PTC®

Info*Engine® User's Guide

PTC Windchill® 10.2 M030

Contents

About This Guide

The *Info*Engine User's Guide* documents the use of Info*Engine functionality and tools. The information provided here is also available as a collection of topics in the Windchill Help Center under **Advanced Customization ▸ Info*Engine User's Guide**.

This guide assumes you have the following knowledge and skills:

- Knowledge of the existing system data structures at your site
- Knowledge of the web architecture used at your site
- Knowledge of JavaServer Page (JSP) capabilities
- Knowledge of HTML, XML, and Java
- UNIX system administration skills (if you are using a UNIX system)
- Windows system administration skills (if you are using a Windows system)

The *Info*Engine User's Guide* includes the following chapters, and is organized so that successive chapters build on earlier chapters:

Info*Engine Architecture
Introduces the Info*Engine architecture, describing how the components of the Info*Engine software work in concert to provide users with information.

Info*Engine Data Management
Describes how Info*Engine manages data, and introduces the core concepts essential to Info*Engine requests: JavaServer Pages (JSP), tasks, webjects, groups, and the virtual database (VDB).

Info*Engine JSP Pages
Provides instructions for authoring JSP pages for use with Info*Engine, as well as a more detailed description of how JSP pages interact with webjects and custom tags.

Info*Engine Tasks
Describes Info*Engine tasks, which can be used to control the retrieval and manipulation of information in Info*Engine. This chapter also includes instructions for authoring tasks and using the Info*Engine Task Editor.

Packages
Introduces the concept of packages and providing LDAP information for packages, including instructions for using the package manager.

Credentials Mapping
Describes the credentials mapping functionality, which can be used to authenticate and manage Windchill users.

Custom Tag Libraries
Discusses the potential benefits of creating a custom tag library, which is a tag library implemented using tasks and that provides a shortcut when reusing tags. This chapter includes instructional information for implementing custom tag libraries and using expression language (EL).

Custom Webjects
Provides reference information for custom webjects, which can be used to write custom Java code for use with the Info*Engine server.

Web Services Framework
Describes the various ways that web services tooling can be used with Info*Engine, and provides an instructional example for writing a C# web services client that employs the username authentication with symmetric keys mechanism.

SOAP Services
Discusses legacy SOAP services and writing tasks with SOAP, as well as how SOAP requests work within an Info*Engine Java EE connector.

Server Access Kit
Provides information for using the Info*Engine Server Access Kit (SAK) to allow Java applications to execute Info*Engine tasks, pass parameters to them, and inspect their results.

Info*Engine Custom Tag Reference
Provides reference information for the tags in the Info*Engine core, directory, and supplied tag libraries. This chapter also provides an overview of common JSP element types, including tips for using elements in Info*Engine JSP pages and tasks.

Display Webject Reference
Provides reference information for all display and image webjects associated with Info*Engine.

Task Webject Reference
Provides reference information for all administrative, group, management, messaging, and Web Event Service webjects associated with Info*Engine tasks.

Related Documentation

The following documentation may be helpful:

- Info*Engine configuration and implementation instructions are available in the *Info*Engine Implementation Guide*
- *PTC Windchill Adapter Guide*
- *JNDI Adapter Guide*
- *JDBC Adapter Guide*
- If the PTC adapter library does not contain an adapter suited to your needs, the *Info*Engine Java Adapter Development Kit Programming Reference* describes how to develop your own native Info*Engine adapters using the Java programming language.

If books are not installed on your system, see your system administrator.

Technical Support

Contact PTC Technical Support through the PTC website, or by phone, email, or fax if you encounter problems using this product or the product documentation. The PTC eSupport portal provides the resources and tools to support your PTC Windchill implementation:

https://support.ptc.com/appserver/cs/portal/

For complete details, see the *PTC Customer Support Guide*:

http://support.ptc.com/appserver/support/csguide/csguide.jsp

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not know your SCN, see "Preparing to contact TS" on the **Processes** tab of the *PTC Customer Support Guide* for information about how to locate it.

Documentation for PTC Products

You can access PTC documentation using the following resources:

- **PTC Windchill Help Center**—The PTC Windchill Help Center includes all PTC Windchill documentation. You can browse the entire documentation set, or use the search capability to perform a keyword search. To access the PTC Windchill Help Center, you can:
  - Click any help icon ? in PTC Windchill
  - Select **Help ▸ Windchill Help Center** from the **Quick Links** menu at the top right of any PTC Windchill page
  - Use the following link to access all PTC help centers:

https://support.ptc.com/appserver/cs/help/help.jsp

- **Reference Documents** website—The Reference Documents website is a library of all PTC guides:

  http://support.ptc.com/appserver/cs/doc/refdoc.jsp

  A Service Contract Number (SCN) is required to access the PTC documentation from the Reference Documents website. If you do not know your SCN, see “Preparing to contact TS” on the **Processes** tab of the *PTC Customer Support Guide* for information about how to locate it:

  http://support.ptc.com/appserver/support/csguide/csguide.jsp

When you enter a keyword in the **Search Our Knowledge** field on the PTC eSupport portal, your search results include both knowledge base articles and PDF guides.

Comments

PTC welcomes your suggestions and comments on its documentation. To submit your feedback, you can:

- Send an email to documentation@ptc.com. To help us more quickly address your concern, include the name of the PTC product and its release number with your comments. If your comments are about a specific help topic or book, include the title.
- Click the feedback icon PTC in the PTC Windchill Help Center toolbar and complete the feedback form. The title of the help topic you were viewing when you clicked the icon is automatically included with your feedback.

1

Info*Engine Architecture

The following topics provide an overview of the Info*Engine architecture.

Identifying the Info*Engine Components

The following components make up the Info*Engine architecture:

Info*Engine Servlets
Info*Engine servlets provide an interface between the web server and Info*Engine.

Info*Engine Server
The Info*Engine server provides a mechanism for retrieving and manipulating the data that users or custom applications want to view or receive.

Naming Service
The Naming Service is the software that supports the operation of Info*Engine components. In the Info*Engine Naming Service, you can identify the LDAP directory servers where entries for the network addresses of Info*Engine components and entries for configuration properties reside.

Info*Engine Service Access Kit (SAK)
The Info*Engine Service Access Kit (SAK) is an application program interface (API) that facilitates the development of Java applications, including JSP pages, that directly utilize the functions and features of Info*Engine. For example, high-level Info*Engine components such as the IE servlet and the Info*Engine server use the SAK to invoke tasks and individual webjects.

Native Adapters
The native adapters provide a direct interface between Info*Engine and information systems.

Non-Native Adapters
The non-native adapters provide an indirect interface between Info*Engine and information systems. These adapters use a different protocol from the protocol used by Info*Engine and therefore cannot connect directly to Info*Engine.

Gateways
Gateways provide an interface between Info*Engine and non-native adapters.

Info*Engine SOAP Servlets
The Info*Engine SOAP servlets catch and process Info*Engine SOAP requests that are made over the web. SOAP (Simple Object Access Protocol) is a lightweight protocol that can be used by third-party applications. By using this protocol, third-party applications can send requests to execute Info*Engine code and return the output that is generated.

The remaining sections describe the relationships among the components.

Identifying Basic Configurations

Info*Engine components can be used in many different software and hardware configurations to meet your business requirements for accessing, managing, and presenting data from different information systems.

