies to using **WWExecute()** in synchronous scripts:
1. Never loop them(call them over and over).
2. Never call several of themin a row and in the same script.
3. Never use them to call a lengthy task in another DDE application.
All three actions, though, are useable in asynchronous scripts.

Example(s)

The following statement executes a macro in Excel:

```
Macro="Macro1!TestMacro";

Command="[Run(" + StringChar(34) + Macro + StringChar(34)
     + ",0)]";

WWExecute("excel","system",Command);
```

When `WWExecute("excel","system",Command);` is processed, the following is sent to Excel (and *TestMacro* runs):

```
[Run("Macro1!TestMacro")]
```

The following QuickScript executes a macro in Microsoft Access:

```
WWExecute("MSAccess","system","MyMacro");
```

WWPoke()

Pokes a value (using a DDE protocol) to a specified application, topic, and item and returns the status.

Category

WWDDE

Syntax

```
Status = WWPoke( Application, Topic, Item, TextValue );
```

Parameter

Application
The application to which you want to send the Poke command. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Item
The item to poke within the topic. Actual string or a string attribute.

TextValue
The value to poke. If the value you want to send is a number, you can convert it using the **Text()**, **StringFromIntg()**, or **StringFromReal()** functions. Actual string or a string attribute.

Return Value

Status is an Integer attribute to which 1, -1, or 0 is written. The **WWPoke()** function returns 1 if the application is running, the topic and item exist, and the value was sent successfully. It returns 0 when the application is busy, and -1 when there is an error.

Remarks

The value *TextValue* is sent to the particular application, topic, and item specified.

Important! Never do the following when using **WWPoke()** in synchronous scripts:
1. Loop them (call them over and over).
2. Call several of them in a row and in the same script.
3. Use them to call a lengthy task in another DDE application.
All three actions can be done in asynchronous scripts.

Example(s)

The following statement converts a value to text and pokes the result to an Excel spreadsheet cell:

```
String=Text(Value,"0");
WWPoke("excel","[Book1.xls]sheet1","r1c1",String);
```

The behavior for **WWPoke()** from within the application "View" to "View" is undefined and is not supported. The **WWPoke()** command is not guaranteed to succeed in this instance, and the command probably times-out without the desired results.

See Also

Text(), StringFromIntg(), StringFromReal()

WWRequest()

Makes a one-time request for a value (using a DDE protocol) from a particular application, topic, and item and returns the status.

Category

WWDDE

Syntax

```
Status = WWRequest( Application, Topic, Item, Attribute );
```

Parameter

Application
The application from which you want to request data. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Item
The item within the topic. Actual string or a string attribute.

Attribute
A string attribute, enclosed in quotes, that contains the requested value from the application, topic, and item. Actual string or a string attribute.

Return Value

Status is an integer attribute to which 1, -1, or 0 is written. The **WWRequest()** function returns 1 if the application is running, the topic and item exist, and the value was returned successfully. It returns 0 when the application is busy, and -1 when there is an error.

Remarks

The DDE value in the particular application, topic, and item is returned into *Attribute*.

The value is returned as a string into a string attribute. If the value is a number, you can then convert it using the **StringToIntg()** or **StringToReal()** functions.

Important! Never do the following when using **WWRequest()** in synchronous scripts:
1. Loop scripts (call them over and over).
2. Call several of scripts in a row and in the same script.
3. Use scripts to call a lengthy task in another DDE application.
All three actions can be done in asynchronous scripts.

Example(s)

The following statement requests a value from an Excel spreadsheet cell and convert the resulting string into a value:

```
WWRequest("excel","[Book1.xls]sheet1","r1c1",Result);

Value=StringToReal(Result);
```

See Also

StringToIntg(), StringToReal()

QuickScript .NET Variables

Variables must be declared before they can be used in QuickScript .NET scripts. They can be used on both the left and right side of statements and expressions.

Local variables or attributes can be used interchangeably in the same script. Except for those declared in the general section of your script, local variables lose their values at the end of the script function they are used in. Those declared in the general section cannot be accessed by other scripts.

Declaration variables maintain their values thoughout the lifetime of the object that the script is associated with.

Each variable must be declared in the script by a separate `DIM` statement followed by a semicolon. The `DIM` statement syntax is as follows:

```
DIM <variable_name> [ ( <upper_bound> [, <upper_bound >[, <
    upper_bound >]] ) ]  [ AS <data_type> ];
```

where:

`DIM`	Required keyword.
`<variable_name>`	Name that begins with a letter (A-Z or a-z) and whose remaining characters can be any combination of letters (A-Z or a-z), digits (0-9) and unscores (_). The variable name is limited to 255 Unicode characters.
`<upper_bound>`	Reference to the upper bound (a number between 1 and 2,147,483,647, inclusive) of an array dimension. Three dimensions are supported in a `DIM` statement, each being nested in the syntax structure. Once the upper bound is specified, it is fixed after the declaration. A statement similar to Visual Basic's `ReDim` is not supported. **Note**: The lower bound of each array dimension is always 1.

`AS`	Optional keyword for declaring the variable's datatype.

`<data_type>`	Any one of the following 11 datatypes: Boolean, Discrete, Integer, ElapsedTime, Float, Real, Double, String, Message, Time or Object.
	
	If you omit the `AS` clause from the `DIM` statement, the variable, by default, is declared as an Integer datatype. For example:
	
	`DIM LocVar1;`
	
	is equivalent to:
	
	`DIM LocVar1 AS Integer;`

Note In contrast to attribute names, variable names must not contain dots. Variable names and the data type identifiers are not case sensitive. If there is a naming conflict between a declared variable and another named entity in the script (for example, attribute name, alias or name of an object leveraged by the script), the variable name takes precedence over the other named entities. If the variable name is the same as an alias name, a warning message appears when the script is validated to indicate that the alias is ignored.

The syntax for specifying the entire array is "[ ]" for both local array variables and for attribute references. For example, to assign an attribute array to a local array, the syntax is:

```
locarr[] = tag.attr[]
```

`DIM` statements can be located anywhere in the script body, but they must precede the first referencing script statement or expression. If a local variable is referenced before the `DIM` statement, script validation done when you save the object containing the script prompts you to define it.

Caution! The validation mentioned above occurs only when you save the object containing the script. This is not the script syntax validation done when you click the Validate Script button.

`DIM` statements cannot be cascaded. For example, the following examples are invalid:

```
DIM LocVar1 AS Integer, LocVar2 AS Real;

DIM LocVar3, LocVar4, LocVar5, AS Message;
```

To declare multiple variables, you must enter separate `DIM` statements for each variable.

When used on the right side of an equation, declared local variables always cause expressions on the left side to have Good quality. For example, assume:

```
dim x as integer;

dim y as integer;

x = 5;

y = 5;

me.attr = 5;
me.attr = x;
me.attr = x+y;
```

In each case of `me.attr`, quality is Good.

Note When a variable is used in an expression on the right-hand side, its Quality is treated as Good for the purpose of data quality propagation.

Important! It is not possible to pass UDAs as parameters for system objects. To work around this issue, either use a local variable as an intermediary, or explicity convert the UDA to a string using an appropriate function call when calling the system object.

Numbers and Strings

Allowed format for integer constants in decimal format is as follows:

```
IntegerConst =  0 or [sign] <non-zero_digit> <digit>*
```

where:

`sign` ::= + | -

`non-zero_digit` ::= 1-9

`digit` ::= 0-9

For example, an integer constant is a zero or consists of an optional sign followed by one or more digits. Leading zeros are not allowed. Integer constants outside the range –2147483648 to 2147483647 cause an overflow error.

Prepending either 0x or 0X causes a literal integer constant to be interpreted as hexadecimal notation. The +/- sign is supported.

Allowed format for integers in hexadecimal is as follows:

```
IntegerHexConst = [<sign>] <0><x (or X)> <hexdigit>*
```

where:

`sign` ::= + or -

`hexdigit` ::= 0-9, A-F, a-f (only eight hexdigits [32-bits] are allowed)

Allowed format for floats is as follows:

```
FloatConst ::= [<sign>] <digit>* .<digit>+ [<exponent>]
```

or

```
[<sign>] <digit>+ [.<digit>* [<exponent>]]
```

where:

`sign` `::=` + or -

`digit` `::=` 0-9 (can be one or more decimal digits)

`exponent` = e (or E) followed by a sign and then digit(s)

Float constants are applicable as values for variables of type float / real or double. For example, float constants do not take the number of bytes into account. Validation of the script (lexer / parser) detects an overflow when too big a float constant is assigned to either a variable of type float / real or double.

If no digits appear before the period (.), at least one must appear after it. If neither an exponent part nor the period appears, a period is assumed to follow the last digit in the string.

If an attribute reference exists that has a format similar to a float constant with an exponent (such as "5E3"), then use the Attribute qualifier, as follows:

```
Attribute("5E3")
```

Allowed format for strings is as follows:

String must be surrounded by double quotes. They are referred to as quoted strings. The double-double quote indicates a single double-quote in the string. For example, the string:

```
Joe said, "Look at that."
```

can be represented in QuickScript .NET as:

```
"Joe said, ""Look at that""."
```

QuickScript .NET Control Structures

QuickScript .NET provides four primary control structures in the scripting environment:

- IF … THEN … ELSEIF … ELSE … ENDIF
- FOR ... TO ... STEP ... NEXT Loop
- FOR EACH ... IN ... NEXT
- WHILE ... ENDWHILE

These are described next.

IF … THEN … ELSEIF … ELSE … ENDIF

IF-THEN-ELSE-ENDIF conditionally executes various instructions based on the state of an expression. The syntax is as follows:

```
IF <boolean_expression> THEN
```

```
    [statements]
[ { ELSEIF
    [statements] } ]
[ ELSE
    [statements] ]
ENDIF;
```

Where: boolean_expression is an expression that can be evaluated as a Boolean. Dependent on the data type returned by the expression, the expression is evaluated to constitute a True or False state according to the following table:

Data Type	Mapping
Boolean, Discrete	Directly used (no mapping needed).
Integer	Value = 0 evaluated as False Value != 0 evaluated as True.
Float, Real	Value = 0 evaluated as False Value != 0 evaluated as True.
Double	Value = 0 evaluated as False Value != 0 evaluated as True.
String, Message	Cannot be mapped. Using an expression that results in a string type as the boolean_expression results in a script validation error.
Time	Cannot be mapped. Using an expression that results in a time type as the boolean_expression results in a script validation error.
ElapsedTime	Cannot be mapped. Using an expression that results in an elapsed time type as the boolean_expression results in a script validation error.
Object	Using an expression that results in an object type. Validates, but at runtime, the object is converted to a Boolean. If the type cannot be converted to a Boolean, a runtime exception is raised.

The first block of statements is executed if boolean_expression evaluates to True. Optionally a second block of statements can be defined after the keyword ELSE. This block is executed if the boolean_expression evaluates to False. To help decide between multiple alternatives, an optional ELSEIF clause can be used as often as needed. The ELSEIF clause allows for easy mimicking of switch statements offered by some other programming languages.

Example:

```
IF value = 0 Then
          Message = ”Value is zero”;
     ELSEIF value > 0 Then
```

```
          Message = ”Value is positive”;
     ELSEIF value < 0 then
          Message = ”Value is negative”;
     ELSE
          {Default. Should never occur in this example}
ENDIF

The following syntax is also supported:

IF <boolean_expression> THEN
           [statements]
[ { ELSEIF
           [statements] } ]
[    ELSE
           [statements] ]
     ENDIF;
ENDIF;
```

This approach basically nests a brand new IF compound statement within a previous one and requires an additional ENDIF.

See Sample Scripts for ideas about using this type of control structure.

FOR … TO … STEP … NEXT Loop

FOR-NEXT performs a function (or set of functions) within a script several times during a single execution of a script. The general format of the FOR-NEXT loop is as follows:

```
FOR <analog_var> = <start_expression> TO <end_expression>
    [STEP <change_expression>]
          [statements]
          [EXIT FOR;]
          [statements]
NEXT;
```

Where:

- analog_var is a variable of type Integer, Float, Real, or Double.
- start_expression is a valid expression, to initialize analog_var to a value for execution of the loop.
- end_expression is a valid expression. If analog_var is greater than end_expression, execution of the script jumps to the statement immediately following the NEXT statement.

- This holds true if loop is incrementing up, otherwise, if loop is decrementing, loop termination occurs if analog_var is less than end_expression.
- change_expression is an expression, to define the increment or decrement value of analog_var after execution of the NEXT statement. The change_expression can be either positive or negative. If change_expression is positive, start_expression must be less than or equal to end_expression or the statements in the loop does not execute. If change_expression is negative, start_expression must be greater than or equal to end_expression for the body of the loop to be executed. If STEP is not set, then change_expression defaults to 1.

It is possible to exit the loop from within the body of the loop through the EXIT FOR statement.

The FOR loop is executed as follows:

1. analog_var is set equal to start_expression.
2. The system tests to see if analog_var is greater than end_expression. If so, the loop exists. (If change_expression is negative, the system tests to see if analog_var is less than end_expression.)
3. The statements in the body of the loop are executed. The loop can potentially be exited via the EXIT FOR statement.
4. analog_var is incremented by 1 - or by change_expression if it is specified.
5. Steps 2 through 4 are repeated

Note FOR-NEXT loops can be nested. The number of levels of nesting possible depends on memory and resource availability.

See Sample Scripts for ideas about using this type of control structure.

FOR EACH … IN … NEXT

FOR EACH loops can only be used with collections exposed by OLE Automation servers. A FOR-EACH loop performs a function (or set of functions) within a script several times during a single execution of a script. The general format of the FOR-EACH loop is as follows:

```
FOR EACH <object_variable> IN <collection_object >
      [statements]
      [EXIT FOR;]
      [statements]
NEXT;
```

Where:

- object_variable is a variable of type <some COM interface>.
- collection_object is a variable holding a collection object.

As in the case of the FOR … TO loop, it is possible to exit the execution of the loop through the statement EXIT FOR from within the loop.

See Sample Scripts for ideas about using this type of control structure.

While Loop

WHILE loops perform a function (or set of functions) within a script several times during a single execution of a script while a condition is true. The general format of the WHILE loop is as follows:

```
WHILE <boolean_expression>
        [statements]
        [EXIT WHILE;]
        [statements]
ENDWHILE;
```

Where: boolean_expression is an expression that can be evaluated as a Boolean as defined in the description of IF…THEN statements.

It is possible to exit the loop from the body of the loop through the EXIT WHILE statement.

The WHILE loop is executed as follows:

1. The system tests whether the boolean_expression is true. If not, the loop exists.
2. The statements in the body of the loop are executed. The loop can potentially be exited through the EXIT WHILE statement.
3. Steps 1 through 2 are repeated.

Note WHILE loops can be nested. The number of levels of nesting possible depends on memory and resource availability.

See Sample Scripts for ideas about using this type of control structure.

Recognition of Keywords, Symbols

The precedence order in which identifiers (between language keywords and attribute names) are recognized is as follows:

1. Language keywords
2. Attribute names

Sample Scripts

Use the following samples to gain an understanding about the QuickScript .NET scripting language, especially how to perform certain operations.

Caution! These sample scripts are provided to show proper syntax. They can be dependent on other resources or system configuration settings to properly execute.

These sample scripts do the following operations:

Write a Text File to Disk

Read a Text File from Disk

Format a Number Using a .NET Format 'Picture'

Format a Time Using a .NET Format 'Picture'

Create an XML Document and Save it to Disk

Load an XML Document from Disk and Do Look-ups on It

Query a SQL Server Database

Execute a SQL Parameterized INSERT Command

Share a SQL Connection (or Any Other .NET Object)

Create a Look-up Table, Then Do a Look-up on It

Fill a String Array, Then Use It

Fill a Two-Dimensional Integer Array, Then Use It

Get the Directories Under c:\

Use Exchange to Send an Email

Use SMTP to Send an Email

Call a Web Service to Send an Email

Call a Web Service to Get the Temperature for a Given Zip Code

Use Screen-Scraping to Get the Temperature for a City

Access an Excel Spreadsheet Using CreateObject

Access an Excel Spreadsheet Using an Imported Type Library

Access an Office XP Excel Spreadsheet Using an Imported Type Library

Use DDE to Access an Excel Spreadsheet

Read a Performance Counter

Write a Text File to Disk

```
dim sw as System.IO.StreamWriter;


sw = System.IO.File.CreateText("C:\MyFile.txt");
sw.WriteLine("one");
sw.WriteLine("two");
sw.WriteLine("three");
sw.Close();
```

Read a Text File from Disk

```
dim sr as System.IO.StreamReader;


sr = System.IO.File.OpenText("c:\MyFile.txt");

while sr.Peek() > -1
   LogMessage(sr.ReadLine());

endwhile;

sr.Close();
```

Format a Number Using a .NET Format 'Picture'

```
dim i as integer;


i = 1234;

LogMessage("Total cost: " + i.ToString("$#,###,###.00"));
```

Format a Time Using a .NET Format 'Picture'

```
dim t as time;


t = Now();

LogMessage("The current time is: " + t.ToString("hh:mm:ss")
   + ".");
```

Create an XML Document and Save it to Disk

```
dim doc       as System.Xml.XmlDocument;

dim catalog   as System.Xml.XmlElement;

dim book      as System.Xml.XmlElement;

dim title     as System.Xml.XmlElement;

dim author    as System.Xml.XmlElement;

dim lastName  as System.Xml.XmlElement;

dim firstName as System.Xml.XmlElement;


' create new XML document rooted in catalog

doc = new System.Xml.XmlDocument;

catalog = doc.CreateElement("catalog");

doc.AppendChild(catalog);


' add a book to the catalog

book = doc.CreateElement("book");

title = doc.CreateElement("title");

author = doc.CreateElement("author");

lastName = doc.CreateElement("lastName");
```

```
firstName = doc.CreateElement("firstName");
author.AppendChild(lastName);
author.AppendChild(firstName);
book.AppendChild(title);
book.AppendChild(author);
catalog.AppendChild(book);
book.SetAttribute("isbn", "0385503822");
title.InnerText = "The Summons";
lastName.InnerText = "Grisham";
firstName.InnerText = "John";

' add another book
book = doc.CreateElement("book");
title = doc.CreateElement("title");
author = doc.CreateElement("author");
lastName = doc.CreateElement("lastName");
firstName = doc.CreateElement("firstName");
author.AppendChild(lastName);
author.AppendChild(firstName);
book.AppendChild(title);
book.AppendChild(author);
catalog.AppendChild(book);
book.SetAttribute("isbn", "044023722X");
title.InnerText = "A Painted House";
lastName.InnerText = "Grisham";
firstName.InnerText = "John";

' save the XML document to disk
doc.Save("c:\catalog.xml");
```

Load an XML Document from Disk and Do Look-ups on It

```
dim doc as System.Xml.XmlDocument;
dim node as System.Xml.XmlNode;

doc = new System.Xml.XmlDocument;
doc.Load("c:\catalog.xml");

' find the title of the book whose isbn is 044023722X
```

```
node =
   doc.SelectSingleNode("/catalog/book[@isbn='044023722X']
   /title");
LogMessage(node.InnerText);


'  find all titles written by Grisham
for each node in
   doc.SelectNodes("/catalog/book[author/lastName='Grisham
   ']/title")
   LogMessage(node.InnerText);
next;
```

Query a SQL Server Database

```
dim connection as System.Data.SqlClient.SqlConnection;
dim command as System.Data.SqlClient.SqlCommand;
dim reader as System.Data.SqlClient.SqlDataReader;


connection = new
   System.Data.SqlClient.SqlConnection("server=(local);uid
   =sa;database=northwind");
connection.Open();


command = new System.Data.SqlClient.SqlCommand("select *
   from customers", connection);
reader = command.ExecuteReader();


while reader.Read()
   LogMessage(reader("CompanyName"));
endwhile;
reader.Close();
connection.Close();
```

Execute a SQL Parameterized INSERT Command

```
dim connection as System.Data.SqlClient.SqlConnection;
dim command as System.Data.SqlClient.SqlCommand;
dim regionId as System.Data.SqlClient.SqlParameter;
dim regionDesc as System.Data.SqlClient.SqlParameter;
dim commandText as string;


connection = new
   System.Data.SqlClient.SqlConnection("server=(local);uid
   =sa;database=northwind");
connection.Open();
```

```
commandText = "INSERT INTO Region (RegionID,
    RegionDescription) VALUES (@id, @desc)";
command = new
    System.Data.SqlClient.SqlCommand(commandText,
    connection);
regionId = command.Parameters.Add("@id",
    System.Data.SqlDbType.Int, 4);
regionDesc = command.Parameters.Add("@desc",
    System.Data.SqlDbType.NChar, 50);
command.Prepare();

regionId.Value = 5;
regionDesc.Value = "Europe";
command.ExecuteNonQuery();

regionId.Value = 6;
regionDesc.Value = "South America";
command.ExecuteNonQuery();

connection.Close();
```

Share a SQL Connection (or Any Other .NET Object)

Note Start by creating the C# library shown below called ObjectCache and importing it into the galaxy. .NET SDK must be installed.

```
set path=%path%;C:\Program Files\Microsoft Visual Studio
    .NET\FrameworkSDK\Bin
csc /target:library ObjectCache.cs
```

-------- ObjectCache.cs-------------

```
using System;
using System.Collections;

public class ObjectCache
{
public static void Add(string objectName, object o)
{
objects[objectName] = o;
}

public static void Remove(string objectName)
{
objects.Remove(objectName);
}

public static object Get(string objectName)
{
return objects[objectName];
}

private static Hashtable objects =
   Hashtable.Synchronized(new Hashtable());
}
```

-------- ObjectCache.cs-------------

Then in UserDefined_001 do this:

```
dim connection as System.Data.SqlClient.SqlConnection;

Startup:
connection = new
   System.Data.SqlClient.SqlConnection("server=(local);uid
   =sa;database=northwind");
connection.Open();
ObjectCache.Add("NorthwindConnection", connection);

Shutdown:
ObjectCache.Remove("NorthwindConnection");
connection.Close();
```

Then in UserDefined_002, UserDefined_003, etc. do this:

```
dim connection as System.Data.SqlClient.SqlConnection;
connection = ObjectCache.Get("NorthwindConnection");

if connection <> null then
System.Threading.Monitor.Enter(connection);
```

```
' use the connection
System.Threading.Monitor.Exit(connection);
endif;
```

Create a Look-up Table, Then Do a Look-up on It

```
dim zipcodes as System.Collections.Hashtable;

zipcodes = new System.Collections.Hashtable;
zipcodes["Irvine"] = 92618;
zipcodes["Mission Viejo"] = 92692;
LogMessage(zipcodes["Irvine"]);
```

Fill a String Array, Then Use It

```
dim numbers[3] as string;
dim s as string;

numbers[1] = "one";
numbers[2] = "two";
numbers[3] = "three";

LogMessage(numbers[3]);

for each s in numbers[]
LogMessage(s);
next;
```

Fill a Two-Dimensional Integer Array, Then Use It

```
dim x[2,3] as integer;
dim i as integer;

x[1, 1] = 1;
x[1, 2] = 2;
x[1, 3] = 3;
x[2, 1] = 4;
x[2, 2] = 5;
x[2, 3] = 6;
```

```
LogMessage(x[2, 3]);

for each i in x[]
LogMessage(i);
next;
```

Get the Directories Under c:\

```
dim dir as System.IO.DirectoryInfo;

for each dir in
   System.IO.DirectoryInfo("c:\").GetDirectories()
LogMessage(dir.FullName);
next;
```

Use Exchange to Send an Email

```
dim session as object;
dim msg as object;

session = CreateObject("MAPI.Session");
session.Logon("Employee");
msg = session.Outbox.Messages.Add();
msg.Recipients.Add("<type valid email address here>");
msg.Subject = "Reminder to self";
msg.Text = "Pick up eggs and milk on your way home.";
msg.Send();
session.Logoff();
```

Use SMTP to Send an Email

```
System.Web.Mail.SmtpMail.Send
(
{from:    } "<type valid email address here>",
{to:      } "<type valid email address here>",
{subject: } "Reminder to self",
{body:    } "Pick up eggs and milk on your way home."
 );
```

Call a Web Service to Send an Email

```
' First, generate a wrapper for the web service (.Net SDK
   must be installed).
' To generate wrapper, run the following commands from the
   DOS prompt:
```

```
' set path=%path%;C:\Program Files\Microsoft Visual Studio
   .NET\FrameworkSDK\Bin
' wsdl /namespace:SendMail http://www.xml-
   webservices.net/services/messaging/smtp_mail/mailsender
   .asmx
' csc /target:library Message.cs
' Next import the generated Message.dll library into your
   galaxy.
' Now write your script:

dim m as SendMail.Message;

m = new SendMail.Message;
m.SendSimpleMail
(
{to:      } "<type valid email address here>",
{from:    } "<type valid email address here>",
{subject: } "Reminder to self",
{body:    } "Pick up eggs and milk on your way home."
 );
```

Call a Web Service to Get the Temperature for a Given Zip Code

```
' Requires input string uda me.zipcode and output float uda
   me.temperature.
' First, generate a wrapper for the web service (.Net SDK
   must be installed).
' To generate wrapper, run the following commands from the
   DOS prompt:
'  set path=%path%;C:\Program Files\Microsoft Visual Studio
   .NET\FrameworkSDK\Bin
'  wsdl http://www.vbws.com/services/weatherretriever.asmx
'  csc /target:library WeatherRetriever.cs
' Next import the generated WeatherRetriever.dll library
   into your galaxy.
' Now write your script:

dim wr as WeatherRetriever;

wr = new WeatherRetriever;
me.temperature = wr.GetTemperature(me.zipcode);
```

Use Screen-Scraping to Get the Temperature for a City

```
' Screen-scraping involves downloading a web page,
' then using a regular expression to retrieve the desired
   data.
' Requires input string uda me.CityState, e.g. "Los
   Angeles,CA"
' and output float uda me.temperature.

dim request as System.Net.WebRequest;
dim reader as System.IO.StreamReader;
dim regex as System.Text.RegularExpressions.Regex;
dim match as System.Text.RegularExpressions.Match;

request = System.Net.WebRequest.Create
(
"http://www.srh.noaa.gov/data/forecasts/zipcity.php?inputs
   tring=" +
System.Web.HttpUtility.UrlEncode(me.CityState)
);
reader = new
   System.IO.StreamReader(request.GetResponse().GetRespons
   eStream());
regex = new
   System.Text.RegularExpressions.Regex("<br><br>(.*)&deg;
   F<br>");

match = regex.Match(reader.ReadToEnd());
me.temperature = match.Groups(1);
```

Access an Excel Spreadsheet Using CreateObject

```
dim app as object;
dim wb as object;
dim ws as object;

app = CreateObject("Excel.Application");
wb = app.Workbooks.Add();
ws = wb.ActiveSheet;

ws.Range("A1") = 20;
ws.Range("A2") = 30;
ws.Range("A3") = "=A1*A2";
LogMessage(ws.Range("A3").Value);

wb.Close(false);
```

Access an Excel Spreadsheet Using an Imported Type Library

```
dim app as Excel._Application;
dim wb as Excel._Workbook;
dim ws as Excel._WorkSheet;

app = new Excel.Application;

wb = app.Workbooks.Add();
ws = wb.ActiveSheet;
ws.get_Range("A1").Value = 1000;
ws.get_Range("A2").Value = 1000;
ws.get_Range("A3").Value = "=A1+A2";
LogMessage(ws.get_Range("A3").Value);
wb.Close(false);
```

Access an Office XP Excel Spreadsheet Using an Imported Type Library

```
dim app as Excel.Application;
dim ws as Excel.Worksheet;
dim wb as Excel.Workbook;
dim a1 as Excel.Range;
dim a2 as Excel.Range;
dim a3 as Excel.Range;

app = new Excel._ExcelApplicationClass;
wb = app.ActiveWorkbook;
ws = app.ActiveSheet;

a1 = ws.Range("A1");
a2 = ws.Range("A2");
a3 = ws.Range("A3");
a1.Value = 1000;
a2.Value = 2000;
a3.Value = "=A1*A2";
```

```
LogMessage(a3.Value);
wb.Close(true, "c:\temp.xls", false);
```

Use DDE to Access an Excel Spreadsheet

```
WWPoke("excel", "sheet1", "r1c1", "Hello");
WRequest("excel", "sheet1", "r1c1", me.Greeting);

' Note: use "" to embed double quotes in strings
WWExecute("excel", "sheet1",
   "[SELECT(""R1C1"")][FONT.PROPERTIES(,""Bold"")]");
```

Read a Performance Counter

```
' Requires output float uda me.PercentProcessorTime.
' Declarations
dim counter as System.Diagnostics.PerformanceCounter;

' Startup
counter = new System.Diagnostics.PerformanceCounter;
counter.CategoryName = "Processor";
counter.CounterName = "% Processor Time";
counter.InstanceName = "0";

' Execute
me.PercentProcessorTime = counter.NextValue();
```

C H A P T E R 7

Working with Containment

Containment is the relationship by which one object can contain another, such as a reactor containing an agitator.

This chapter describes the concept of containment and how to use it to build your Industrial Application Server application.

Contents

Contained Objects

Containment relationships allow objects to be organized in a hierarchical manner. You can build objects that represent complex devices consisting of smaller, simpler devices. Higher level objects can contain lower level objects. This allows you to more closely model complex plant equipment.

ApplicationObjects can only be contained by other ApplicationObjects. Areas can only be contained by other Areas.

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

The fully-qualified name of a contained object includes the name of the object that contains it. This is the object's hierarchical name.

For example, an instance of a `$Reactor` is named `Reactor1`. An instance of `$Valve` called `Valve1` is contained within the instance of `$Reactor`.

Change the contained name of `Valve1` to `InletValve`. Now `Valve1` 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.

For example, the contained name of the `Reactor1.InletValve` valve can be changed to `Outlet`. The name (tagname) of the `InletValve` object has not changed. Only its hierarchical name is simplified within the context of the containment.

Examples of Containment

UserDefined objects are simple objects that are commonly used to contain other objects (for example, a Tank object containing two DiscreteDevice objects to represent its inlet and outlet valves).

The following scenario is an example of containment:

1. Create the following instances: `Tank1` from $UserDefined and `Valve1` from $DiscreteDevice. At this point, `Valve1` has only one name, `Valve1`.
2. In the **Model** or **Deployment View**, drag `Valve1` on to `Tank1`.

Note If `Tank1` already contains an object whose contained name is `Valve1`, the Galaxy generates a unique contained name for the newly contained object (such as `Valve1_1`).

3. Change the contained name of `Valve1` within `Tank1` to `Outlet`. `Valve1` now has two names, `Valve1` and `Tank1.Outlet`.
4. Create the following instance: `Reactor1` from $UserDefined.
5. In the **Model** or **Deployment View**, drag `Tank1` on to `Reactor1`.
6. Change the contained name of `Tank1` to `Tank`. `Tank1` now has two names, `Tank1` and `Reactor1.Tank`. Also, `Valve1` has a three-part contained name: `Reactor1.Tank.Outlet`.

For the three objects in this example (`Reactor1` containing `Tank1` containing `Valve1`), the following hierarchy is in place:

Tagname	Hierarchical Name
`Reactor1`	-
`Tank1`	`Reactor1.Tank`
`Valve1`	`Reactor1.Tank.Outlet` or `Tank1.Outlet`

The following scenario is an example of a template containing another template:

Note Templates do not have Tagnames, which are reserved for instances.

1. Derive the following templates: `$Tank` from `$UserDefined` and `$Valve` from `$DiscreteDevice`.
2. Derive `$Inlet` from `$Valve`.

   Note The following examples assume instance names that were renamed from the default names provided when first created.

3. In the **Template Toolbox**, drags-and-drops `$Inlet` on to `$Tank`. If `$Tank` already contains a template named `Inlet`, the Galaxy generates a unique name for the new contained template (such as `Inlet_1`.).

   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`.

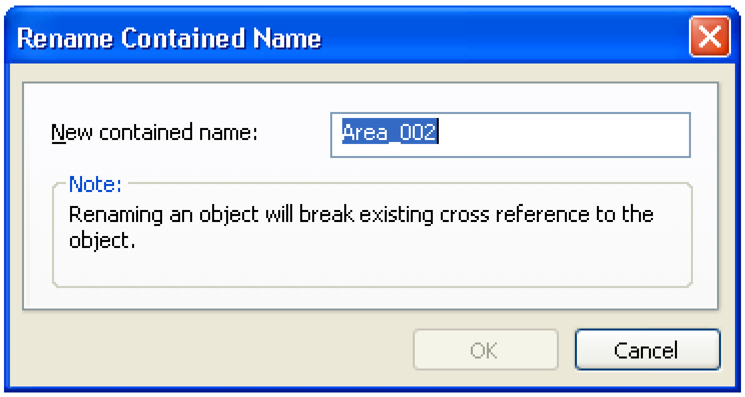
Renaming an Object's Contained Name

When renaming an object's contained name, ensure that the object is neither checked out to another user nor currently deployed.

The new object's contained name must comply with naming restrictions and cannot be the same contained name as an existing contained object within the same level of hierarchy in the containment relationship. See Allowed Names/Characters.

To rename an object's contained name

1. Select the object in an **Application View**.
2. On the **Edit** menu, click **Rename Contained Name**. The **Rename Contained Name** dialog box appears.



New contained name: Area_002

Note:
Renaming an object will break existing cross reference to the object.

3. Type a new contained name.
4. Click **OK** to change the object's contained name. After renaming, all IDEs connected to the Galaxy reflect the object's new contained name.

WARNING! References from other objects to the object being renamed are invalidated. Objects with broken references receive bad quality data at runtime.

ApplicationObject Containment

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

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

- Inlet Valve
- Agitator

- Outlet Valve
- Level

To enable referencing and flexibility within scripting, these objects can be referenced in several different ways. Each must have a unique Tagname, which is equivalent to their plant identification number, such as:

- Inlet Valve = V101
- Agitator = A102
- Outlet Valve = V103
- Level = LT104

There is no way to write a generic script for controlling the tank that references these objects in a template. However, you can rename the contained name of each object to `Inlet`, `Agitator`, `Outlet` and `Level`, respectively. Within the context of each hierarchy, those contained names are unique. So if the tank is called `T001`, with naming convention becomes the following:

- `T001.Inlet`
- `T001.Agitator`
- `T001.Outlet`
- `T001.Level`

This naming convention adds context to the instances contained by `T001`. In addition, you can use containment references in scripts such as:

- `Me.Outlet`: Allows a script running within the parent object to reference generically its children.
- `MyContainer.Inlet`: Allows a script running in any of the children objects to reference other children or properties of the parent object.

This relationship appears for templates in the **Template Toolbox**. For instances, the relationship appears in both the **Model** and **Deployment Views**. The **Template Toolbox** is divided into toolset sections. Within each toolset, templates are shown in a tree. If a template is one that contains other templates, it can be expanded to show the containment under that template. The template can have many layers of containment (cannot exceed 10 layers).

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

The **Derivation View** does not show containment relationships. It shows templates and instances with regard to containment in the follow ways:

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

Containment of instances is limited to Areas containing other Areas and AppObjects containing other AppObjects.

Renaming can be done on an instance’s tagname and contained name. A template only has a template name.

C H A P T E R 8

Working with References

References are the primary component of the system of identification and communication between objects in the ArchestrA environment. Each object, each attribute and each property can be referenced in a unique way. Those references are communicated over the ArchestrA messaging system.

This chapter describes the concept of references and how to use reference strings in creating your Industrial Application Server application.

Contents

Messaging Exchange and Attributes

All object attributes have properties, such as Value and Quality. Any data read or write attempted over ArchestrA messaging exchange system is deterministic in nature. If a requested operation cannot be performed, the requesting client is notified about the problem.

Messaging exchange provides the following features:

- Guaranteed response
- Name signatures
- Status
- Data quality
- Message order preservation within a priority system
- AppEngine-to-AppEngine buffering (snapshotting)
- Publish-subscribe heartbeats

Reference Strings

Reference strings refer to an object or to data within one of its attributes. A reference string consists of an object's reference string plus an attribute's reference string.

```
AutomationObject Reference + Attribute Reference
```

In the example, `TIC101.PV`, `TIC101` is the AutomationObject reference and `PV` is the attribute reference.

Even more simply, a reference string is the object name plus the attribute name.

```
ObjectName.AttributeName
```

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 chars, which are separated by periods. A substring cannot contain a period. Mathematical operator characters are not allowed. Also, at least one character in each substring must be non-numeric.

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:

- Me
- MyContainer
- MyArea
- MyPlatform
- MyEngine.

Relative references are especially important in templates because absolute references are usually inappropriate.

The following are property references:

- .Name
- .Value
- .Type
- .Quality.

If a string has a property name that is reserved in the ArchestrA environment, you can still use it. The PROPERTY keyword must be part of the string (for example, PROPERTY(propertyName)). In all other cases, the PROPERTY keyword (which is not case sensitive) is not required.

The Value property is assumed if no property reference is specified.

A reference string can optionally 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.

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:

```
Tank1
   Inlet Valve (InletValve)
   Outlet Valve (OutletValve)
```

Reference String Format

These symbols apply to the reference strings that follow:

::=	means	“can be replaced by”
|	means	“or”
[]	means	“contents optional”
{}	means	“contents can be left out, used once or repeated”

Other items outside of angle brackets “<>” are literals

- 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 runtime)
- 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 :=]
- quotes are not allowed in tagNames, primitive names, or attribute names
- galaxy_identifier ::= <letter> | <digit>

Notes:

- <tag_name> is an object's unique name.
- <contained_name> is an object's (optional) contained name that 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 and this 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 supplied explicitly. The exception to this rule is a reference that is preceded with an @ sign. Such a 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 an 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. Industrial Application Server 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.

Important! The Galaxy has primary responsibility for resolving reference strings. If the GR is not available, resolution is done on a peer-to-peer level. After initial resolution occurs, an object is provided an alias that efficiently handles references to its location across your network. If an object is relocated or renamed, the reference string resolution function is repeated and a new alias provided.