Setting up your Info*Engine environment can be accomplished by:

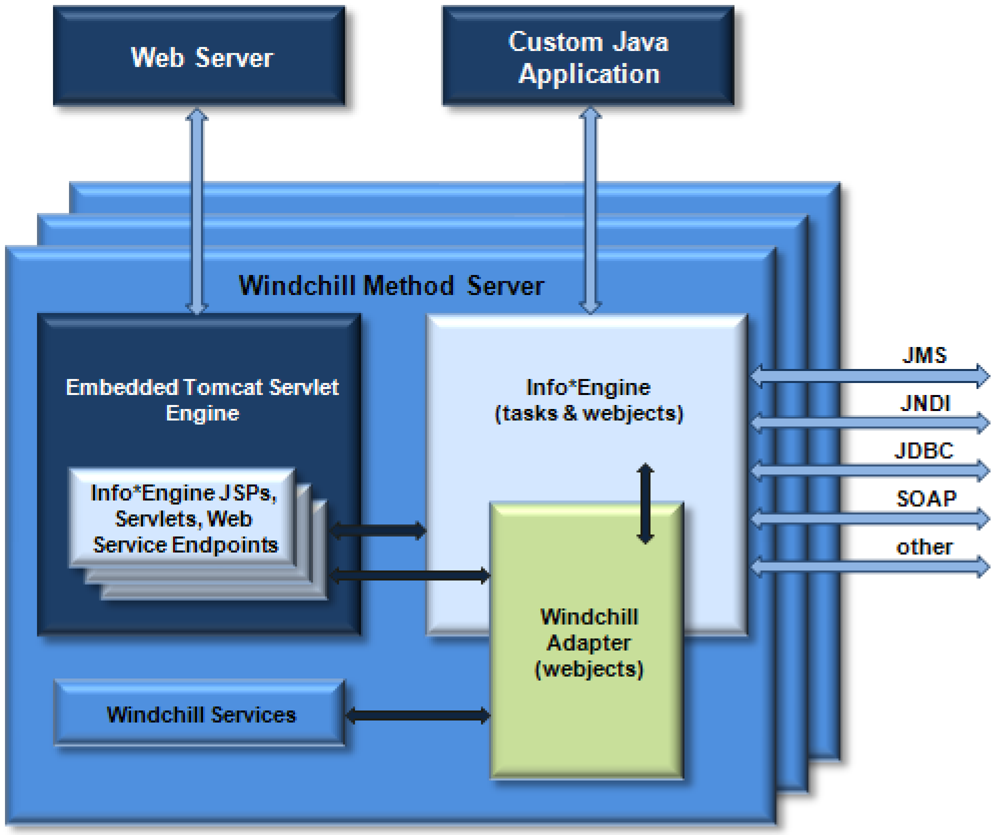
- Establishing interactions with Info*Engine.
- Managing the execution of Info*Engine tasks.
- Starting and managing Info*Engine components.
- Managing connections to the information systems where the data of interest resides.

Interacting with Info*Engine

Initiating an interaction with Info*Engine can be accomplished by using one or more of the following:

- JavaServer Pages (JSP)
- Custom external Java applications
- Simple Object Access Protocol (SOAP) web service
- Java Message Service (JMS)
- External applications processing XML over HTTP (or HTTPS)

The following diagram shows how Info*Engine components and other customer software components can interact to execute Info*Engine code:


Web Server
Custom Java Application
Windchill Method Server
Embedded Tomcat Servlet Engine
Info*Engine JSPs, Servlets, Web Service Endpoints
Info*Engine (tasks & webjects)
Windchill Adapter (webjects)
Windchill Services
JMS
JNDI
JDBC
SOAP
other

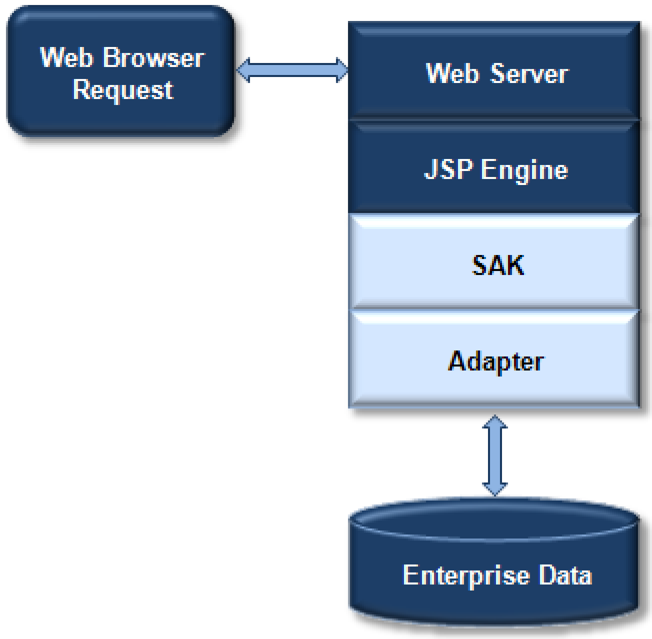
Info*Engine code consists of Java classes that are accessed through the Info*Engine API. The API is available through the SAK and externalizes predefined functions called webjects and tasks. Webjects and tasks can be easily instantiated and invoked as Java objects from a Java application or directly from other Info*Engine tasks (stored as text files).

The following sections provide more detail about how to use the Info*Engine components with your software.

JavaServer Pages

The installation process guides you through a procedure that deploys Info*Engine as a web application. Going through the installation process sets up your web server and its servlet engine to identify Info*Engine requests and pass those requests on to Info*Engine components for processing. This sets up your Info*Engine environment so that the requests from web browsers to execute JSP and HTML pages are processed correctly.

The following diagram shows the relationships among the components that process web browser requests for JSP pages:



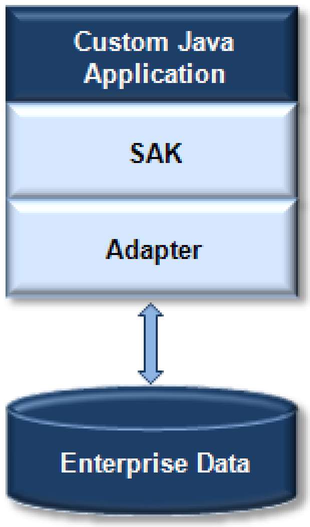
This diagram shows the components that are used when a request specifies that Info*Engine execute a JSP page. By default, Info*Engine and web server configuration specifies that JSP pages are processed in the JSP engine of the Tomcat servlet engine installed within the Windchill Method Server. The JSP engine creates an instance of the SAK, which is then used to execute the Info*Engine-specific code on the page. For example, if a user clicks a link or uses a URL in a browser window that serves as a JSP request for information from Info*Engine, the JSP engine and the SAK work together to manage the request.

The SAK processes the request and, as needed, connects to specialized Info*Engine adapters that communicate with external applications such as Oracle databases, PDM systems, various legacy systems, and ERP systems. After the requested information is obtained from the external applications, the process reverses itself and ultimately displays information in the user's browser window.