References and Cross-References

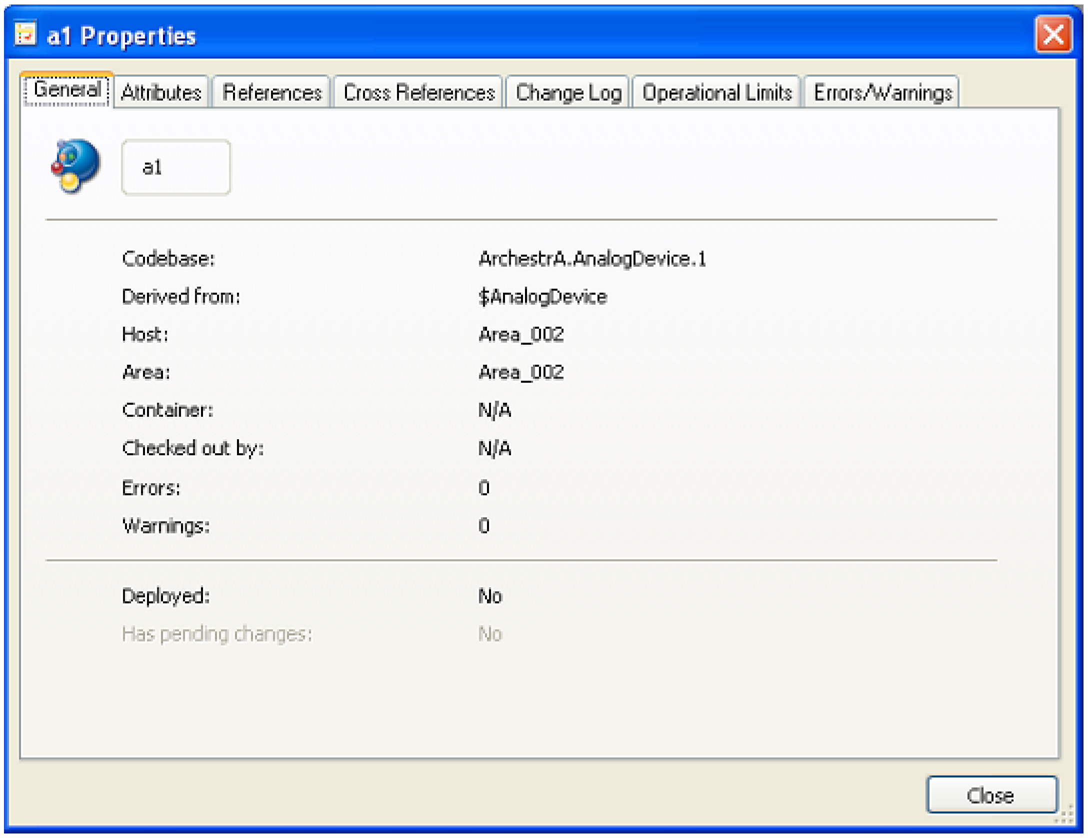
To determine an object’s references (which objects it references) and cross-references (which object is referencing it), use the object **Properties** dialog box. To open the **Properties** dialog box, click **Properties** on the **Galaxy** menu.

Note References and cross-references shown in the Properties dialog box only refer to interobject communications. Area associations, containment or host assignments are not included.

Properties Dialog Box

This dialog lists referencing, cross referencing, change log, and errors/warnings (among other data) specific to the selected object. Use this information when making decisions about managing objects.

For example, to delete an object, use this dialog box to check which objects it references and which objects are cross-referencing it. Deleting an object (as well as renaming instances and contained names) can break references.

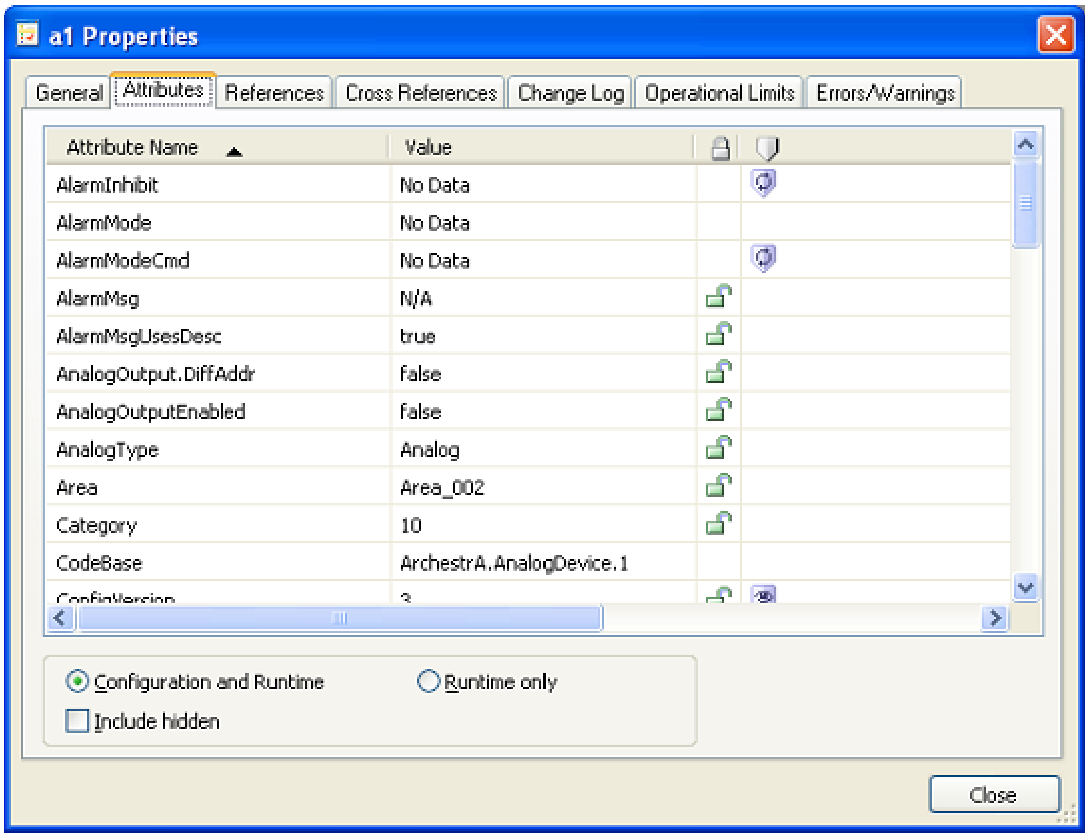


a1

Codebase:	ArchestrA.AnalogDevice.1
Derived from:	$AnalogDevice
Host:	Area_002
Area:	Area_002
Container:	N/A
Checked out by:	N/A
Errors:	0
Warnings:	0
Deployed:	No
Has pending changes:	No

The **General** page shows the following data:

- **Codebase**: The object name and version.
- **Derived From**: The object’s parent template.
- **Host**: The object’s host object.
- **Area**: The object’s area object.
- **Container**: The object’s container object.
- **Checked Out By**: Who has the object checked out.
- **Errors**: The number of errors currently set against the object.
- **Warnings**: The number of warnings currently set against the object.
- **Deployed**: Whether the object is deployed or not.
- **Has Pending Changes**: Whether the object has configuration changes pending.



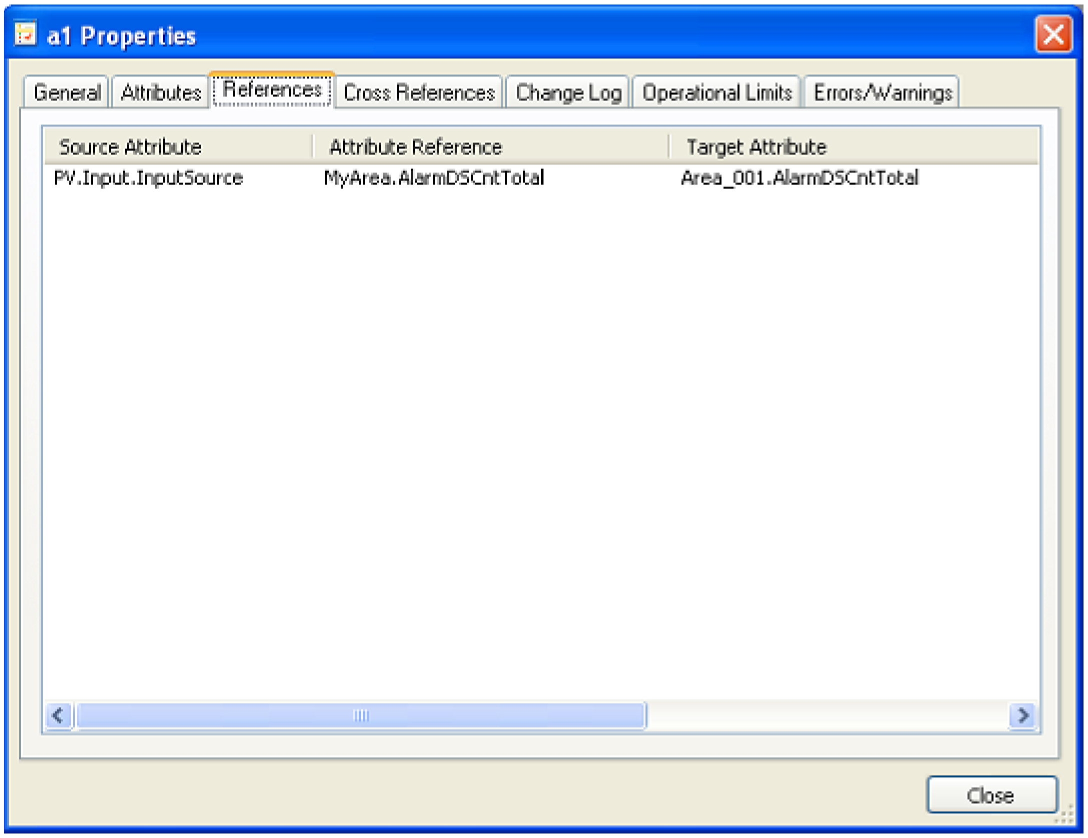
Attribute Name ▲	Value	🔒	🛡
AlarmInhibit	No Data		🛡
AlarmMode	No Data		
AlarmModeCmd	No Data		🛡
AlarmMsg	N/A	🔓	
AlarmMsgUsesDesc	true	🔓	
AnalogOutput.DiffAddr	false	🔓	
AnalogOutputEnabled	false	🔓	
AnalogType	Analog	🔓	
Area	Area_002	🔓	
Category	10	🔓	
CodeBase	ArchestrA.AnalogDevice.1		
ConfigVersion	3	🔓	🛡

[✓ Configuration and Runtime] Runtime only

☐ Include hidden

The **Attributes** page shows the following data:

- **Attribute Name** list: The selected object’s attributes, as well as their current value, and locked and security status. The content of this list depends on whether **Configuration and Runtime** or **Runtime Only** is selected.
- **Configuration and Runtime**: Select this option button to show both configuration and runtime attributes of the selected object.
- **Runtime Only**: Select this option button to show only runtime attributes of the selected object.
- **Include Hidden**: Select this check box to show hidden attributes of the selected object.



Source Attribute	Attribute Reference	Target Attribute
PV.Input.InputSource	MyArea.AlarmDSCntTotal	Area_001.AlarmDSCntTotal

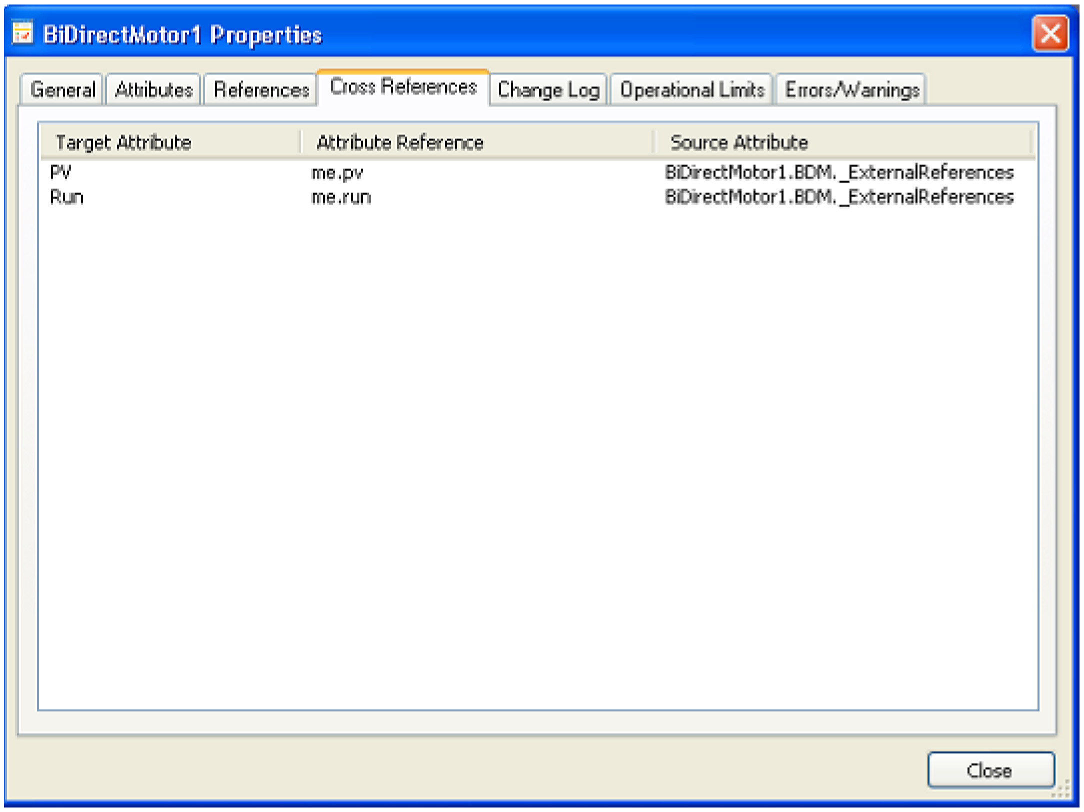
The **References** page shows references in the selected object to other objects in the Galaxy.

The “Source Attribute” is the attribute which contains a reference to another attribute in the galaxy application.

The “Attribute Reference” is the reference string within the “Source Attribute”. This can be either an absolute reference or a relative reference.

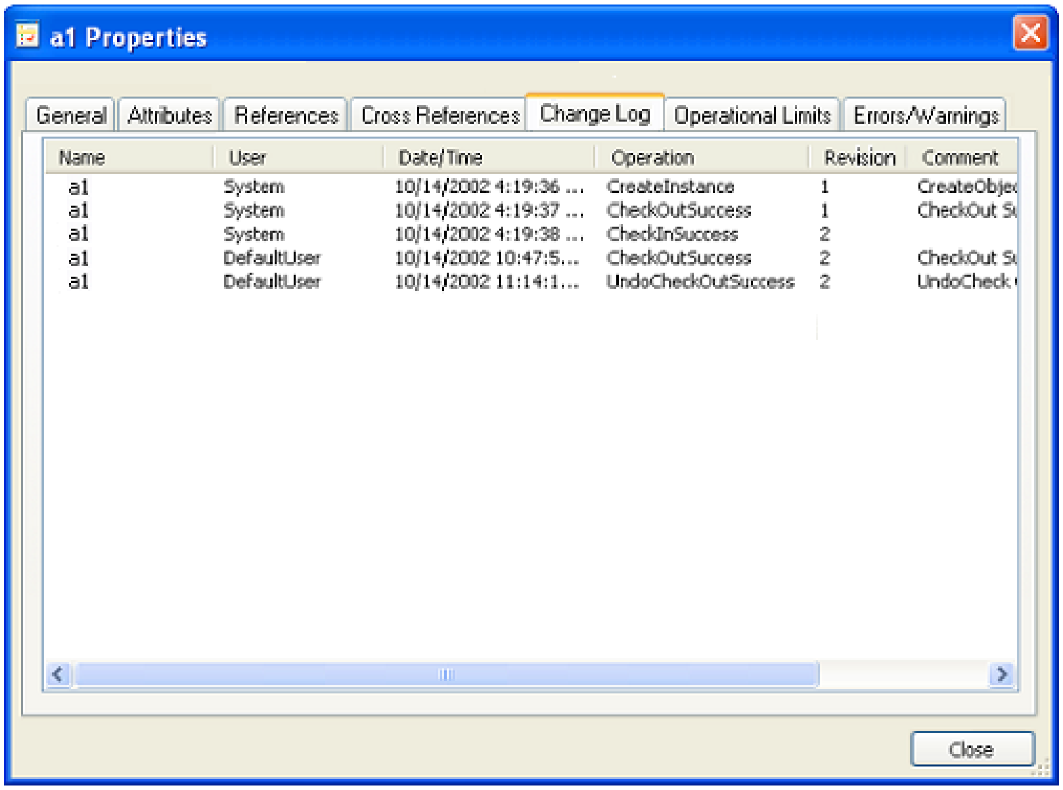
The “Target Attribute” is the absolute reference of the “Attribute Reference”. In the case of a reference to a dynamic attribute, the “Target Attribute” only lists the Tagname name of the instance.

A dynamic attribute is an attribute that gets created at runtime and exists only in runtime. This is the case with a device integration (DI) reference to a hardware register. The attribute referencing a hardware register does not exist at configuration time by default. DI instances create the attribute dynamically at runtime if the hardware register exist in the target device.



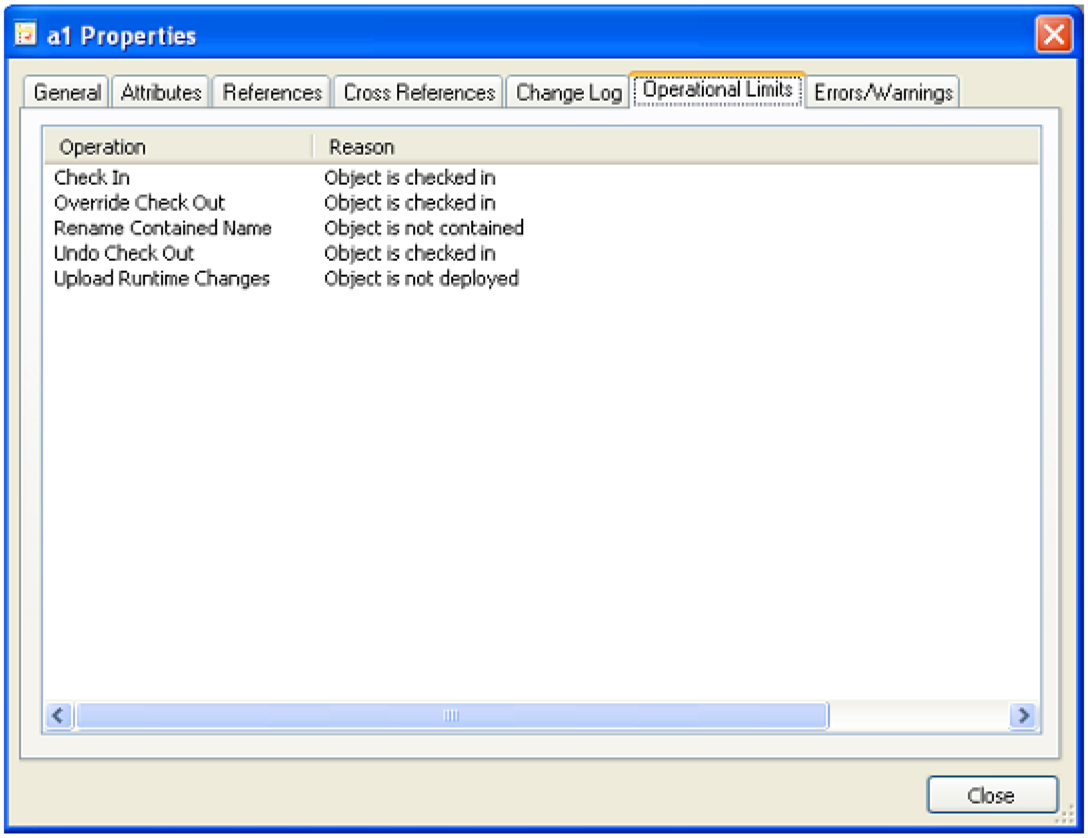
Target Attribute	Attribute Reference	Source Attribute
PV	me.pv	BiDirectMotor1.BDM._ExternalReferences
Run	me.run	BiDirectMotor1.BDM._ExternalReferences

The **Cross References** page shows references to the selected object from other objects in the Galaxy.