External Custom Java Applications

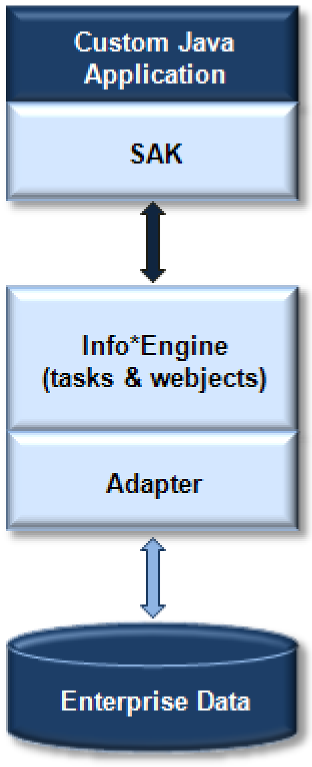
By coding a custom application in Java, you can have quick and easy access to Info*Engine without the added complexity of a web server. By using the API defined in the SAK, you can execute Info*Engine webjects, tasks, and other Info*Engine code in the Java Virtual Machine (JVM) where the application resides.

The following diagram shows the SAK and adapter classes being used in the application to access data in a remote database.



Within a Java application, you also have the flexibility of executing Info*Engine tasks that are maintained outside of the application. An Info*Engine task consists of a set of webjects and surrounding code that supports the processing of the webjects. These tasks can then be processed either in the JVM of any Info*Engine server or in the JVM of the application.

The following diagram shows the Info*Engine components that are used when an application executes a task in an Info*Engine server. In this case the application requests that a task be executed in the server that accesses data in a remote database.



Simple Object Access Protocol (SOAP) Web Service

By coding Info*Engine tasks and registering command delegates for them together under a common type identifier, you can use Info*Engine to expose a web service that can be consumed in a programming language agnostic way. The legacy RPC and SimpleTaskDispatcher servlets come pre-configured with your installation and are capable of exposing dynamic WS-I compliant web services that do not require additional deployment steps on your part. The schema these services make use of are dictated by the set of tasks comprising the web service and they are always protected by web server authentication.

Note

If your site is using form-based authentication, programmatic clients attempting to access the RPC or SimpleTaskDispatcher servlet must use the `/protocolAuth` URL prefix. For example:

```
http://<host>/Windchill/protocolAuth/servlet/
SimpleTaskDispatcher
```

```
http://<host>/Windchill/protocolAuth/servlet/RPC
```

For more information about form-based authentication, see the PTC Windchill Advanced Deployment Guide.

Alternately you can choose to generate and deploy a JAX-WS based web service that is based on a set of Info*Engine tasks and take more control over the mechanism securing that web service. It is also possible with JAX-WS based web services to write a web service purely in java code that may or may not make use of Info*Engine. In this scenario each web service is deployed as its own Java servlet.

Java Message Service (JMS)

The Info*Engine MSG and WES webjects can be used to integrate with Message Oriented Middleware using the Java Message Service (JMS) APIs. Subscriptions to JMS queues or topics can result in the execution of Info*Engine tasks within an Info*Engine Java Virtual Machine. The tasks can then interact with any of the systems connected to that Info*Engine server (including the Windchill server in which it is embedded). JMS can be a powerful way to integrate an enterprise application in a multitude of ways and can facilitate one or two way communication between Info*Engine and some other enterprise system through a Message Oriented Middleware server.

External Applications Processing XML over HTTP (or HTTPS)

Info*Engine comes pre-installed with the IE servlet that can be used to invoke Info*Engine tasks directly over HTTP or HTTPS. When tasks are invoked through the IE servlet the Info*Engine Virtual Database (VDB) is rendered as XML in the response.

Alternately, Info*Engine tasks can condition the data in the response by writing binary data in place of XML or by transforming the Info*Engine VDB to an alternate XML representation of their choosing.

By using the IE servlet simple HTTP or HTTPS clients (written in any language), an application can issue requests to send and retrieve data from Info*Engine.

Note

If your site is using form-based authentication, programmatic clients attempting to access the IE servlet must use the `/protocolAuth` URL prefix. For example:

```
http://<host>/Windchill/protocolAuth/servlet/IE
```

For more information about form-based authentication, see the PTC Windchill Advanced Deployment Guide.

Managing the Execution of Info*Engine Tasks

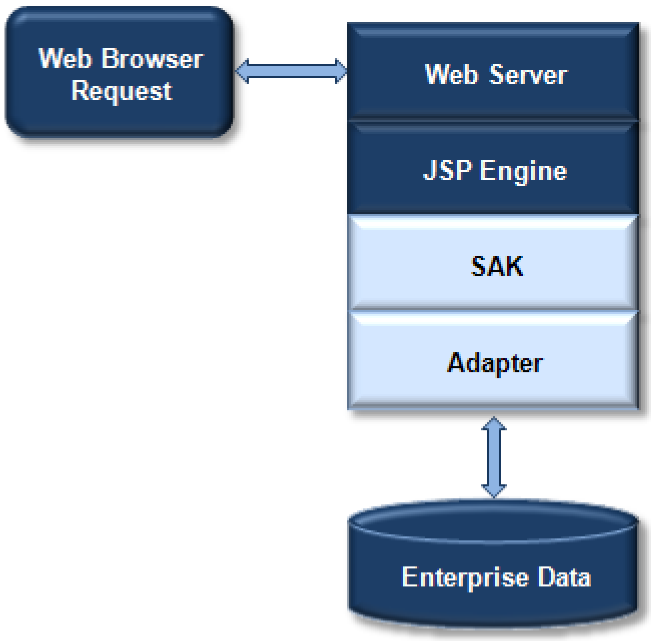
Info*Engine tasks control the retrieval and manipulation of data. Tasks consist of the following:

- Invocation of other Info*Engine tasks using nested Info*Engine tags.
- Other custom tags, which can be either supplied as part of the product or customized.
- Embedded Java source code (scriptlets or method declarations).

Info*Engine tasks are always stored as text documents within the Info*Engine task root. There are several ways to execute tasks, including the following:

- A standard HTTP request through the IE servlet, made directly from a web browser.
- Processing a JMS Topic of Queue subscription.
- An external Simple Object Access Protocol (SOAP) request.
- Using Java source code, typically either from a custom SAK application or from a Windchill workflow expression robot.

The decisions about how and where to execute Info*Engine tasks depend on your system requirements. For example, if you have a dedicated environment where one system contains both your Info*Engine application and all of the required software components, and the tasks to execute do not require any complex processing, you may choose to execute your tasks from within JSP pages that are also used to display the results. In this case, the environment used could be similar to the following:



The JSP engine depicted in the diagram instantiates an instance of the SAK within the JVM of the JSP engine. The SAK is then used to process the Info*Engine custom tags. Some of the Info*Engine tags can execute webjects that extract data from enterprise systems through an adapter, while others can display the data. In this example, all of the webjects are contained in the same JSP page.

In a more complex environment where you have a large Java application that executes complex tasks, you can manage the tasks more efficiently by separating them into individual documents, rather than coding them directly in the application. When a task is contained in its own document, it is called a standalone task. For a standalone task, the following processing options are available:

- You can specify where you want a standalone task to execute, whether it is in the same JVM as the application or in the JVM of any Info*Engine server that is part of your environment.
- You can specify how you want to execute standalone tasks that do not execute in the same JVM as the application. There are three ways to execute these standalone tasks:
    - Requesting, through a TCP/IP connection, that the task executes in a specific Info*Engine server. Each Info*Engine server listens for task requests and executes them upon arrival.

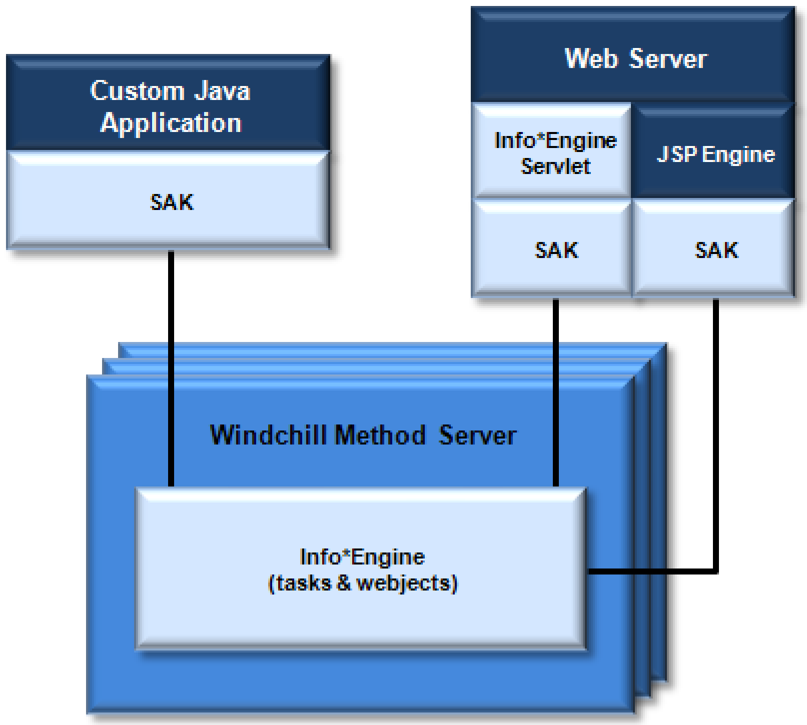
- Implementing a specific event that executes tasks. Establishing events through an Info*Engine Web Event Service allows you to execute tasks based on specific actions that can occur in your environment.
- Queuing a task for execution. After you queue a task, you can disconnect from your application. Any results are queued for later retrieval either by you or others. By queuing a task, you can also guarantee that the task is completed, even if it is interrupted due to a system problem.

By performing the basic Windchill installation, the Info*Engine server is set up to receive task requests. To use either queues or events for executing tasks, you must install and configure additional Message-Oriented Middleware (MOM) software and then update your Info*Engine configuration.

The following sections describe the architecture of the components used in executing standalone tasks.

Requesting Task Execution through a TCP/IP Connection

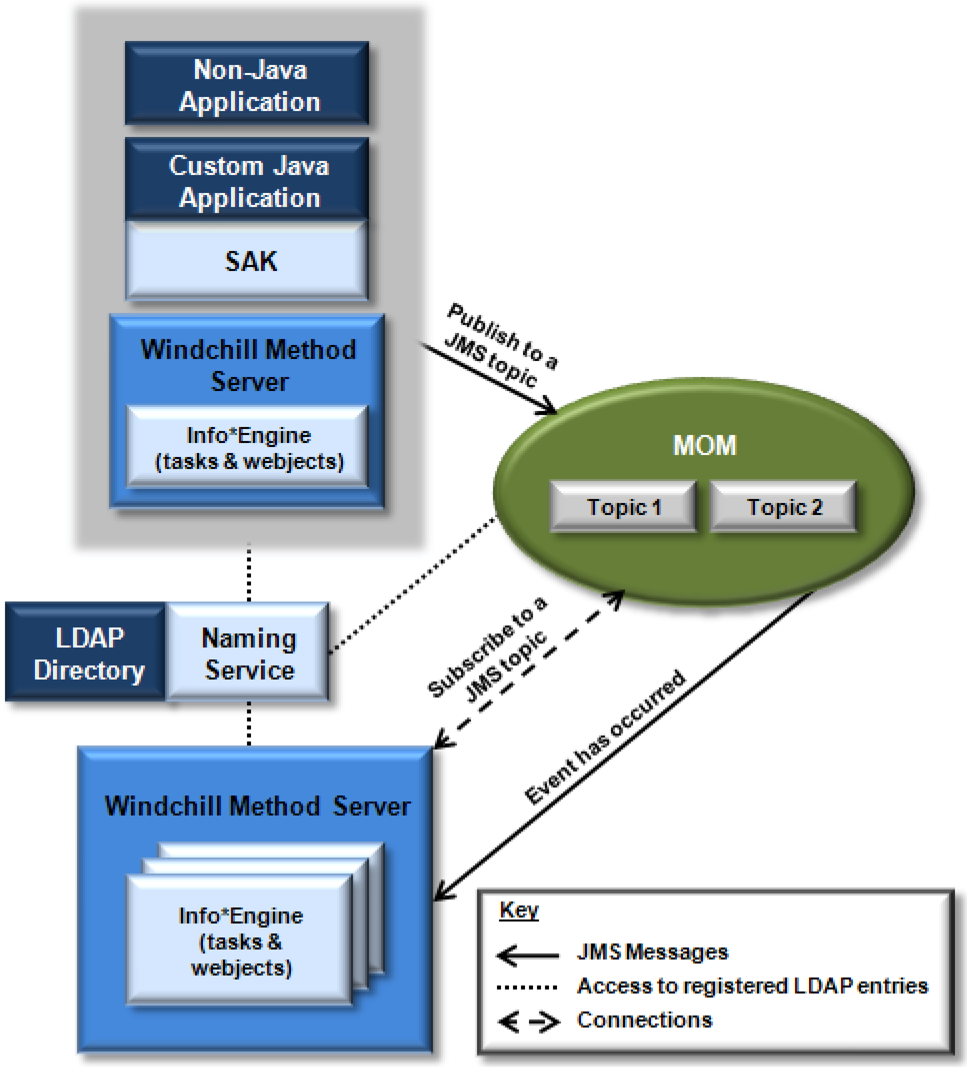
The following diagram shows the components used to request the execution of a task through a TCP/IP connection:



All of these components can be configured through the basic Windchill installation. As shown in the diagram, requests to execute standalone tasks can come from both custom Java applications and JSP pages. In addition, requests can come from SOAP requests that come from third-party applications through the Info*Engine SOAP servlets.

Implementing Events That Execute Tasks

The following diagram shows the components that can be used to implement executing tasks through Info*Engine events:



You can think of events as internet newsgroups, and subscribing to an event is like signing up for a newsgroup, where the name of the event is the name of the newsgroup. When a message is posted to a newsgroup, those who signed up receive the message. In a similar manner, when an event occurs those subscribed to the topic associated with the event receive a message indicating that the event has occurred.