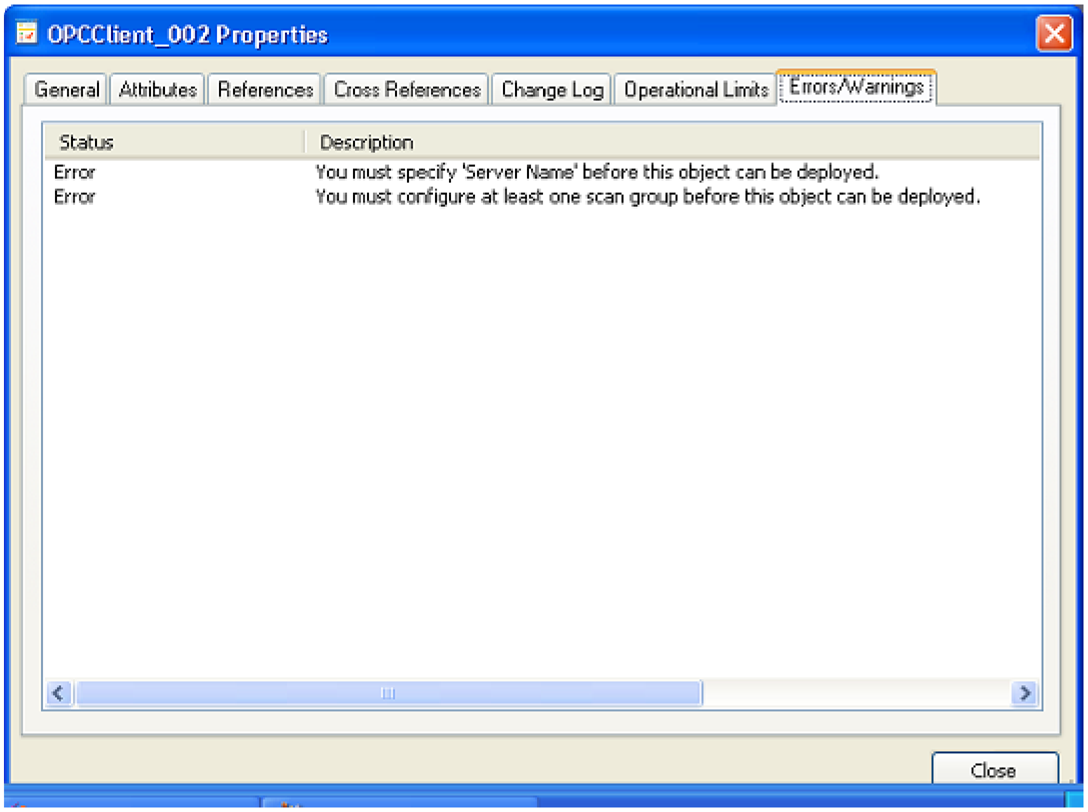
Name	User	Date/Time	Operation	Revision	Comment
a1	System	10/14/2002 4:19:36 ...	CreateInstance	1	CreateObjec
a1	System	10/14/2002 4:19:37 ...	CheckOutSuccess	1	CheckOut Su
a1	System	10/14/2002 4:19:38 ...	CheckInSuccess	2	
a1	DefaultUser	10/14/2002 10:47:5...	CheckOutSuccess	2	CheckOut Su
a1	DefaultUser	10/14/2002 11:14:1...	UndoCheckOutSuccess	2	UndoCheck

The **Change Log** page shows a log of operations that occurred for the selected object. Data include the user responsible for the operation and comments the user made regarding the changes to the object.



Operation	Reason
Check In	Object is checked in
Override Check Out	Object is checked in
Rename Contained Name	Object is not contained
Undo Check Out	Object is checked in
Upload Runtime Changes	Object is not deployed

The **Operational Limits** page shows the operational conditions the object is in. For example, planning to do an upload of runtime changes on this object is pointless because the “Object is not deployed.”

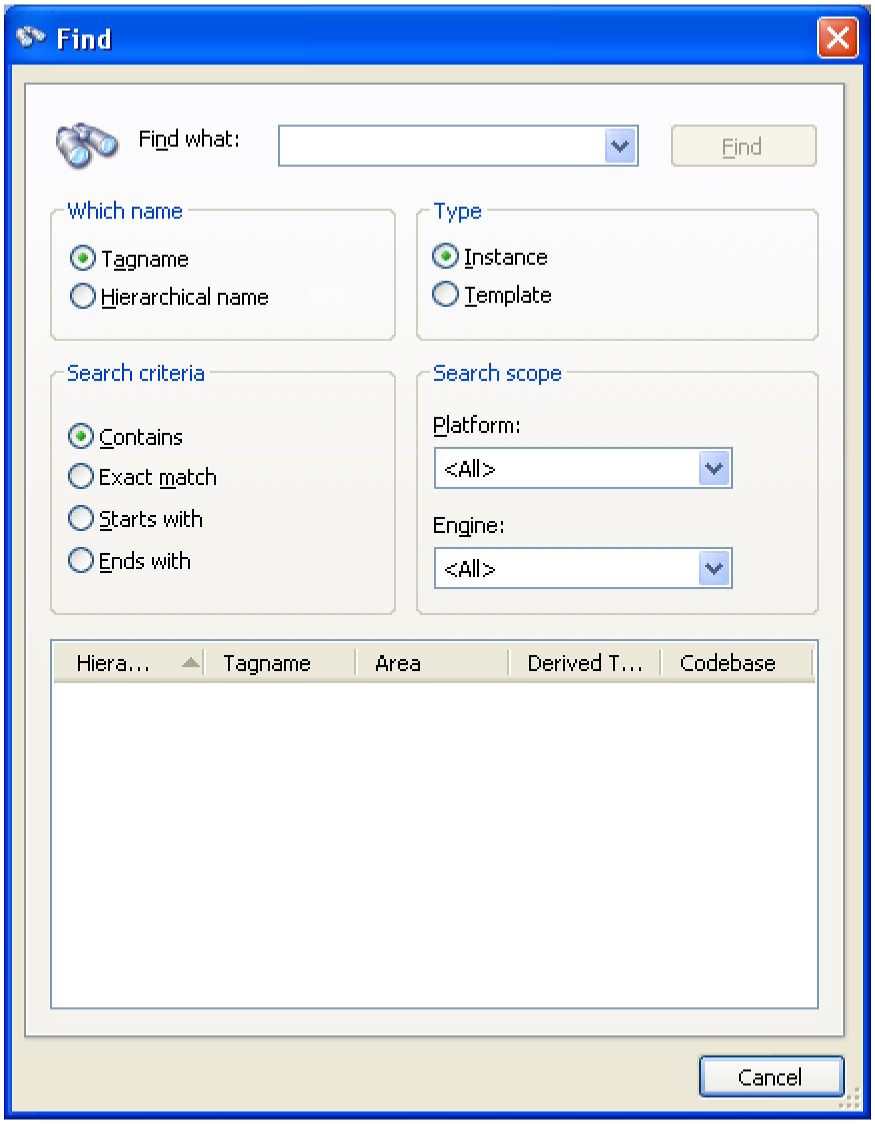


Status	Description
Error	You must specify 'Server Name' before this object can be deployed.
Error	You must configure at least one scan group before this object can be deployed.

The **Errors/Warnings** page shows any problem conditions with respect to the selected objects configuration and runtime.

Find Object

Use the **Find** dialog box to help you find objects you are having difficulty finding manually in your Galaxy.


Find

Find what: Find

Which name
[✓ Tagname]
Hierarchical name

Type
[✓ Instance]
Template

Search criteria
[✓ Contains]
Exact match
Starts with
Ends with

Search scope
Platform: <All>
Engine: <All>

Hiera...	Tagname	Area	Derived T...	Codebase

Cancel

This dialog box provides several options for developing search criteria.

- Enter the object’s name in the **Find What** box.
- Select either **Tagname** or **Hierarchical Name** in the **Which Name** group to indicate to the search engine what type of name you have entered in the **Find What** box.
- Provide which kind of object you are searching for in the **Type** group: **Instance** or **Template**. If you select **Template**, the **Which Name** and **Search Scope** groups are made unavailable.
- Indicate in the **Search Criteria** group how your entry in the **Find What** box relates to the object’s name: **Contains**, **Exact Match**, **Starts With** or **Ends With**.
- Limit the search scope by entering the proper data in the **Search Scope** group. After you enter the information, click **Find**. Search results appear in the bottom pane.

Double-click an object in the results pane, and the object is located and highlighted for you in the **Application Views** pane of the IDE's Main Window.

If you double-click a Backup AppEngine (see ArchestrA Redundancy for more information), the IDE focus shifts to the Deployment View and the object is highlighted there.

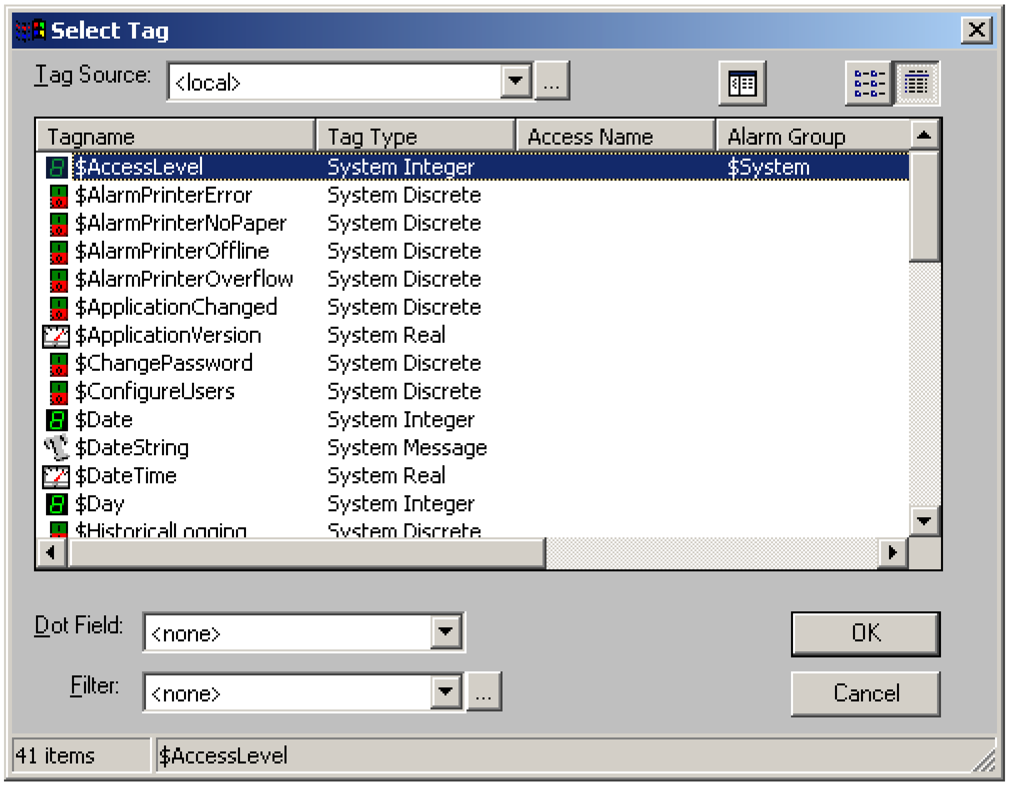
Browsing Galaxy References from InTouch

Use the InTouch Tag Browser in unlimited selection mode to browse and include references from an Industrial Application Server Galaxy in InTouch applications you are developing.

Note The Bootstrap and IDE must be installed on the InTouch node to create the TagSource to allow for browsing of Galaxy objects.

The following lists the primary methods that you can use to access the Tag Browser in unlimited selection mode:

- Double-clicking an animation link tagname or expression input box.
- Double-clicking an ActiveX or wizard tagname or expression input box.
- Double-clicking a blank area in any InTouch QuickScript window.
- In the InTouch QuickScript editor, selecting the **Tagname** command on the **Insert** menu.
- Pressing the **Alt+N** keys in the InTouch QuickScript editor.
- Double-clicking a blank **New Name** box in the **Substitute Tagnames** dialog box.
- Double-clicking the **TagnamedotfieldsName** input box in the SQL Access **Bind List Configuration** dialog box.



Select Tag

Tag Source: <local>

Tagname	Tag Type	Access Name	Alarm Group
$AccessLevel	System Integer		$System
$AlarmPrinterError	System Discrete		
$AlarmPrinterNoPaper	System Discrete		
$AlarmPrinterOffline	System Discrete		
$AlarmPrinterOverflow	System Discrete		
$ApplicationChanged	System Discrete		
$ApplicationVersion	System Real		
$ChangePassword	System Discrete		
$ConfigureUsers	System Discrete		
$Date	System Integer		
$DateString	System Message		
$DateTime	System Real		
$Day	System Integer		
$HistoricalLogging	System Discrete		

Dot Field: <none>

Filter: <none>

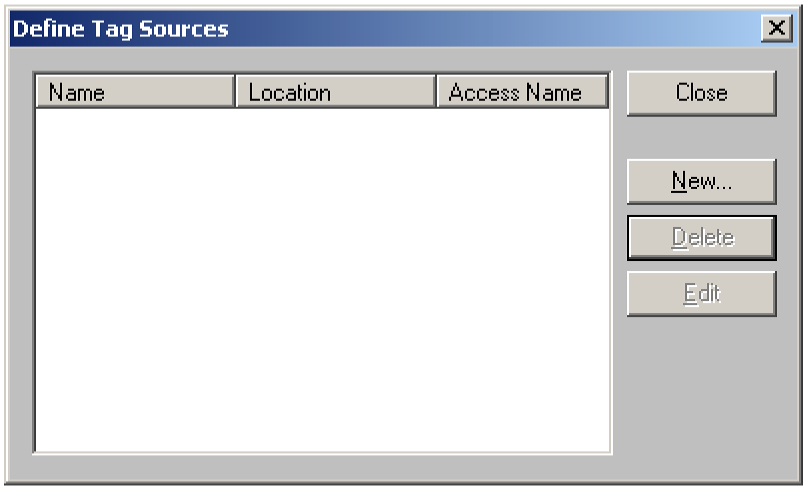
41 items $AccessLevel

For complete information about using the InTouch Tag Browser, refer to the InTouch User’s Guide, The Tag Browser section of Chapter 6, Tagname Dictionary.

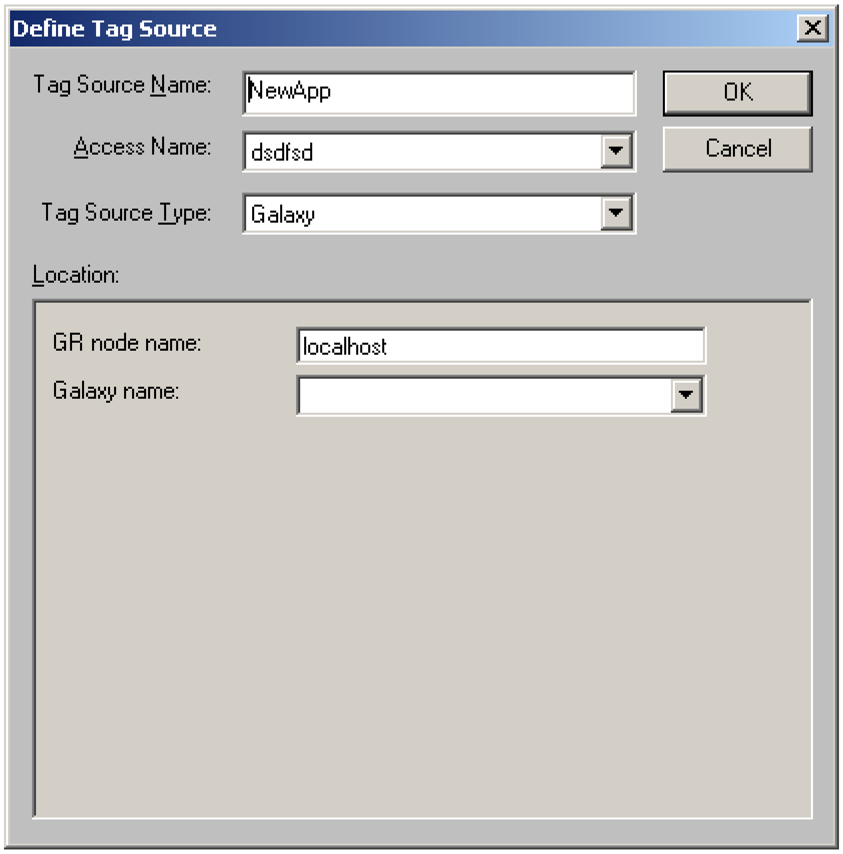
Before you can browse Galaxy references, you must define a new tag source.

To define a new tag source

1. Click the **Define Tag Sources** button (ellipses button to the right of the **Tag Source** box). The **Define Tag Sources** dialog box appears.



2. Click **New** to open the **Define Tag Source** dialog box.



Tag Source Name: NewApp

Access Name: dsdfsd

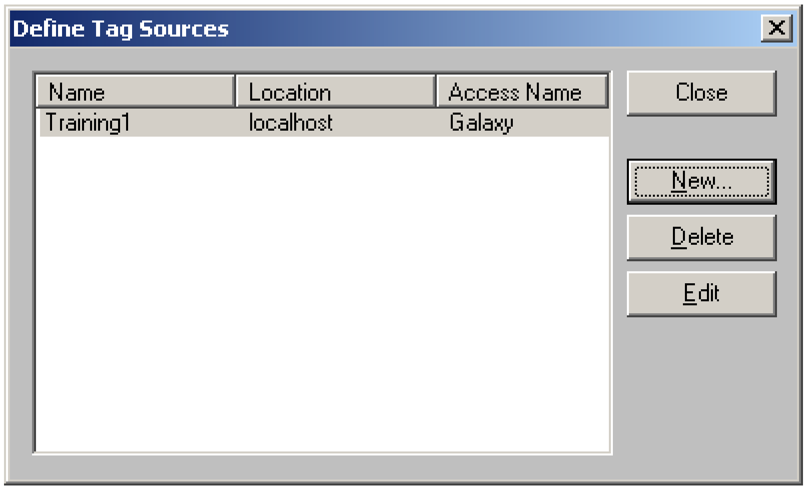
Tag Source Type: Galaxy

Location:

GR node name: localhost

Galaxy name:

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. Type in the **GR Node Name** box the Host Name of the computer on which the Galaxy is located. If the Galaxy is located on the same computer, use the default setting (localhost).
6. Select the name of your Galaxy from the **Galaxy Name** list. For example: training. Assuming the examples given in steps 3 and 5, the **Define Tag Sources** dialog box appears as follows.