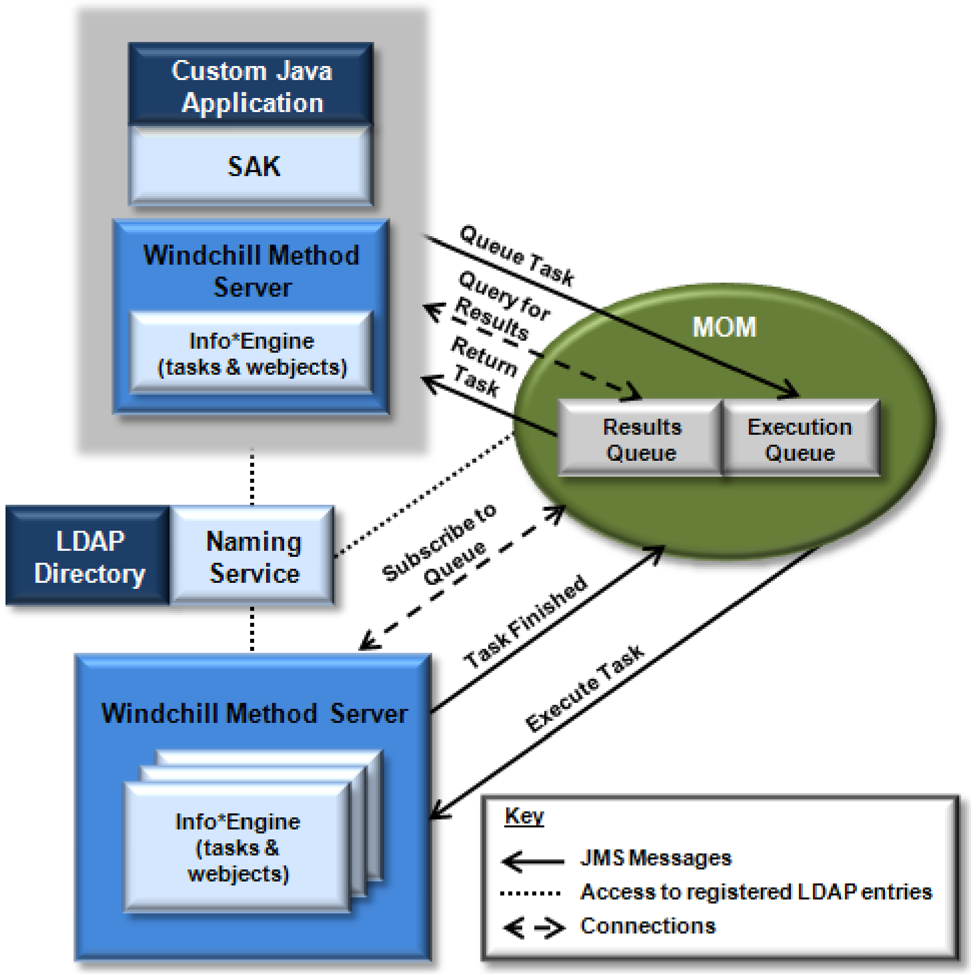
To implement executing tasks through Info*Engine events, you must install a Message-Oriented Middleware (MOM) software product such as IBM MQSeries, and you must create the required topics and a topic connection factory. You must also use the MOM utility to create managed object entries in your Naming Service LDAP directory so Info*Engine can locate the topics.

After topics and a topic connection factory have been created and registered, you can use an Info*Engine Web Event Service (WES) webject to subscribe Info*Engine servers to events. On a webject parameter, you name the task to execute when the event occurs. In the subscription process, Info*Engine builds a connection with the MOM. The MOM manages the topics and returns a message to all subscribed Info*Engine servers when the event occurs. The previous diagram shows three Info*Engine servers subscribed to a topic. Each server executes a task when the event occurs.

Info*Engine also provides a WES webject that you can use to make an event occur. The webject creates a message that adheres to the Java Message Service (JMS) specification and sends the message to a specific JMS topic, indicating that the event has occurred. You can also use any other messaging tool in your environment that can publish to the JMS topic to make the event occur. The previous diagram shows events coming from three difference sources: a non-Java application, a Java application, and a task executed from an Info*Engine server.

Queuing Tasks for Execution

The following diagram shows one possible set of components that can be used to queue tasks for execution:



This diagram shows two components (an application and an Info*Engine server) queuing tasks and one component (an Info*Engine server) executing tasks. The components used in queuing and executing tasks can easily be expanded to include multiple custom applications, Info*Engine servers, and MOMs, allowing you to scale your environment to accommodate a large number of users or tasks originating from varied locations.

To set up the queues required for queuing Info*Engine tasks, you must install a Message-Oriented Middleware (MOM) software product such as IBM MQSeries and configure the required queues and a queue connection factory. In addition, you must use the MOM utility to create managed object entries in your Naming Service LDAP directory so that Info*Engine can locate the queues.

After the queues and queue connection factory are set up and registered, you can use Info*Engine messaging (MSG) webjects to queue tasks to execute and to query for the results. When Info*Engine processes a webject that queues a task, it builds a message that adheres to the Java Message Service (JMS) specification and sends the message to the specified execution queue in the MOM.

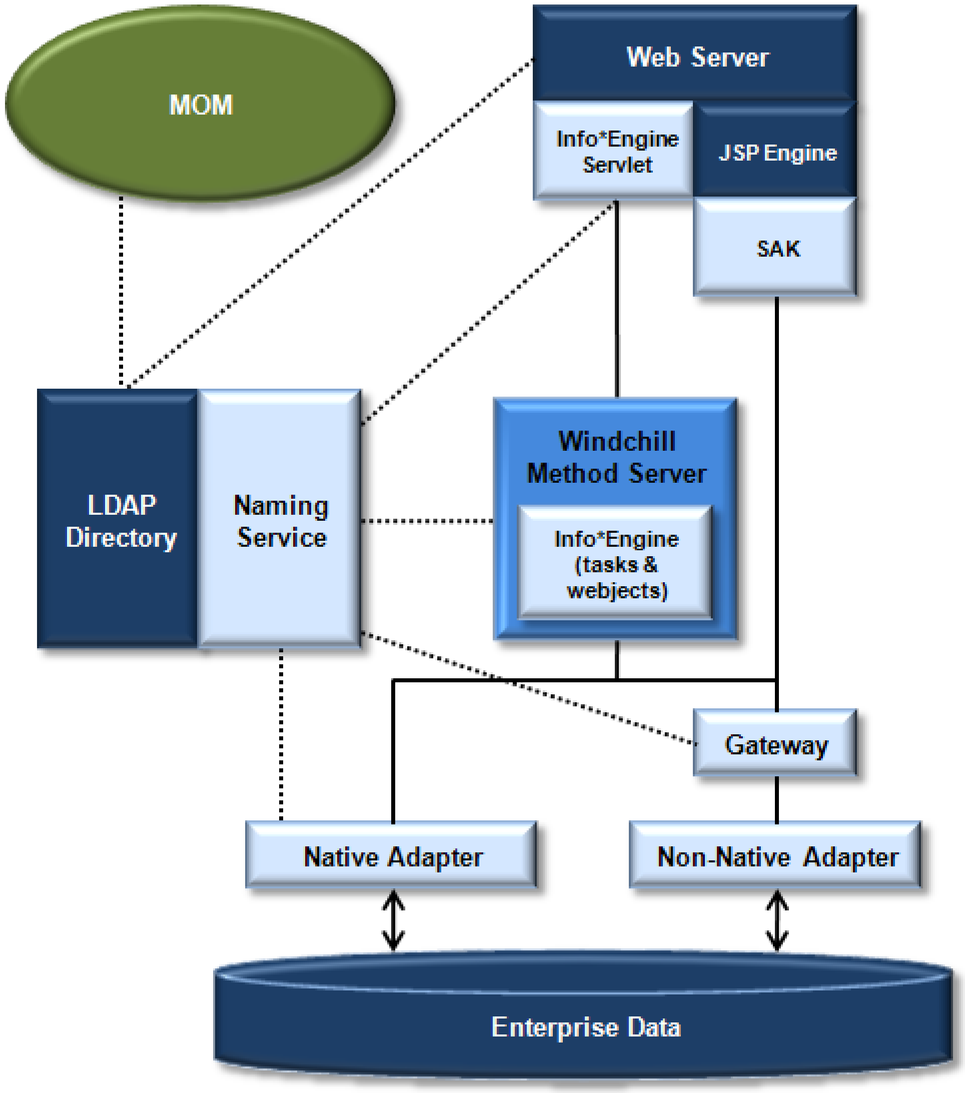
The MOM manages the queues and therefore can identify any Info*Engine server that has subscribed to the queue in order to execute tasks named in messages that are sent to the queue. If multiple Info*Engine servers have subscribed, the MOM determines which server executes the task.

After a task completes, the Info*Engine server sends a message to the results queue in the MOM. At any time, you can browse the messages in the results queue and retrieve results for tasks that have completed.

Starting and Locating Info*Engine Components

The Naming Service uses an LDAP directory to provide the IE servlet, the Info*Engine server, the native adapters, and the Info*Engine gateways with a means of locating each other, acting as a traffic director of sorts.

In the following diagram, dashed lines represent the communication between the Naming Service, Info*Engine components, and third-party software that could be installed.



Additionally, there is an Info*Engine SOAP servlet entry in the Naming Service.

The Naming Service can be used to automatically start Info*Engine components residing on the same hardware system. Depending on where you install adapters and gateways, you may want to configure the Naming Service to start them as well.

Setting Up Connections Through Adapters

Adapters provide a connection between the Info*Engine server and information systems. One side of the adapter communicates with the Info*Engine server and the other side communicates with the information system. The adapter translates Info*Engine server requests into information system requests.

Info*Engine provides two types of adapters:

- Native adapters are implemented in the Java language and conform to the formal Info*Engine interface specification. For example, the JDBC and JNDI adapters are native adapters.
- Non-native adapters are implemented in a non-Java language or do not conform to the formal Info*Engine interface specification. Because the implementation is different from Info*Engine, you must also define a gateway for each non-native adapter you install. Gateways translate Info*Engine requests so the adapters can process them. After an adapter receives a request, the adapter sends it to the associated database or data repository. The adapter also returns any information obtained from the data repository to the gateway where it is translated and passed back to the Info*Engine server.

The adapters you use are determined by the information systems from which you want to retrieve information. Info*Engine provides a unique adapter for each information system.

Native adapters can be installed as follows:

- Residing in the same Java Virtual Machine as the Info*Engine webject that accesses the adapter (known as in-process adapters).
- Distributed in their own Java Virtual Machine on the same hardware system or on remote hardware systems (known as out-of-process adapters).

How to install native adapters is determined by your site:

- Gateways usually reside in the same Java Virtual Machine as the calling webject since the code for gateways is installed as part of Info*Engine.
- Non-native adapters are always distributed in their own environment and are run as out-of-process adapters.

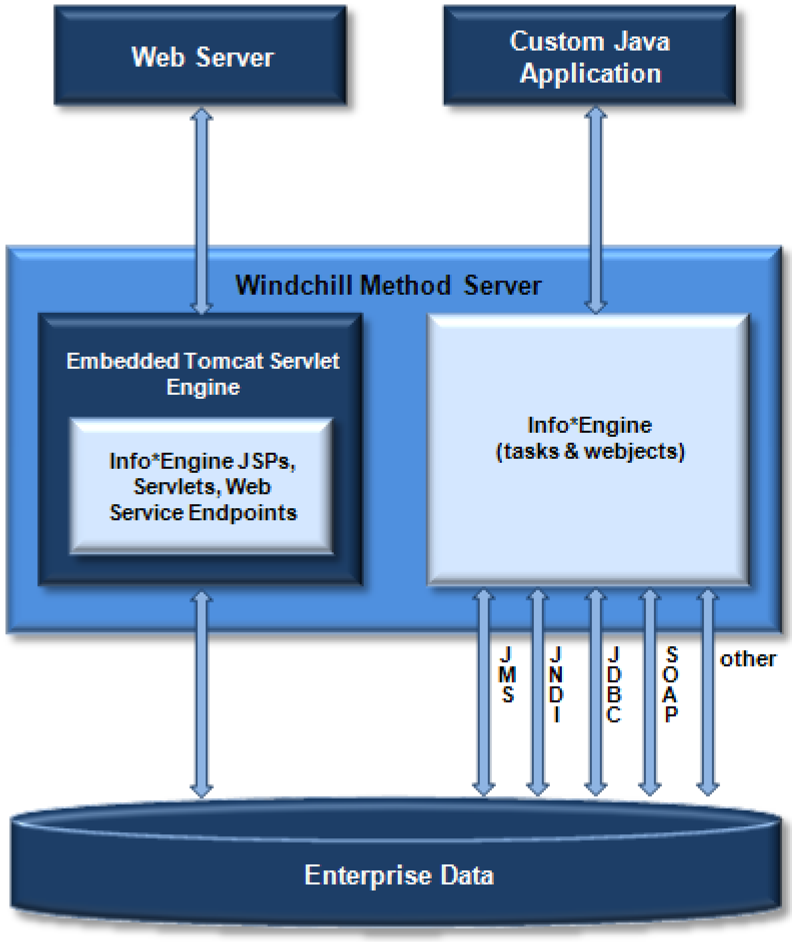
The following sections expand on the installation options.

Using In-Process Adapters and Gateways

In-process adapters and gateways are installed and run in the same Java Virtual Machine (JVM) as the calling webject. Only native adapters and gateways can be configured to run in the same JVM as the calling webject. The SAK determines which classes are required when processing webjects for an in-process adapter or gateway, and instantiates the classes in the JVM. Therefore, the communication between the webject and the adapter or gateway is very efficient.

Configuring in-process adapters and gateways minimizes communication delays and resource usage; however, the total resource usage of the machine hosting the Info*Engine code may be increased because of the additional load of running the adapter or gateway.