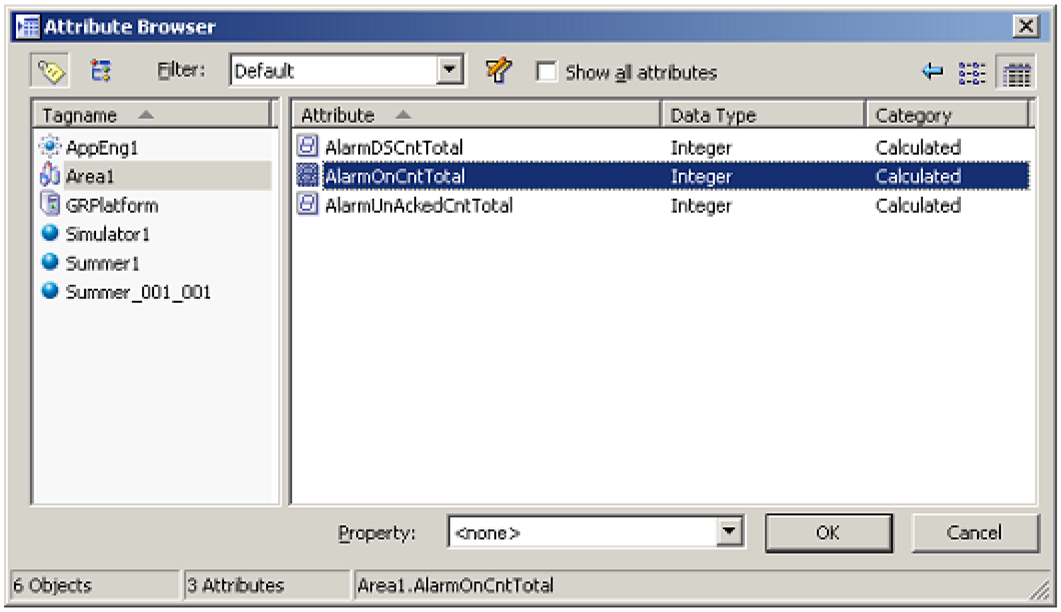


Name	Location	Access Name
Training1	localhost	Galaxy

7. Click **Close**. The Tag Browser appears.

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 procedure (Training1 in the example). A small message box shows the name of the Galaxy you are connecting to (Connecting to “training”), and the Attribute Browser appears.



Filter: Default

Tagname
AppEng1
Area1
GRPlatform
Simulator1
Summer1
Summer_001_001

Attribute	Data Type	Category
AlarmDSCntTotal	Integer	Calculated
AlarmOnCntTotal	Integer	Calculated
AlarmUnAckedCntTotal	Integer	Calculated

Property: <none>

6 Objects | 3 Attributes | Area1.AlarmOnCntTotal

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** is automatically opened. 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 Attribute Browser.

C H A P T E R 9

Working with Security

ArchestrA security prevents users from performing unauthorized activities, including users of:

- The IDE when configuring and managing objects.
- The ArchestrA System Management Console (SMC) when performing maintenance and system administration functions.
- Any runtime operations.

Security not only controls access to user interfaces in the ArchestrA environment but also access to object attributes and the data they represent. The security icons associated with object attributes (see “Working with Object Editors“) map directly to control points in the ArchestrA security model.

Each Galaxy in the Galaxy Repository manages its own security model. The security schema managed in a Galaxy is a three-level configuration model that includes the creation and maintenance of the following:

- security groups associated with specific objects in the Galaxy
- user roles associated with specific system administration, configuration and runtime (operational) permissions, which map to security groups
- users associated with specific roles

This kind of security matrix defines a cascading model of users associated with specific roles that are associated with specific security groups associated with specific objects. This kind of model allows a user’s runtime permissions to vary from object to object, action to action and process to process.

Use the **Configure Security** dialog box to configure a Galaxy’s level of security and manage its capabilities.

This chapter describes the architecture of ArchestrA security and how to use it to manage access to configuration and runtime aspects of your Industrial Application Server application.

For more information on ArchestrA security, refer to your *Factory Suite A*^{2} *Deployment Guide.*

Contents

Configuring Security

Three conditions must be met before you can gain access to a Galaxy’s security editor:

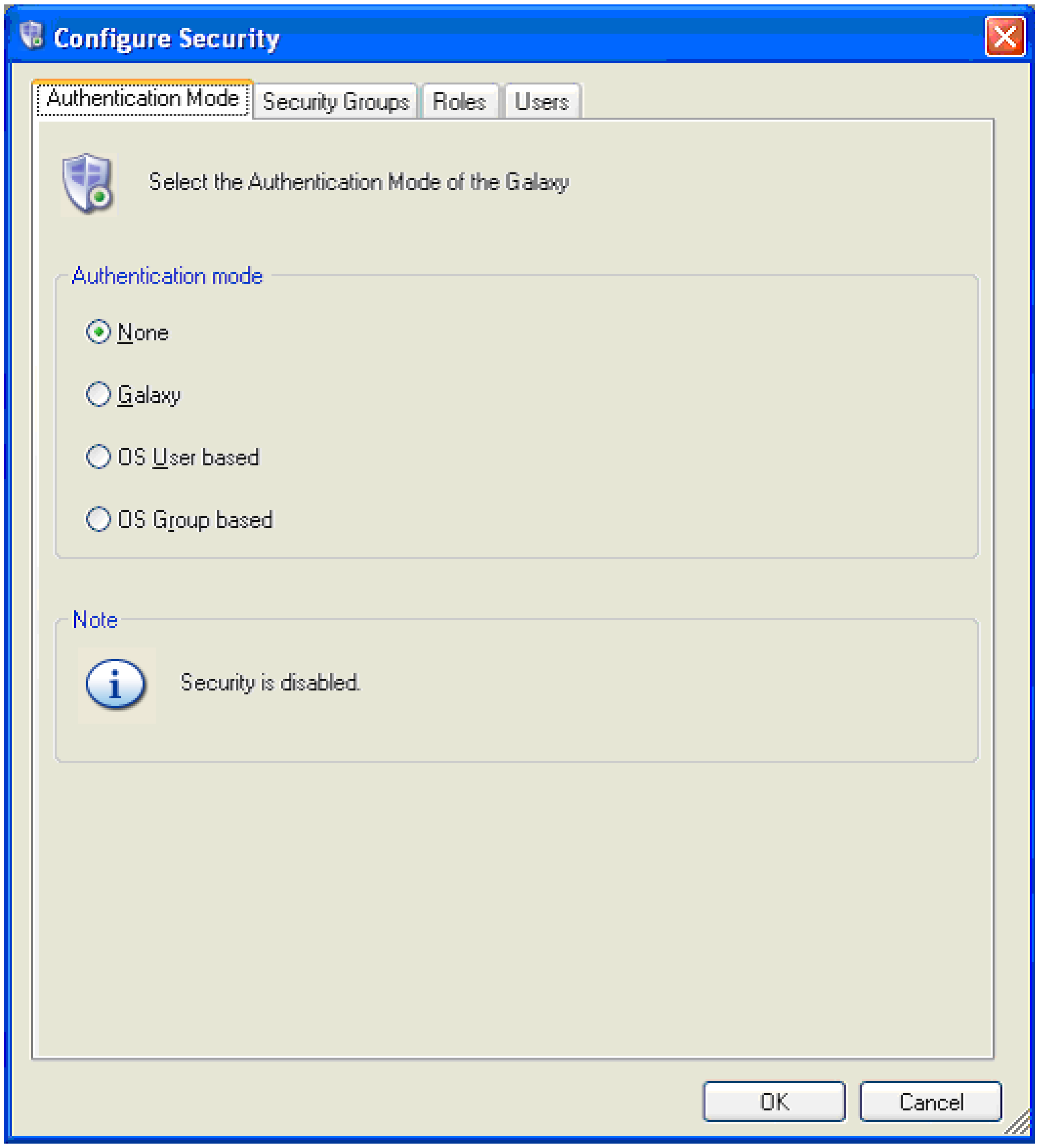
- No other user can be connected to the Galaxy.
- All objects in the Galaxy must be checked in.
- Your user profile must have configuration permissions to change Framework Configuration/Modify Security Model, if security was configured previously.

If you attempt to open the security editor before all three conditions are met, a warning message appears and you are denied access. Another user who attempts to open the IDE while you are configuring security is denied access to the Galaxy.

You can see the current security model in read-only mode by pointing to **Configure** on the **Galaxy** menu and then clicking **View Security**.

Begin configuring Galaxy security by pointing to **Configure** on the **Galaxy** menu and then clicking **Security**. The **Configure Security** dialog box appears.

Authentication Mode



Select the Authentication Mode of the Galaxy

Authentication mode

[✓ None]
Galaxy
OS User based
OS Group based

Note

Security is disabled.

On the **Authentication Mode** page, select the mode of Galaxy security:

- **None**: The default setting for new Galaxies, this mode is considered Open Security. It leaves all functions open to all users. No authentication is provided for any operations in the ArchestrA configuration or runtime environment. No login dialog boxes appear for operating Industrial Application Server utilities or runtime processes.
- **Galaxy**: This mode uses local galaxy configuration to authenticate the user. Use this setting to create a user security system controlled by the Galaxy database.

- **OS User Based**: This mode enables the Authorization of individual OS users. Use this setting to take advantage of the operating system's (NT) user authentication system, on an individual user basis.
- **OS Group Based**: This mode enables the Authorization for users based on which OS Groups they have been assigned to. Use this setting to take advantage of the operating system's user authentication system, on a group basis. When you select **OS Group Based** Authentication mode, the following **Configurable Intervals** options appear:
  - **Login Time**: This value (in milliseconds) is the timeout period during which the system validates the user’s membership against the OS groups selected as ArchestrA Roles. After this timeout period, the login operation defaults to the local cache. The result of a successful login for a value other than 0 (zero) is that local cache is stored/updated. If the login operation times out and the user has performed a previous ArchestrA login on the computer, local cache is used; if the user has never performed an ArchestrA login on the computer, the ArchestrA login fails. Minimum allowed value is 0 (zero); maximum is 9,999,999. Default value 1000. A value of 0 (zero), turns off this feature, in that the operation does not time out. Use the **Login Time** option primarily for intermittent or slow network scenarios. The value to use in this option is determined by the speed of your network and by the number of groups configured in ArchestrA. The slower the network or the higher the number of groups, the greater the value for **Login Time**.
  - **Role Update**: This value (in milliseconds) is the time between each validation attempt per OS group for the user’s membership when a login is attempted. The user membership update is done one role per **Role Update** interval to minimize network usage. Minimum allowed value is 0 (zero); maximum is 9,999,999. Default value is 0 (zero), which switches off this feature (the operation does not pause between validating user membership and groups). Use this option primarily for intermittent or slow network scenarios. 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.

Note When you select Authentication Modes, note the messages relevant to your ArchestrA installation that appear in the Note box.

Authentication Mode selections provide the following general results:

- Open security gives all users the Defaultuser credentials. No login dialog boxes are shown to users during configuration, administration or runtime operations. Login dialog boxes appear for all other Authentication Modes.
- If you previously configured security under one Authentication Mode and then switch authentication modes, only those users created while configuring the new mode are enabled. Other users are not deleted, just disabled in the new mode.
- When you close the **Configure Security** dialog box after selecting any **Authentication Mode** other than **None**, you must login. This action ensures that the new security model can be enforced.

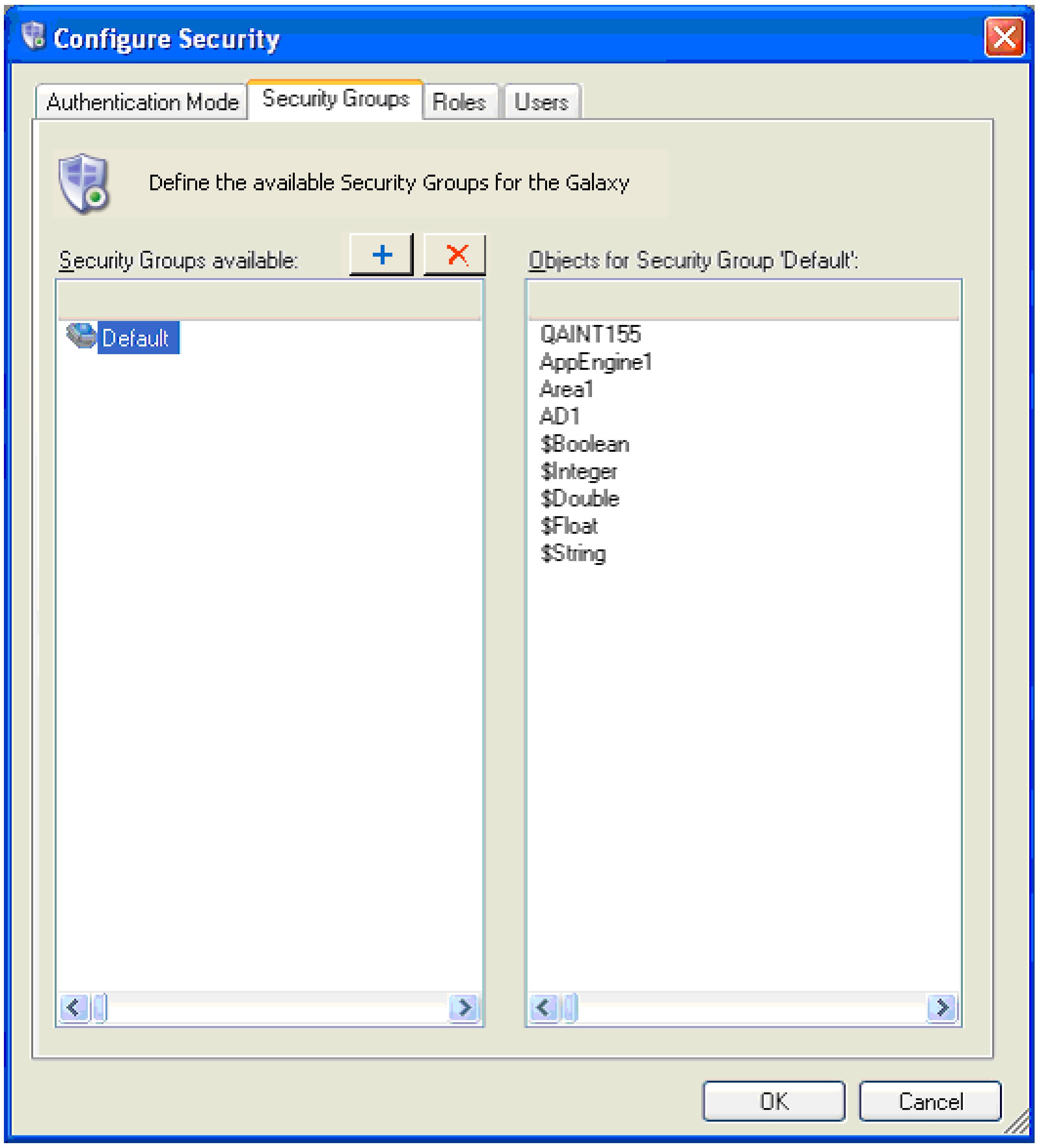
- When you switch to **None** from another **Authentication Mode** and click **OK**, the IDE shuts down.
- When Galaxy authentication is selected, each user must provide a user name and password in a login dialog box. The security system authenticates the user's credentials against Galaxy user data. Access to all operations in the IDE and anywhere in the ArchestrA environment are granted based on the logged in user's associated roles and permissions. The IDE customizes the user interface to the user's previous preferences (for example, which **Application Views** are shown). The logged in user's name is shown in the status bar of the IDE.
- When OS User Based authentication is selected, each user must provide a domain, user name and password in a login dialog box. The security system ensures that the OS user is authorized to use the IDE.
- When OS Group Based authentication is selected, each user must provide a domain, user name and password in a login dialog box. The security system first ensures that the user is authorized to use the IDE. Then the system authorizes the user's credentials against operating system groups mapped to security roles in the Galaxy.

  Note In both OS-based authentication modes, a user is not shown a log in dialog box if that user has authorization to use that ArchestrA utility.

- A user can have multiple accounts within the security system. 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 ArchestrA environment. To switch between levels of authority, the person must login as the new user. To do this, click **Change User** on the **Galaxy** menu.

For more information about **OS Group Based** authentication, see to About OS Group Based Security.

Security Groups



Define the available Security Groups for the Galaxy

Security Groups available:

Default

Objects for Security Group 'Default':

QAINT155
AppEngine1
Area1
AD1
$Boolean
$Integer
$Double
$Float
$String

On the **Security Groups** page, create and manage the security groups that make sense for your organization. Every object in the Galaxy belongs to only one security group. These security groups are mapped to roles on the **Roles** page. In the example above, all objects in the Galaxy (shown in the right pane) are associated with the Default security group (shown in the left pane).

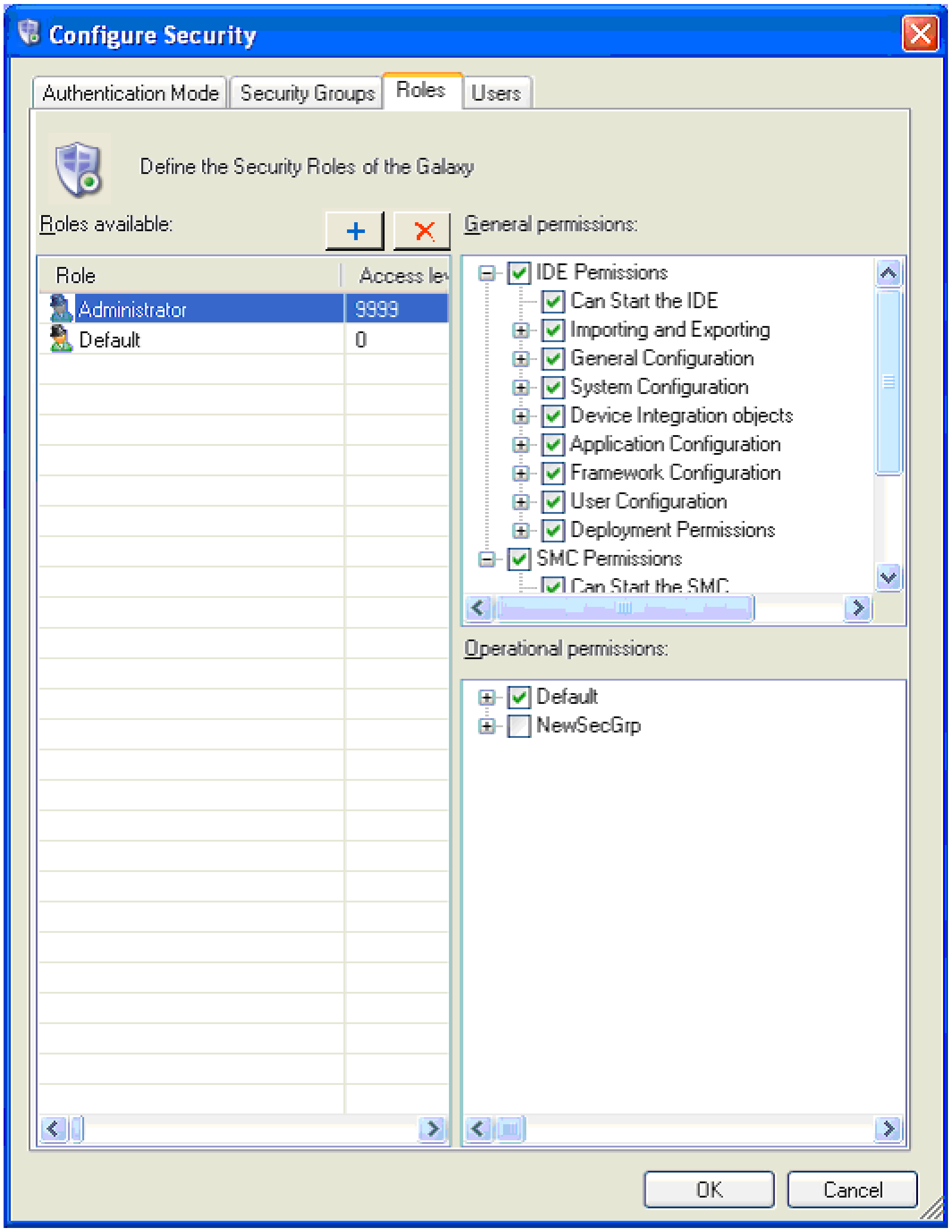
Create a new security group by clicking the Add button blue (plus sign) and then typing a name for the new group in the **Security Groups Available** pane. If you create additional security groups (for example, a supervisory group), you must select the Default or other group and then drag those objects you want to associate with the new security group.