When an adapter is configured to be an in-process adapter, the adapter classes can be instantiated by any SAK that executes adapter webjects. The following diagram shows adapter classes residing in the JVM of a custom Java application, the web server, and the Info*Engine server:


Web Server
Custom Java Application
Windchill Method Server
Embedded Tomcat Servlet Engine
Info*Engine JSPs, Servlets, Web Service Endpoints
Info*Engine (tasks & webjects)
JMS
JNDI
JDBC
SOAP
other
Enterprise Data

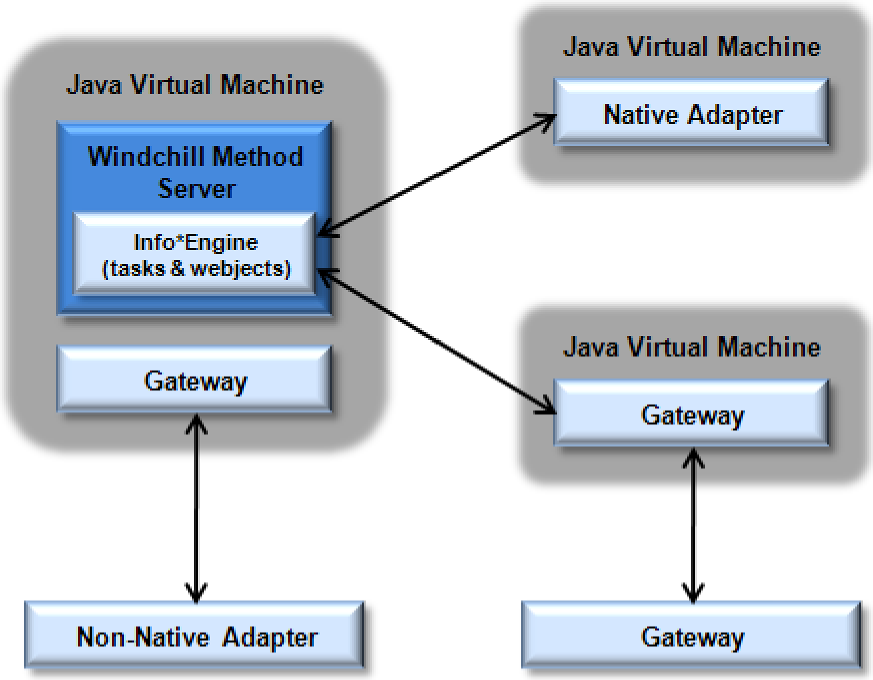
As shown in the diagram, no external communication is needed between the SAK and the adapter when the adapter is in the same process.

Running in-process native adapters and gateways is generally the preferred configuration if the resource usage on a single system is not excessive.

Using Out-of-Process Adapters and Gateways

Distributing adapters across multiple hardware systems reduces the overall resource usage on the machine hosting the Info*Engine code; however, it does introduce some delay and resource usage associated with using a TCP/IP connection for communicating between Info*Engine components and each adapter.

The following diagram shows the communication lines that are used when three adapters and one gateway are distributed.



Distributed native adapters and gateways are installed and run in their own Java Virtual Machine (JVM). These virtual machines can be on the same hardware system as the Info*Engine server or on a different hardware system. Non-native adapters can only be configured as out-of-process adapters, and they always run as separate processes. Although gateways for non-native adapters are typically configured as in-process gateways to minimize the communication delays, they do not need to be in the same process.

The deployment of distributed adapters at your site may be determined by a company policy that requires the adapter to be located near the application it accesses, or it may be based on administrative reasons. One reason for running a native adapter in its own Java Virtual Machine could be to better manage the resource usage of the virtual machine.

2

Info*Engine Data Management

The following topics introduce the concepts of JSP pages and Info*Engine tasks, and briefly explains how these document types use webjects to influence how information appears and behaves. These topics also discuss virtual databases (VDB) and their relationship to JSP pages, tasks, and context groups.

About Info*Engine Data Management

All basic Info*Engine solutions take advantage of five fundamental concepts: JSP pages, tasks, webjects, groups, and the virtual database (VDB). Both JSP pages and tasks are text-based documents that define how Info*Engine either displays or retrieves information. Webjects are the means by which Info*Engine JSP pages and tasks gather, manipulate, and display data. Groups are the chunks of information generated and manipulated by JSP pages and tasks. The virtual database is the special holding area where groups are stored until they are manipulated or passed along by JSP pages and tasks.

Note

In earlier releases, Info*Engine HTML templates provided the basic method for displaying information. Although templates are still supported, the use of JSP pages (or custom Java applications) is now the recommended method for displaying information.

JSP Pages and Tasks

Text-based documents called Info*Engine JavaServer Pages (JSP) and Info*Engine standalone tasks control the information Info*Engine uses for everything it does.

The technology used in Info*Engine JSP pages implements the JSP specification, which is the product of industry-wide collaboration with industry leaders in the enterprise software and tools markets. Using this JSP technology allows you to rapidly develop and easily maintain information-rich, dynamic web pages that leverage existing business systems. As part of the Java family, JSP technology enables the development of web-based applications that are platform independent.

JSP technology uses XML-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page. Info*Engine JSP pages typically contain:

- Static HTML (Hypertext Markup Language) and XML (Extensible Markup Language) components.
- JSP tags (such as expressions, declarations, and directives) and Info*Engine custom tags.
- Optionally, snippets of code written in the Java programming language called scriptlets.

Consequently, you can create and maintain Info*Engine JSP pages by conventional HTML/XML tools.

Info*Engine standalone tasks are XML-based documents that provide you with a way to control the retrieval and manipulation of data outside of your JSP pages or application. The main difference between JSP pages and tasks is that JSP pages can contain content that generates data and then displays that data back to the user. Tasks generate the data, but they do not provide a way to display the data. The Info*Engine task compiler recognizes static XML components, JSP tags and scriptlets, and Info*Engine custom tags. So generally, your Info*Engine task can be formatted just like your JSP pages, but without any display elements such as HTML tags.

Info*Engine provides custom tags that encapsulate recurring functionality so that the same functionality can be reused in multiple Info*Engine JSP pages or Info*Engine standalone tasks. Using Info*Engine custom tags reduces the necessity to embed large amounts of Java code in JSP pages and in standalone tasks, and allows you to quickly create the pages and tasks that are required for your application. For example, the **webject** tag allows you to easily execute the Info*Engine webjects (described in the next section) that all Info*Engine documents use.

Both JSP pages and tasks use JSP specific syntaxes to execute their duties. This guide does not describe everything you need to know about writing JSP or XML documents. Instead, it describes the rules and enhancements of the markup language that are important to understand when you are writing Info*Engine JSP pages and tasks, and it documents the Info*Engine custom tags. For now, understanding the basic appearance of JSP pages and tasks, as well as how webjects, groups, and how the VDB works with them should give you a good idea of how Info*Engine manages information. The actual Info*Engine JSP and task rules are described later in this guide.

Webjects

A webject is a command that executes a specific Info*Engine feature that has been provided through an external webject name. To include a webject in your JSP page or task, use the Info*Engine webject and parameter custom tags. Using custom tags requires no knowledge of programming beyond basic JSP coding techniques. The Info*Engine custom tags are used in the same manner as other JSP tags. They help determine how information appears within a web page or define information that can be manipulated by a program. Webjects can be added to any JSP page or task and can be used to dynamically organize and manipulate information.

Think of webjects as specialized objects that distill complicated programming into abstract or condensed terms. These specialized objects can be reused without requiring changes by a programmer, and each time the object is used, it can return different results. While understanding how to write webjects is important to the success of an Info*Engine solution, for now simply understanding the various types of webjects should help you understand how Info*Engine manages data.

The following types of webjects are available:

Display

Display webjects transform Info*Engine groups into HTML for display.

Display webjects can only be specified in JSP pages (or a custom application). In a display webject, you name an existing group that Info*Engine then codes for display. For more information, see Display Webjects for HTML on page 300.

Image

Image webjects display JPEG images based on group data. The data is displayed in a graph or a chart. For more information, see Image Webjects for JPEG on page 344.

Query

Query webjects search external databases for objects that match specified criteria. Each adapter supports a unique set of query webjects because the adapter must handle queries differently due to the nature of the underlying data repositories.