Note Security group names are not case sensitive. You cannot create two security groups with the same name where the only distinction between them is capitalization.

Delete a security group by selecting it and clicking the Delete button (red X). Security groups cannot be deleted if AutomationObjects are associated with them. The Default security group cannot be deleted.

See Allowed Names/Characters in Introduction for more information about security group naming restrictions.

Roles



Define the Security Roles of the Galaxy

Roles available:

Role	Access lev
Administrator	9999
Default	0

General permissions:

- [✓] IDE Permissions
  - [✓] Can Start the IDE
  - [✓] Importing and Exporting
  - [✓] General Configuration
  - [✓] System Configuration
  - [✓] Device Integration objects
  - [✓] Application Configuration
  - [✓] Framework Configuration
  - [✓] User Configuration
  - [✓] Deployment Permissions
- [✓] SMC Permissions
  - [✓] Can Start the SMC

Operational permissions:

- [✓] Default
- [ ] NewSecGrp

On the **Roles** page, 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.

Click each to show the **General** and **Operational Permissions** assigned to that role. The **General Permissions** relate to application configuration and administration tasks. The **Operational Permissions** relate to the security groups displayed on the **Security Groups** page. By default, the Administrator has all permissions.

Note The Administrator role's **General Permissions** cannot be modified.

Create a new role by clicking the Add button (blue plus sign) and then typing a name for the new role in the **Roles Available** pane. If you create additional roles (for example, plant_shutdown), you must select those **General** and **Operational Permissions** the new role should have.

Delete a role by selecting it and clicking the Delete button (red X). Roles cannot be deleted if any users are associated with them. The Default and Administrator roles cannot be deleted, and the Administrator role always has full functional permissions.

See Allowed Names/Characters in Introduction for more information about role naming restrictions.

Access Level is an InTouch function. In InTouch, access levels are a schema for prioritizing runtime functions. In the Industrial Application Server security model, it only maps to InTouch values and has no inherent prioritizing characteristics. Maximum value is 9999; minimum is 0 (zero). If a user is assigned more than one role with different access levels, the higher value is passed on to InTouch.

Note Roles in bold red text are invalid with respect to the **Authentication Mode** selected.

Configuration Security (General Permissions)

The following is a list of the **General Permissions** that can be associated with a new role:

Can Start the IDE

- Importing and Exporting
  - Can Import
  - Can Utilize Galaxy Load/Galaxy Dump
  - Can Export
- General Configuration
  - Can Modify Deployed Instances (this permission serves as a superset for all types of instances, including those in the System Configuration, Device Integration Objects, and Application Configuration groups)

Important! If a role is given “Can Modify Deployed Instances” permission, ensure that appropriate “Can Create/Modify/Delete...” permissions in the System Configuration, Device Integration Objects, and Application Configuration groups are also selected so as to give the role check in and undo checkout capabilities.

- Can disable change comments
- Can override checkout
- Can upload from runtime

- System Configuration
  - Can Create/Modify/Delete System Object Templates (Platforms and Engines)
  - Can Create/Modify/Delete System Object Instances (Platforms and Engines)
  - Can Create/Modify/Delete Area Objects
- Device Integration Objects
  - Can Create/Modify/Delete Device Integration Object Templates
  - Can Create/Modify/Delete Device Integration object Instances
- Application Configuration
  - Can Create/Modify/Delete Application Object Templates
  - Can Create/Modify/Delete Application Object Instances
- Framework Configuration
  - Can Modify the Security Model
  - Can Modify Localization Settings
  - Can Modify the Time synchronization settings
- User Configuration
  - Can Create/Modify/Delete users
  - Can Modify own User Information
- Deployment Permissions
  - Can Deploy/Undeploy System Objects
  - Can Deploy/Undeploy Area Objects
  - Can Deploy/Undeploy Application Objects
  - Can Deploy/Undeploy DeviceIntegration Objects
  - Can Mark an Object as Undeployed
- Can Start the SMC
  - Can Start/Stop Engine/Platform
  - Can write to object Attributes using Object Viewer

Runtime Security (Operational Permissions)

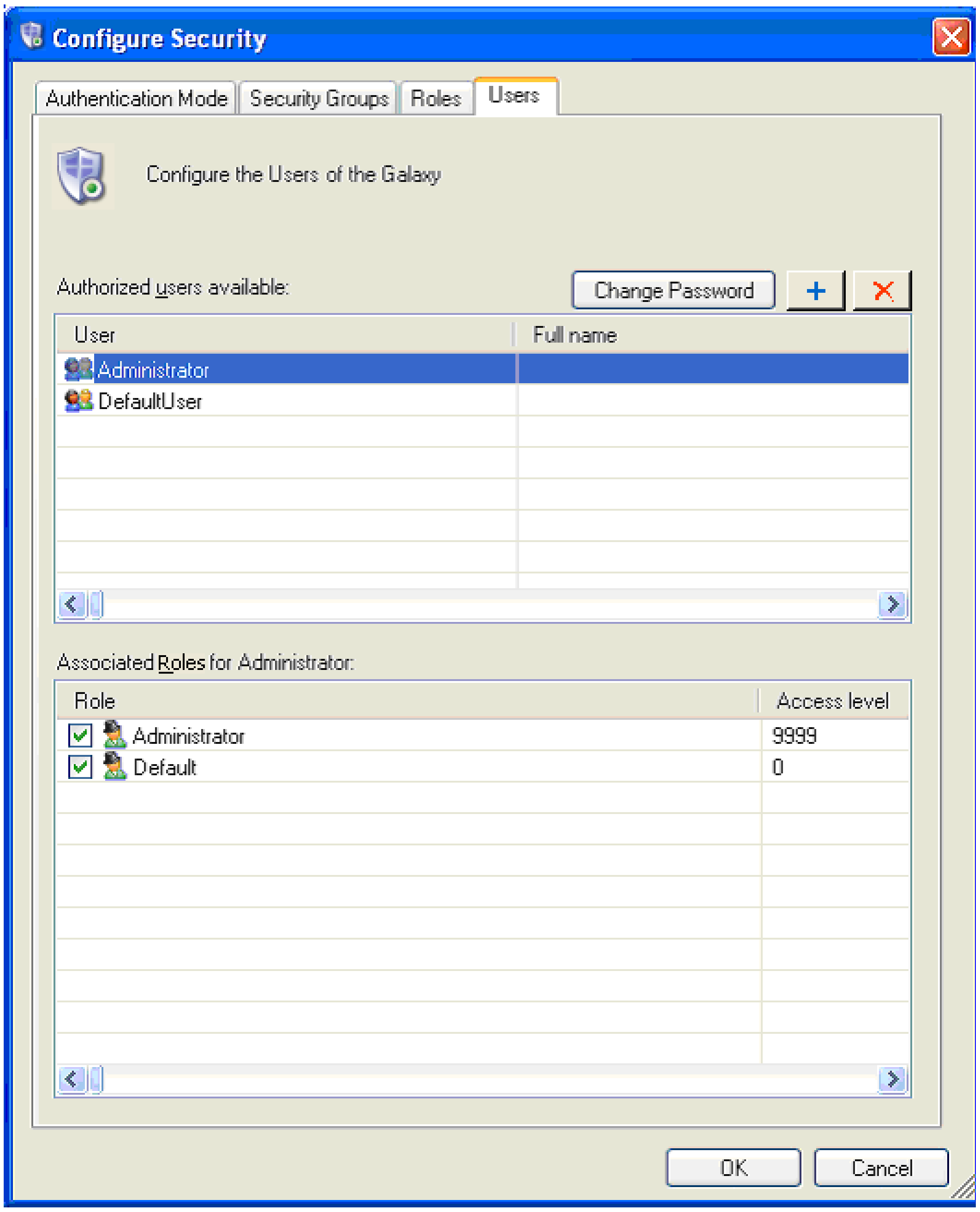
The following is a list of the **Operational Permissions** that can be associated with a role:

- 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 runtime 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 runtime environment.

If **OS Group Based** is selected on the **Authentication Mode** page, then use the browser to select from a list of domains and the user groups within the domain.

So far you associated specific objects with specific security groups, you associated specific roles with general permissions and with operational permissions based on security groups. Now, you must associate users with roles.

Users



Configure the Users of the Galaxy

Authorized users available:

User	Full name
Administrator	
DefaultUser	

Associated Roles for Administrator:

Role	Access level
✓ Administrator	9999
✓ Default	0

On the **Users** page, all users in the Galaxy and the roles they are assigned are shown. To assign a role to a user, select the user in the **Authorized Users Available** pane, and then select a role in the **Associated Roles for <user name>** pane. Users can be assigned to more than one role.

This page is available only for viewing if **OS Group Based** is selected on the **Authentication Mode** page. Users are automatically added to the **Authorized Users Available** list when they log on to the computer.

Important! If either OS based authentication mode is selected, users with local machine accounts are added to the **Authorized Users Available** list in the following format: `.\<username>`. While viewing Industrial Application Server events and alarms in InTouch, the "." appears as the user's domain. If **OS Group Based** authentication mode is selected, any such local machine 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.

Note **Users** and **Roles** in bold red text are invalid with respect to the **Authentication Mode** selected.

Add a new user by clicking the Add button (blue plus sign) and typing the user's name in the **Authorized Users Available** pane. A user's name must be unique in the Galaxy. Delete a user by selecting it and clicking the Delete button (red X).

Note You cannot delete the user who is currently logged in.

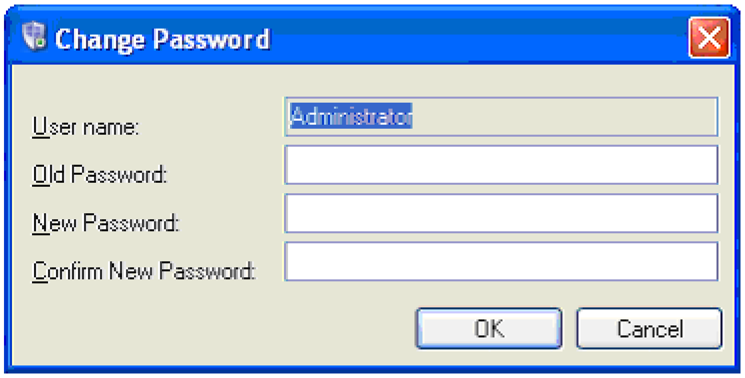
See Allowed Names/Characters in Introduction for more information about user naming restrictions.

By default, the newly-created user is associated with the Default role (this cannot be changed as every user belongs to the Default role), but not the Administrator role. Double-click in a text box to put it in edit mode. Change as appropriate.

Note All users are automatically assigned to the Default role in addition to any new roles you create and assign to them. To ensure that users assigned newly-created roles have only those unique permissions set in each role, you must change the permissions for the Default role. Otherwise, all users have the larger set of permissions given by the Default role as provided by default after product installation.

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 user's OS password.



User name: Administrator

Enter appropriate data, used in the configuration, administration and runtime environment to authenticate users.

Note The Administrator user can log in any authentication mode except when security is disabled. When logged in 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 provided and cannot be edited.

When you are finished configuring your security model, click **OK** to implement the security settings.

After You Change Security

Take note of the following behaviors and conditions that occur after you change ArchestrA security settings.

- When you change security, the IDE is restarted.
- 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 users, users’ full name and OS groups if some data in the OS has changed.

About OS Group Based Security

If you use **OS Group Based** Authentication Mode, familiarize yourself indepth with the functions of the Windows operating system, particularly its user permissions, groups and security features. ArchestrA OS Group Based security leverages those Windows features.

The following behaviors are unique to **OS Group Based** Authentication Mode:

- A newly-added user working on a computer with no access to the Galaxy 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 runtime operational permissions to do writes. To enable remote writing capabilities, log on to the remote node at least one time.
- If you log in to ArchestrA on a workstation that belongs to Domain A and Domain Controller A fails, locally cached login data is used on subsequent logins. When the domain controller returns to operation, your login fails during the time period that trusts are being reestablished by the controller. If during the controller outage, your username/password data was changed, you can use the old login data if you intend to work locally. If you want to perform remote operations, you should attempt to log in with the new login data. If that fails, the trusts are being reestablished by the controller, and you should retry at a later time.
- If you attempt to log in to ArchestrA on a workstation running Windows 2000, login fails until you properly set the TCB privilege in Local Security Policies. Do this as follows:

  On a Windows 2000 Server computer – on the **Start** menu, point to **Programs** and then **Administrative Tools**, and then click **Local Security Policies** (On a Windows 2000 Professional computer – on the **Start** menu, point to **Settings** and then click **Control Panel**. In the **Control Panel**, double-click **Administrative Tools**. In the **Administrative Tools** utility, double-click **Local Security Settings**.). In the left pane of the **Local Security Settings** dialog box, expand the Local Policies folder and click the User Rights Assignment folder. In the right pane, double-click **Act as a part of operating system**. In the **Local Security Policy Setting** dialog box, add the user (the user logged in to the computer) by selecting the **Local Policy Setting** check box, and then click **OK**. Log off and log in to the computer again to implement the new TCB privilege. You must be an administrator to set TCB privilege.
- The list of domains and user groups appears differently in the group browser depending on whether you have configured your domain as a Mixed or Native domain. Your unique appearance should map 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” rather than “Security” cannot be used for security purposes.
- 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 in to ArchestrA for the first time. If the user previously logged in to ArchestrA 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 in to ArchestrA.

C H A P T E R 1 0

Galaxy Repository Management

The Galaxy Repository is the central database through which you manage your Industrial Application Server applications. Each application is equivalent to an individual Galaxy in the Galaxy Repository.

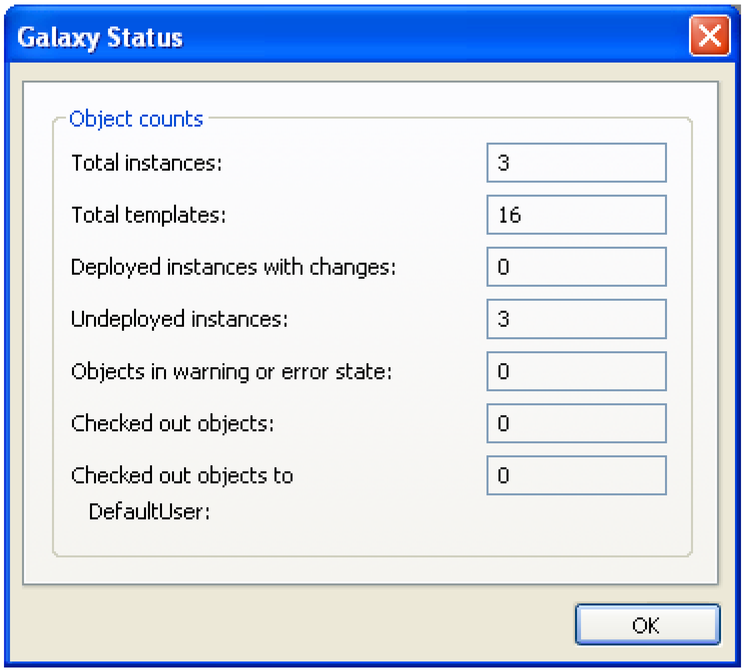
This chapter describes tools and functions for managing your Galaxy Repository and the Galaxies it contains.

Contents

Determining Galaxy Status

To determine the status of a Galaxy in your Galaxy Repository

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



Object counts

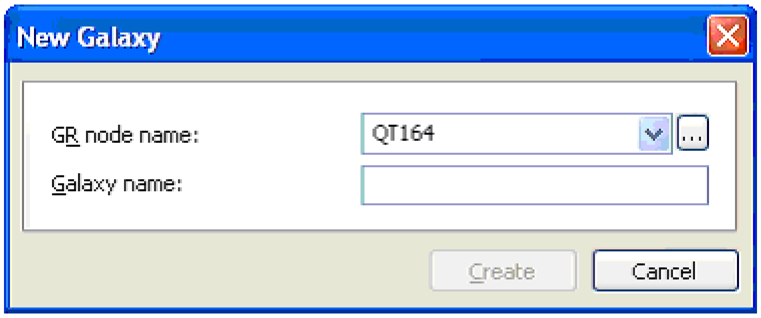
Total instances:	3
Total templates:	16
Deployed instances with changes:	0
Undeployed instances:	3
Objects in warning or error state:	0
Checked out objects:	0
Checked out objects to DefaultUser:	0

Use the information in this dialog box as a first step toward troubleshooting and generally managing the Galaxy. Of particular interest are the number of objects in warning or error state.

Creating a New Galaxy

To create a new Galaxy

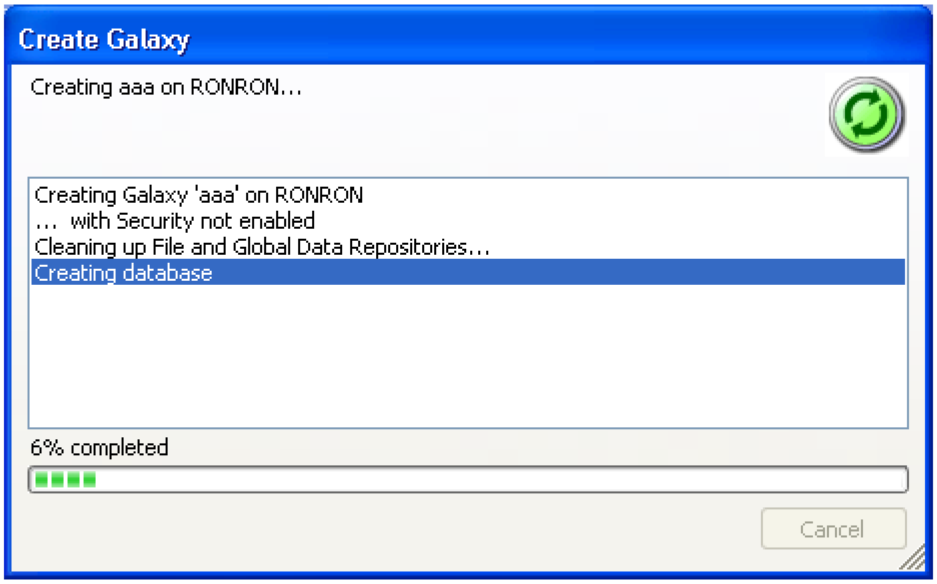
1. On the **Connect to Galaxy** dialog box, click **New Galaxy**. The **New Galaxy** dialog box appears.



GR node name: QT164

Galaxy name:

2. Select the computer where you want to create the new Galaxy from the **GR Node Name** list.
3. Type the name of Galaxy you want to create in the **Galaxy Name** box.
4. Click **Create** to begin creating the new Galaxy. The **Create Galaxy** progress box appears.



Creating aaa on RONRON...

Creating Galaxy 'aaa' on RONRON
... with Security not enabled
Cleaning up File and Global Data Repositories...
Creating database

6% completed

Note All new Galaxies are created with no security. See Working with Security for information about setting security for your new Galaxy.

Click **Close** to return to the **Connect to Galaxy** dialog box.

Multiple Galaxies on one Galaxy Repository Node

You can create and configure multiple Galaxies in a single Galaxy Repository at the same time. You can configure one Galaxy from an IDE and then change Galaxies and configure the second one from the same IDE.

However, you can only deploy one Galaxy from the Galaxy Repository at a time to the computers on your network.

You can deploy objects from one Galaxy, undeploy them and then deploy objects from a second Galaxy. But if you try to deploy objects from the second Galaxy to a computer that hosts deployed objects from the first Galaxy, the deploy function fails.

Loading/Unloading a Galaxy

Objects and their configuration data can be exported to a comma-delimited format Galaxy dump file (`.csv` extension) and then imported into a Galaxy. See “Importing a Galaxy Load File“and “Exporting a Galaxy Dump File” in “Working with Objects” for complete details about performing this function.

Uploading Runtime Configuration to Galaxy

Changes to certain attributes (Writeable_UC, Writeable_UC_Lockable, Writeable_USC, Writeable_USC_Lockable) can be made in the configuration environment and then later changed at runtime. As a result, the values of these attributes can differ between the runtime and configuration environments.

You can upload runtime configuration changes to the Galaxy database so that the attribute values in the database match those in the runtime. If the uploaded object is ever redeployed, or undeployed and then deployed, its runtime configuration changes are deployed with it.

Selected objects have not been edited and checked in since last deployment or upload. They are not in Pending Update state. The selected objects are checked in.

To upload runtime configuration to the Galaxy

1. Select the object in the **Model** or **Deployment View**. Multi-select is allowed with **Shift**+**Click** or **Ctrl**+**Click**. For example, you could select an entire hierarchy from AppEngine down.
2. On the **Object** menu, click **Upload Runtime Changes**. The runtime attributes of the selected objects are copied over those in the Galaxy database.

If you select an object that is currently checked out to you, a warning appears during runtime 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 runtime attributes are copied to the Galaxy database.

Note Performing this function on objects checked out to other users is not allowed.

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

Backup up, Restoring a Galaxy

Periodically, you should back up your Galaxy. This action provides a hedge against catastrophic computer failure or malicious attack on your Galaxy database.

Use the Galaxy Database Manager utility to do this function. The Galaxy Database Manager is part of the suite of ArchestrA System Management Console utilities. To start this utility, click `Start`, point to `Programs` and then to `Wonderware`, and then click `System Management Console`.

See Galaxy Database Manager documentation for more information about backing up your Galaxy.

Configuring a Galaxy Time Master

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.

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 this function can work. 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`

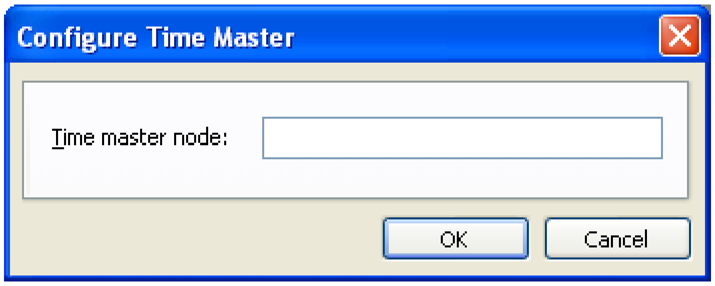
Contact Microsoft through its Product Support Services to obtain these hotfixes.

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 files that can set those Registry keys.

Important! These software/Registry updates must be completed before configuring a node as a time master.

To configure a time master node

1. Before configuring a time master, apply all hotfixes and Registry settings as specified above.
2. On the **Galaxy** menu, point to **Configure** and then click **Time Master**. The **Configure Time Master** dialog box appears.



3. Type the fully-qualified node name in the **Time Master Node** box.
4. Click **OK**.

The node clock you designate serves as the master clock for all timestamping functions. ArchestrA does not implement its own time synchronization algorithm; time synchronization is based on Microsoft's Windows Time Service. The default synchronization period is one time every 45 minutes until three successful synchronizations occur, and then one time every eight hours afterwards.

All WinPlatforms begin synchronizing the time on their node when they are deployed.

The time master node can be in another time zone. The time on each Industrial Application Server node is consistent for the time zone specified on that node.

Important! If the ArchestrA nodes in the Galaxy are already members of a Windows 2000 domain, the system administrator of that domain might have already configured time synchronization. In this case, configuring a Galaxy Time Master is unnecessary and can conflict with the existing time synchronization scheme.
Time synchronization in your Galaxy is also critical if one or more nodes run Windows XP operating system. Windows XP supports a network authentication protocol that requires that time be synchronized between two nodes to enable communication between them. This protocol causes communication between the nodes to fail if the time on the two nodes differs by a predetermined amount (for example, five minutes). Therefore, if a node in your Galaxy runs Windows XP operating system and its system time is beyond the allowed variance, ArchestrA operations, like deployment, fail.

C H A P T E R 1 1

ArchestrA Redundancy

Redundancy is the strategic duplication of critical components. The main goal is to ensure high availability of the functions those components represent.

The ArchestrA infrastructure provides two types of redundancy to ensure continued run-time operation of your applications. You can configure redundant AppEngine object pairings for computer or software failures, and you can configure redundant data acquisition communications to one or more PLCs.

A failover is the condition during which run-time operations are transitioned 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 on redundancy, refer to your *Factory Suite A^2 Deployment Guide.*

Contents

Overview

ArchestrA provides redundancy in two critical functions: the AppEngine and data acquisition. In both cases, these components must be configured for redundancy to achieve duplicated functionality. AppEngine and data acquisition redundancy are mutually exclusive.

To enable AppEngine redundancy, both an AppEngine and two WinPlatforms must be configured. To enable data acquisition redundancy, two DIObjects (data sources) and a RedundantDIObject must be configured.

If you are upgrading from a previous version of Industrial Application Server, you cannot upgrade redundant engines individually. Platforms hosting redundant engines need to be undeployed as a pair and then upgraded together.

The rest of this chapter describes:

- How to configure AppEngine/WinPlatform pairings and RedundantDIObject/DIObject data source groupings.

- How to deploy redundancy-related objects and do other operations common to ArchestrA objects.
- What happens during run-time when one component fails.

Terminology

Two sets of terms are critical to understanding the functions of redundant objects. These are described below.

During Configuration

- **Primary object**: The object intended as the main or central provider of the functionality in the run-time. For AppEngines, it is the object you enable for redundancy. For data acquisition, it is the DIObject you intend to use first as your data source in the run-time.
- **Backup object**: The object that provides the functionality of the Primary object when it fails. For AppEngines, it is the object created by the ArchestrA infrastructure when the Primary object is enabled for redundancy. For data acquisition, it is the DIObject you do not intend to use first as your data source in the run-time.

During Run-Time

- **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 passive object is the one that is waiting for a failure in the Active object’s condition 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.

The Primary/Backup and Active/Standby objects form a redundancy pair. For AppEngine pairs, only the Primary and its hierarchy of assigned ApplicationObjects must be created, configured and deployed. The Backup is handled completely by the ArchestrA infrastructure (for example, it is deployed separately from the Primary).

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 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.

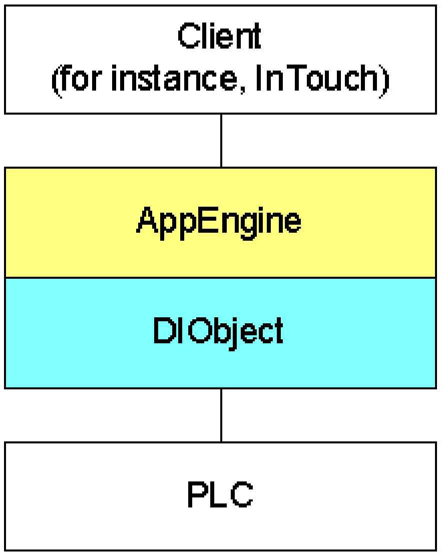
Configuration values for Primary and Backup also can be changed after you deploy the objects.

Important! In the case of AppEngine redundancy, ArchestrA 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.

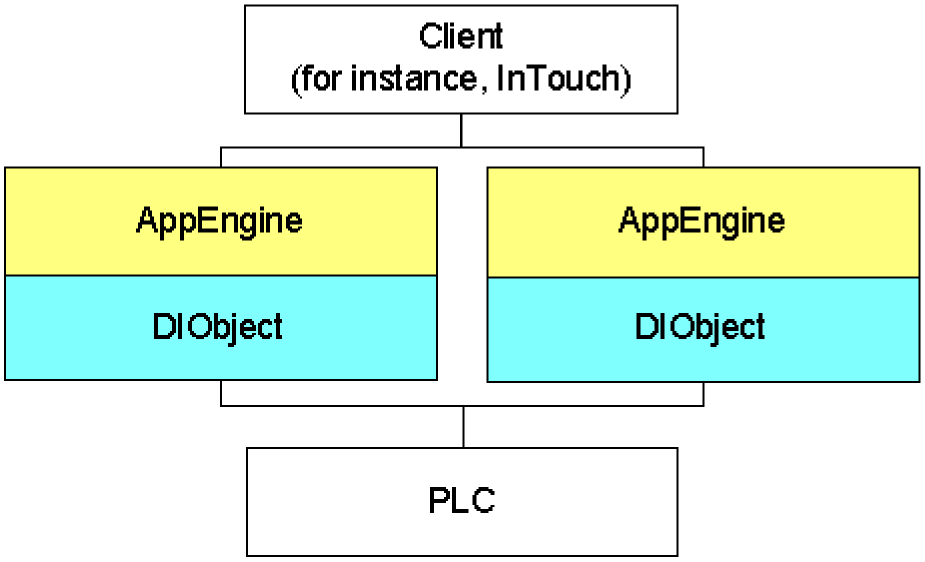
Examples of Redundancy

The following images show non-redundant and redundant combinations of AppEngine and data acquisition objects in the context of a supervisory environment.

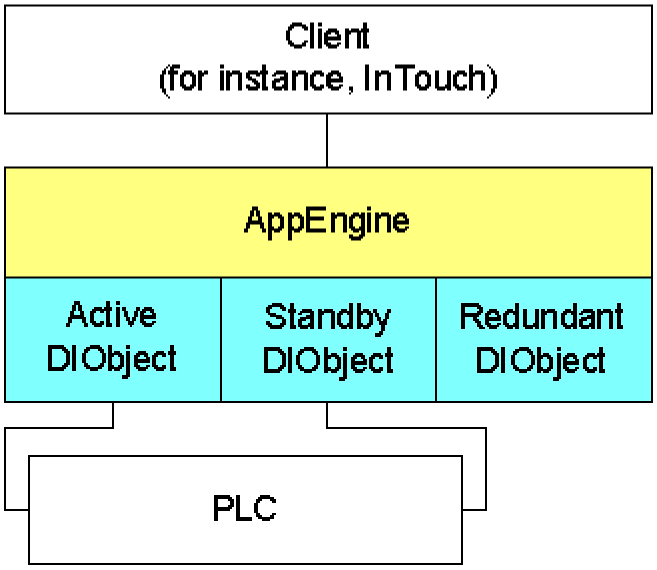
The following image shows no redundancy in either AppEngine or data acquisition. A failure in the computer hosting the AppEngine, in the AppEngine itself, or in the DIObject breaks communications between the client and PLC.



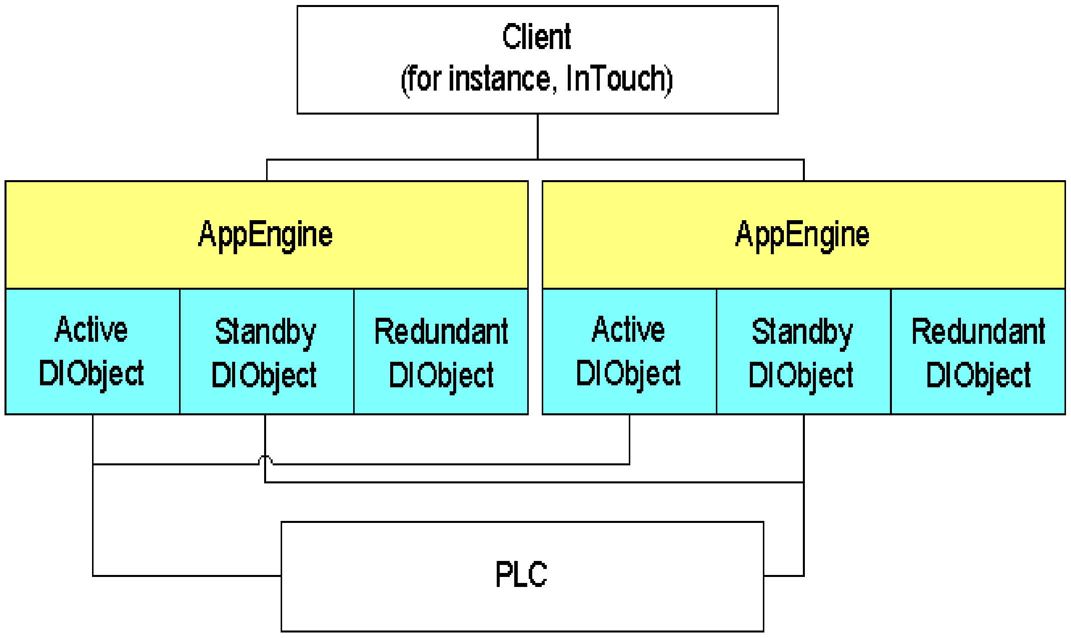
The following image shows redundancy in the AppEngine but not in data acquisition. Assume a failure in the computer hosting one AppEngine or in the AppEngine itself. Such a failure does not break communications between the client and the PLC because the second computer’s AppEngine-DIObject stack becomes Active and maintains communications.



The following image shows redundancy in data acquisition but not in the AppEngine. Assume a failure in the Active DIObject. Such a failure does not break communications between the client and the PLC because the RedundantDIObject switches the Standby DIObject to Active and maintains communications.



The following image shows redundancy in both the AppEngine and data acquisition. Assume a failure in the computer hosting one AppEngine or in the AppEngine itself. Such a failure does not break communications between the client and PLC because of the AppEngine redundancy. Likewise, assume a failure in the Active DIObject on the computer that currently hosts the Active AppEngine. Such a failure does not break communications between the client and PLC because of the DIObject redundancy.



AppEngine Redundancy

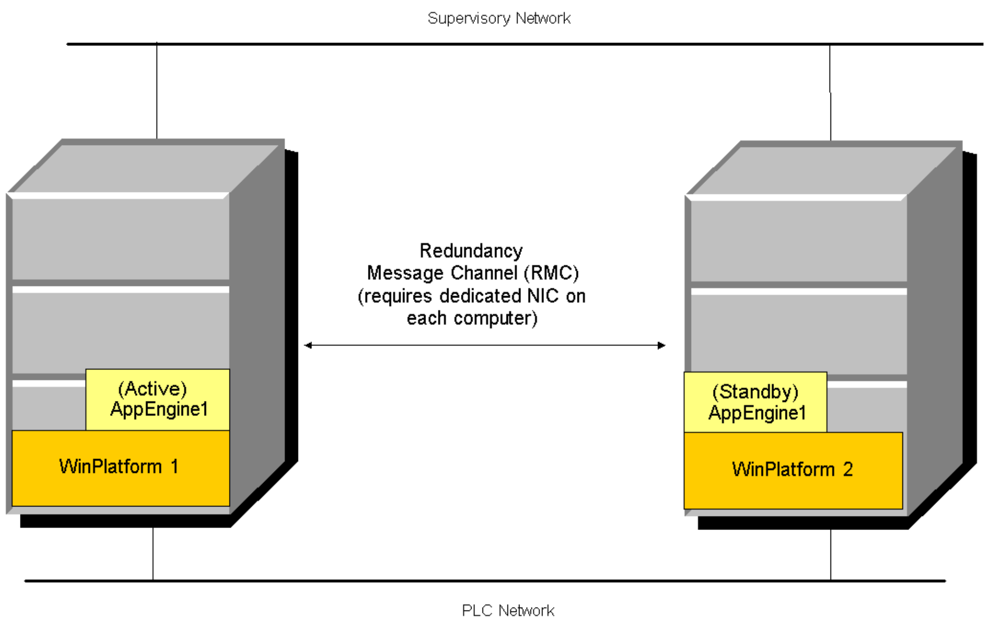
You enable AppEngine redundancy in the Primary AppEngine. You must also configure two WinPlatforms for redundancy, one to host the Primary AppEngine and one to host the Backup AppEngine.

The configuration of both WinPlatforms should be the same. At a minimum, the following configurations must be common to both WinPlatforms:

- Store-Forward directories
- Common UDAs
- Common Scripts

Note During configuration, the system allows you to assign Primary and Backup AppEngines to the same WinPlatform. You cannot deploy them that way. You must assign the Primary and Backup AppEngines to different WinPlatforms to be deployable.

Each production system computer hosting a redundancy-enabled AppEngine must have a minimum of two network cards (see image below). One NIC is for the supervisory network (and PLC network, if the computer has only two network cards) and one must be for a dedicated Ethernet crossover cable between computers for the redundancy message channel (RMC). The RMC handles redundancy monitoring, message handling and data synchronization between redundant AppEngine pairs.



Configuration

Configure redundancy in AppEngine/WinPlatforms objects in the IDE through their editors.

Important! For redundancy to function properly, WinPlatforms hosting redundancy-enabled AppEngines must be deployed to computers running the same operating system.

An AppEngine that is part of a redundancy pair has a deployment status that indicates its own status and that of its partner object. These statuses are visually indicated in the IDE Application Views. There are four deployment statuses:

- **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.

Important! See Creating Instances from Templates for a description of icon states associated with the above statuses.

Configuring Redundancy in the AppEngine

The redundancy-related parameters in the AppEngine object editor are located on the **Redundancy** and **General** tabs.

Begin the process by selecting **Enable Redundancy** on the **Redundancy** tab of the Primary AppEngine. Then, configure the remaining redundancy parameters as you wish. See the AppEngine's help file for more specific information about these parameters.

Then, on the **General** tab, set the **Engine Failure Timeout** option to 2000 milliseconds. This parameter can require additional tuning between these values (2000 ms and the default 10000 ms), depending on your application's requirements.

Note The actual engine failure timeout is three times this parameter's value. Therefore, if you set the parameter to 2000 ms (2 seconds), a failover would occur if the AppEngine failed to communicate with the computer's Bookstrap for 6 seconds. On the other end of the range, a setting of 10000 ms (10 seconds) can be too long of a wait period (30 seconds) for a well-functioning redundancy operation.

Finally, on the **General** tab, clear the **Restart the Engine When it Fails** check box.

When you save the object's configuration and check it into the Galaxy, notice that the object's icon changes from the normal AppEngine icon. At that point, the ArchestrA infrastructure created a Backup AppEngine with the same configuration as the Primary object. The Backup AppEngine is hosted by the Unassigned Host or the default WinPlatform if you have designated one in the **Configure User Information** dialog box.

Subsequently, if you change the configuration and check in the Primary AppEngine, the Backup AppEngine is checked out in the background, its configuration is updated, and then it is checked in without notification to any connected clients.

You can create a redundancy-enabled AppEngine template. You cannot lock the enabled redundancy check box in the template. When you create an instance of the template, both the Primary and Backup instances are created, in that order.

You can disable redundancy in an AppEngine after the ArchestrA infrastructure has created 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 which case the check in of the newly configured Primary object fails. To delete a Backup AppEngine from the Galaxy, it must be undeployed first.

Important! 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 then becomes the Active AppEngine. For more about failover functionality, see What Happens in Run-time.

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.

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.

Caution! Never shut down the SMC before undeploying from a Winplatform. Always undeploy the galaxy before you stop the SMC for that platform.

Configuring Redundancy in the WinPlatform

The essential redundancy-related parameters you must configure in WinPlatforms are the following: **Redundancy Message Channel IP Address**, **Redundancy Message Channel Port** and **Redundancy Primary Channel Port**. The **Redundancy Message Channel IP Address** must be the fixed IP address of a computer. See the WinPlatform help file for more information about these parameters.

Important! 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"). Do this in the **Network Connections** dialog box. Then click **Advanced Settings**, and use 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 field device connection, and the RMC), the supervisory net must be listed first and the RMC connection can be listed in any other position.

Important! You must configure the DNS settings as follows 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 Multiple NIC Computers.

Application Views

The Primary and Backup AppEngines appear in different ways in the IDE's Application Views.

The Primary AppEngine is shown throughout all views in the same way that it is when redundancy is disabled. See Working with Objects and Working with Object Editors for more information about using objects in the Application Views.

The Backup AppEngine appears only in Deployment View and it has the same object name as the Primary (with the suffix "(Backup)") but its icon is different from the Primary and it does not show any ApplicationObjects hosted by the Primary. You cannot assign ApplicationObjects to a Backup. The Backup object's editor is viewable only in read-only mode. If you select a Backup AppEngine in Deployment view and then switch to another view, its Primary AppEngine is selected automatically.

Application View Icons

When redundancy is enabled in an AppEngine, the following object icons appear:

Icon	Object
⚙	Primary AppEngine
⚙	Backup AppEngine

All object icon conditions are the same as those that apply to all other objects. See Application Views in the IDE User Interface section of the Introduction for a description of these conditions.

IDE Commands

You can perform all object-related commands on the Primary AppEngine while using the IDE user interface. A subset of those commands can be used on the Backup AppEngine. The following table shows which commands can be used on which objects.

Command	Primary AppEngine	Backup AppEngine
Assign To	Yes	Yes
Check In	Yes	No
Check Out	Yes	No
Delete	Yes	No
Deploy	Yes	Yes
Export	Yes	Yes

Command	Primary AppEngine	Backup AppEngine
Galaxy Dump	Yes	Yes
Galaxy Load	Yes	Yes
Go to Partner	Yes	Yes
Import	Yes	Yes
Object Help	Yes	Yes
Open	Yes	Yes
Open Read-Only	Yes	Yes
Override Check Out	Yes	No
Properties	Yes	Yes
Rename	Yes	No
Rename Contained Name	Yes	No
Set As Default	Yes	No
Unassign	Yes	Yes
Undeploy	Yes	Yes
Undo Check Out	Yes	No
Upload Runtime Changes	Yes	No
Validate	Yes	Yes
View in Object Viewer	Yes	Yes

The following information highlights some background details on usage of some of the commands above:

- Using the **Export** command to export a Primary or Backup AppEngine results in the redundant pair of objects being exported.
- Using the **Import** command to import a Primary or Backup AppEngine results in the redundant pair being imported. Name conflict rules for renaming objects are applied to both the Primary and Backup AppEngine being imported.
- Using the **Galaxy Dump** command results in a redundant pair of objects being dumped.
- Using the **Galaxy Load** command results in a redundant pair being loaded. Name conflict rules apply to both Primary and Backup objects.
- Using the **Upload Runtime Changes** command uploads the current Active AppEngine to the Primary object's configuration, and that uploaded configuration is then copied to the Backup AppEngine. The **Upload Runtime Changes** command is allowed when both objects are deployed or when only the Primary object is deployed.
- Using the **Rename** command on a Primary AppEngine results in the renaming of both Primary and Backup objects. This operation can be done only when both Primary and Backup objects are undeployed. You cannot rename a Backup AppEngine alone.

- When redundant AppEngines are configured, use the **Go to Partner** command in the shortcut menu to find the object's partner. Do this by right clicking either the Primary or Backup AppEngine in the Deployment view and selecting **Go to Partner**.
- Using the **View in Object Viewer** command always shows the Active AppEngine no matter which object (Primary or Backup) is selected.

Errors and Warnings

Certain requirements (for example, the order you configure an object pair for redundancy) are validated by the system infrastructure. If a requirement is not met, you can see error messages that describe the following:

- You can configure an AppEngine for redundancy before its associated WinPlatform, but if you do, you get 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, with 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 an attempt is made to deploy both engines, both the Primary and Backup fail to deploy with a message noting 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 address the same network card, you get a warning message when you save the configuration. These two parameters must address different network cards.

Deployment

Primary and Backup AppEngines can be deployed together or individually. When they are deployed together (no matter 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, ensure that network communications 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.

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
- Deployment of the WinPlatform that hosts the Primary or Backup AppEngine

See the following table for 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
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 redundancy pairs of AppEngines is similar to any regular object undeployment. The Active and Backup AppEngines can be undeployed separately. Also, they can be undeployed as a pair by selecting one of the objects in an Application View, selecting the **Undeploy** command, and selecting **Include Redundant Partner** in the **Undeploy** dialog box.

Important! Undeployment of any AutomationObjects, 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.

What Happens in Run-time

During initial deployment of a redundant pair of AppEngines, first code modules and other files for the Primary AppEngine are deployed, followed by those files for its assigned ApplicationObjects. All of these files are then 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.

AutomationObjects are always assigned to the Primary AppEngine in the configuration environment. But in the run-time, AutomationObjects are always deployed to the Active AppEngine whether or not it was initially configured as the Primary object. All files are then deployed by the Active AppEngine's WinPlatform to the Backup AppEngine as described above.

In the run-time environment, the Active and Standby AppEngines first attempt to establish communications across the RMC. This occurs when an AppEngine belonging to a redundant pair first starts up. 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. Then, if you want to move the new Standby AppEngine from its hosting computer, undeploy this AppEngine by using the **On failure mark as undeployed** option on the **Undeploy** dialog box, and then 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 has been detected. In this state, an AppEngine schedules and executes 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 have not been successfully transmitted to the Standby object, a heartbeat ping has not been received from the Standby object, or notification is received that the Standby AppEngine has 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 timeout 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) a heartbeat ping has not been received from its Active partner within a configured timeout period, 2) the Active AppEngine fails or hangs up, or 3) the Active AppEngine is shutdown on purpose. When in this state, the Standby object attempts to determine whether or not the Active object failed. If a manual failover is initiated (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 has lost communications with its partner object or it maintains communications with its partner but has 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 could complete synchronizing data. Typically, the AppEngine's partner is in one of the following states: Active-Standby not Available, Active, or Standby-Missed Heartbeats.
- **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, checks that all files required for execution are in sync with the Active engine, and 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. Once all code modules are synchronized, the AppEngine transitions to Standby-Sync'd Code state.
- **Standby - Sync'd Code**: The state of a Standby AppEngine that has successfully synchronized all code modules with the Active object.
- **Standby - Sync'd 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 temporary, transitional state when a Standby AppEngine is commanded to become Active.
- **Switching to Standby**: A temporary, transitional state when an Active AppEngine is commanded to become Standby.
- **Unknown**: The state of a redundant partner when a communication loss occurs between AppEngines or when the partner AppEngine is not running.

Important! Before rebooting a computer that hosts one of a redundant pair of AppEngines (either the Active or Backup), ensure that the Primary Network is connected. A reboot while the Primary Network is disconnected causes the Primary Network to bind to the RMC's IP address. An incorrect redundancy state occurs, indicating that redundancy functionality is good. Anytime you reboot a redundancy-enabled computer, you should check for proper network connections afterwards. For more information, see Multiple NIC Computers.

Alarm Generation

When failover conditions occur, the ArchestrA system reports alarms to the Logger. 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.

Important! Depending on the scenario that causes a failover, the Standby AppEngine can become the Active in an offscan state and alarms might not be generated. If the Active AppEngine is shutdown offscan, the checkpointer can transfer that state to the Standby, and when the latter becomes the Active, it starts up offscan. When the AppEngine is put onscan, alarms then are generated.

Alarms reported are the following:

Alarm	Previous State	Current State	Alarm Raised When	Alarm Cleared When	Alarm Reported By
Standby Not Ready [1]	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

Legend:

[1] 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 does not report alarms outstanding from the old Active AppEngine. The state of those old alarms, though, is reflected on the new Active AppEngine as follows:

- Out of alarm
- Unacknowledged
- Unacknowledged-Return to normal
- Acknowledged-Return to normal
- Acknowledged

Timestamps are also 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

Note 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. Any alarms that occur between scan cycles, therefore, are not reported to the Standby object if the Active object fails. 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 one fails.

History Generation

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.

Loss of connectivity with the historian does not cause a failover. The Active AppEngine then goes into store-forward mode and caches data every 30 seconds. Store-forward data (when the historian is unavailable) is synchronized with the Standby AppEngine.

When failover conditions occur, no more than 30 seconds of history data should be lost in the transition from Standby to Active status. This is done through a combination of store-forward operations when the historian is unavailable and normal reporting operations when it is available.

Forced Failover

You can force failover to happen. Do this through the ForceFailoverCmd attribute of the AppEngine. For example, you can link multiple conditions in a script or use the Object Viewer utility to trigger a forced failover. See the AppEngine’s object help for more information about the ForceFailoverCmd attribute.

Data Acquisition Redundancy

The RedundantDIObject monitors and controls the redundant DIObject data sources. Unlike redundant AppEngines, individual DIObject data sources do not have redundancy-related states. For all intents and 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 which are reflected in and channeled through the RedundantDIObject, which monitors the two DIObject data sources and determines which one is Active at any given time. Both data sources must also have the same item address space.

Configuration

Configure redundancy in data acquisition objects in the IDE through their editors.

Data acquisition redundancy objects (two DIObjects and the RedundantDIObject) do not have redundancy-related deployment statuses.

Configuring RedundantDIObjects

In data acquisition redundancy, you must configure all three components: a Primary DIObject data source, a Backup one, and a Redundant DIObject.

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 help that is associated with each DIObject for assistance in configuring its object editor. Also see the help associated with the RedundantDIObject for configuration assistance.

The main focus of RedundantDIObject configuration is:

- Setting the **Primary DI Source** and **Backup DI Source** on the **General** tab of the object's editor.
- Setting up the common scan groups, and block read and block write groups on the respective tabs of the object's editor.

Important! You must configure at least one scan group before you can deploy the RedundantDIObject. Also, only scan, block read, and block write groups shared by the Primary and Backup DIObjects should be configured in the RedundantDIObject.

Deployment

Deployment for data acquisition redundancy is the same as individually deploying unrelated objects. No special conditions apply to each DIObject data source and the RedundantDIObject.

See Working with Objects for more information about deploying objects.

What Happens in Run-time

The three objects in the data acquisition redundancy scheme (RedundantDIObject and its two DIObject data sources) function like any other ArchestrA object with respect to deployment, alarming and historization. They have no special redundancy-related states or restrictions with respect to deployment, undeployment and redeployment.

During run-time, the RedundantDIObject monitors the status of the DIObject data sources, and handles the switching from Active to Standby object when failure conditions arise.

Glossary of ArchestrA Terms

Terms unique to the ArchestrA environment that are used throughout the documentation are defined below.

application	A collection of objects within a Galaxy Repository that performs an automation task. Synonymous with Galaxy. There can be one or more applications within a Galaxy Repository.
ApplicationEngine (AppEngine)	A real-time engine that hosts and executes the runtime logic contained within AutomationObjects.
ApplicationObject Toolkit	A programmer’s tool used to create new ApplicationObject templates, including their configuration and run-time implementations.
ApplicationObject	An AutomationObject that represents some element of your application. This can include things such as (but not limited to) an automation process component (for example, a thermocouple, pump, motor, valve, reactor, or tank) or associated application component (for example, function block, PID loop, Sequential Function Chart, Ladder Logic program, batch phase, or SPC data sheet).
Area	A logical grouping of AutomationObjects that represents an area or unit of a plant. It groups related AutomationObjects for alarm, history, and security purposes. It is represented by an Area AutomationObject.
assignment	The designation of a host for an AutomationObject. For example, an AppEngine AutomationObject is assigned to a WinPlatform AutomationObject.
attribute	An externally accessible data item of an AutomationObject.
attribute reference string	An unambiguous text string that references an attribute of an AutomationObject.
AutomationObject	A type of object that represents permanent things in your plant (such as hardware, software, or engines) as objects with user-designated, unique names within the Galaxy. It provides a standard way to create, name, download, execute, and monitor the represented component.
base template	A root template at the top of a derivational hierarchy. Unlike other templates, a base template is not derived from another template but developed with the ApplicationObject Toolkit and imported into a Galaxy.

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 have been set. The block of data is then sent to the external data device. When the block write is complete, it indicates the completion status.
Change Log	The revision history that tracks the life cycle activities of ArchestrA, such as object creation, check-in/check-out, deployment, save, rename, undeploy, undo checkout, override checkout and assignment.
check in	IDE operation for persisting changes to an object to the Galaxy Repository and 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, making it unavailable for other users to check-out.
checkpoint	The act of saving on 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.
contained name	The name of an object as it exists within the context of its container. For example, a valve object with the TagName “Valve101” can be contained within a reactor and given the ContainedName “Inlet” within the context of its container. The ContainedName must be unique within the context of the containing object.
containment	The concept of placing one or more AutomationObjects within another AutomationObject, resulting in a collection of AutomationObjects in an organized hierarchy that matches the application model and allows for better naming and manipulation. After placed in another AutomationObject, the contained object takes on a new name in addition to its unique tagname, known as the HierarchicalName, that includes the container name and its name within the context of its container. For example, a level transmitter called TIC101 could be placed within a container object called Reactor1 and given the name Level within it, resulting in the HierarchicalName Reactor1.Level.
DAGroup	A data access group associated with DeviceIntegration objects. It defines how communications is achieved with external data sources. It can contain subscription, block read, and block write scan groups.
Data Access Server (DAServer)	The server executable that interfaces with DINetwork Objects and DIDevice Objects in the ArchestrA environment or with any third-party client, via various client protocols including OPC, DDE and SuiteLink.
Data Access Server Toolkit (DAS Toolkit)	A developer tool used to build Data Access Servers (DAServers).

DAServer Manager	The Microsoft Management Console (MMC) snap-in supplied by the DAServer that provides the required user interface for activation, configuration, and diagnosis of the DAServer.
deployment	The operation of creating an AutomationObject on its target PC. Includes installing the necessary software, the object's configuration data, and starting the object up.
derivation	The creation of a new template based on an existing template.
derived template	Any template with a parent template.
DeviceIntegration object (DIObjects)	An AutomationObject that represents the communication with external devices. DIObjects run on an AppEngine, and include DINetwork Objects and DIDevice Objects.
DIDevice Object	A representation of an actual external device (for example, a PLC or RTU) that is associated with a DINetwork Object.
DINetwork Object	A representation of a physical connection to a DIDevice Object via the Data Access Server.
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 or an acknowledgement is made).
export	The act of generating a Package file (.aaPKG file extension) from persisted data in the Galaxy Database. The resulting .aaPKG file can be imported into another Galaxy through the IDE import mechanism.
framework	The ArchestrA infrastructure consisting of a common set of services, components, and interfaces for creating and deploying AutomationObjects that collect, store, visualize, control, track, report, and analyze plant floor processes and information.
Galaxy database	The relational database containing all persistent configuration information for all objects in a Galaxy.
Galaxy Repository	The software sub-system consisting of one or more Galaxy Databases.
Galaxy	Your whole application. The complete ArchestrA system consisting of a single logical name space and a collection of WinPlatforms, AppEngines and objects. One or more networked PC's that constitute an automation system. It defines the name space that all components and objects live in and defines the common set of system level policies that all components and objects comply with.

hierarchical name	The fully qualified name of a contained object, including the container object's TagName. For example, a valve object with a contained name of "Inlet" within a reactor named "Reactor1" would have the HierarchicalName "Reactor1.Inlet". **Note** The valve object also has a unique TagName distinct from its HierarchicalName, such as Valve101.
historical storage system	The time series data storage system, used to compress and store high volumes of time series data for latter retrieval. Industrial Application Server leverages InSQL as its historical storage system.
host	An AutomationObject to which other objects are assigned (for example, a WinPlatform is a host for an AppEngine).
import	The act of adding templates or instances to the Galaxy Database from an external file.
Industrial Application Server	The name of the product inside FactorySuite that forms a central application backbone. It includes the Galaxy Repository, one IDE, a WinPlatform, an AppEngine, and a basic library of ApplicationObjects.
instance	A uniquely configured representation of an application component based on a template.
instantiation	The creation of a new object based on a corresponding template.
Integrated Development Environment (IDE)	Consists of various configuration editors that are used to configure the total system, for general framework use and for the creation of your application.
Log Viewer	A Microsoft Management Console (MMC) snap-in that provides a user interface for viewing messages reported to the LogViewer.
Message Exchange	The object-to-object messaging system.
object	Any template or instance found in a Galaxy Database. A common characteristic of all objects is they are stored as separate components in the Galaxy Repository.
off scan	The state of an AutomationObject that indicates it is idle and not ready to execute its normal runtime processing.
on scan	The state of an AutomationObject in which it is performing its normal runtime processing based on a configured schedule.

Package Definition File (.aaPDF)	The standard description file that contains the configuration data and implementation code for a base template. File extension is typically.aaPDF.
Package File (.aaPKG)	The result of the export function of the IDE. The description file that contains the configuration data and implementation code for a template and/or instance. File extension is typically .aaPKG.
PLC	Programmable logic controller.
properties	Data common to all attributes of objects, such as name, value, quality, and data type.
reference	A string that refers to an object or to data within one of its attributes.
scan group	A DAGroup that requires only the update interval defined, and the data is retrieved at the requested rate.
System Management Console (SMC)	The central runtime system administration/management user interface.
TagName	The unique name given to an instance. For example, an object might have the following TagName: V1101.
template	An object containing configuration information and optionally the code modules that can be used to create new objects, including derived templates and instances.
toolset	A named collection of templates listed together in the IDE Template ToolBox.
UserDefined object	An AutomationObject created from the $UserDefined template. This template does not have any application-specific attributes or logic. Therefore, the user must define these attributes and associated logic.
WinPlatform object	An object that represents a single computer in a Galaxy, consisting of a systemwide message exchange component, a set of basic services, the operating system, and the physical hardware. This object hosts all AppEngines.

Index