When Info*Engine encounters a query webject, it passes the webject to the appropriate adapter. The adapter then performs the specified query. The group of objects returned by a successful query is stored by Info*Engine. Thus, making the results available to other webjects. For more information, see Task Webject Reference on page 357.

Administrative

Administrative webjects perform specific administrative functions, such as gathering simple statistics, or causing service properties to reload at runtime. For more information, see Administrative Webjects on page 358

Action

Action webjects perform actions such as creating, copying, and updating. Each adapter supports a unique set of action webjects because the actions relate directly to the data repository to which the adapter connects.

When Info*Engine encounters an action webject, it passes the webject to the appropriate adapter. The adapter then performs the specified actions. For more information, see Task Webject Reference on page 357

Group

Group webjects compare, combine, or sort one or more existing groups of data that have been generated as a result of other query, action, or group webjects. For more information, see Group Webjects on page 360.

Management

Management webjects provide some common functions, such as getting properties, mapping credentials, and throwing exceptions that can be useful in managing your JSP pages or tasks. For more information, see Management Webjects on page 416

Message

Message webjects provide a set of webjects that can be used in conjunction with a third-party MOM for generic messaging functions and task queuing functions. For more information, see Message Webjects on page 440

Web Event Service

Web Event Service webjects provides a set of webjects that can be used in conjunction with a third-party MOM (or JMS service) for handling Info*Engine events. For more information, see Web Event Service Webjects on page 486

Query, administrative, action, group, management, message, and Web Event Service webjects can be specified in either JSP pages or in standalone tasks, and are often referred to as "task" webjects.

You can also create external custom webjects by writing custom Java code. With custom webjects, you have access to all internal classes of Info*Engine and can extend the functionality of Info*Engine. External custom webjects are dynamically loaded by Info*Engine and can be available to any task or JSP page.

Groups of Objects and the Virtual Database

When a webject returns information from a data management system or from some other source, the information is returned as a group of objects. Info*Engine stores each group of objects in a special holding area called the virtual database (VDB).

Groups of objects are created when query, action, group, management, message, and Web Event Service webjects perform their function. Several webjects can be contained within a task or JSP page, each creating its own group of objects to be placed within the VDB. In defining a webject, you name the output group using the GROUP_OUT parameter. It is important to name each group of objects uniquely so it can be identified by a subsequent webject when manipulating or displaying group data.

The VDB does not allow groups to have duplicate names. If a task generates two groups with the same name and both are placed in the VDB, the second group overwrites the first. If any group is unnamed and a previous group was unnamed, the new unnamed group overwrites the previous unnamed group.

Example VDB

Objects within a particular group in the VDB can be thought of as a table of data. Although group data is not stored as an actual table, it can help to understand groups within the VDB using a table such as the one below:

ename	phone	department	title
Law, Gracie	873-2200	Sales	Sales Manager
Burton, Jack	873-2302	Sales	Sales Representative
LoPan, David	873-3313	Sales	Administrative Assistant

This example table contains a heading row and three rows of data. You can think of each data row as an object in the VDB group. Therefore, in this example there are three objects in the group. Each object consists of attributes which are represented by the columns in the table. The heading row names the attributes. In this example the attributes of each object are "ename" (for the Employee Name), "phone," "department," and "title."

In addition to general data groups like the one described in this example, Info*Engine maintains several context groups whose attributes and values are available to all webjects. The context groups contain special information such as the HTTP protocol and HTML form data attributes and values that are retrieved by Info*Engine. For more information, see Context Groups on page 43.

Basic Group Creation and Display

Groups of objects are stored in the VDB so that other webjects can make use of the information. The following scenario demonstrates how you can use the VDB to pass information from a task webject to a display webject in order to perform a database query and return the results to the browser.

In this scenario, assume that the following sequence of events comprise the actions required in the scenario:

1. From the browser, a user enters a URL that identifies an Info*Engine JSP page.
2. A task webject on the page executes a query to a database.
3. The output from the task webject is formatted by a display webject and is returned to the browser.

To accomplish this scenario, start by authoring a JSP page that queries a database for information about employees. The Query-Objects webject on the page might look like this:

```
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
  <ie:param name="CLASS"     data="salesemp"/>
  <ie:param name="WHERE"     data="()"/>
  <ie:param name="SORTED"    data="ASC"/>
  <ie:param name="SORTBY"    data="ename"/>
  <ie:param name="GROUP_OUT" data="employees"/>
</ie:webject>
```

This webject executes a query of the `salesemp` table using the adapter identified in the INSTANCE parameter. The CLASS parameter defines which table should be queried, and because the WHERE parameter does not limit what is returned, all of the data in the table is returned. The SORTED parameter has a value of ASC, which sorts the data returned from the database in ascending order. The SORTBY parameter names the attribute on which sorting is done. In this scenario, the `ename` attribute set in the SORTBY parameter holds the employee name. Therefore, rather than sorting the information based on the first attribute of each item it finds, sorting is based on the employee names.

After all of the information is gathered from the query, the webject places the group named “employees” in the VDB. This group is then available for use by other webjects. If the query returns the same data as described in the earlier VDB example, you can think of the “employees” group in terms of the following table:

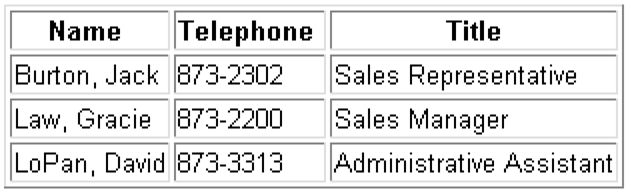
ename	phone	department	title
Burton, Jack	873-2302	Sales	Sales Representative
Law, Gracie	873-2200	Sales	Sales Manager
LoPan, David	873-3313	Sales	Administrative Assistant

Notice that the “ename” column, which contains the Employee Name data, is sorted in ascending order in the VDB.

To view the employee information that is stored in the “employees” group, you can add a display webject to the JSP page. In the display webject, you specify which group to format as the input group. By default, Info*Engine surrounds the data in the group with HTML so it can be displayed. A display webject that is designed to display the information as a table might look like this:

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"  data="employees"/>
  <ie:param name="BORDER"    data="1"/>
  <ie:param name="ATTRIBUTE" data="ename,phone,title"
                                  delim=","/>
  <ie:param name="HEADER"    data="Name,Telephone,Title"
                               delim=","/>
</ie:webject>
```

The Display-Table webject names three of the four attributes stored in the VDB and specifies header values for the attributes in the ATTRIBUTE and HEADER parameters. The resulting display is the following table, which includes the three attributes for each object under their corresponding header values:



Name	Telephone	Title
Burton, Jack	873-2302	Sales Representative
Law, Gracie	873-2200	Sales Manager
LoPan, David	873-3313	Administrative Assistant

Managing and Manipulating Groups with Info*Engine

Understanding how Info*Engine operates with more than just basic queries and displays of those queries can help you use Info*Engine to develop solutions at your site. The following scenarios can give you a clearer understanding of how you can manipulate data using Info*Engine.

Manipulating Several Groups and Displaying the Resulting Group

Expanding on the employee example described previously, this scenario demonstrates how to obtain a single list of employees from two distinct departments. The following example combines the employees from the sales department and the development department of your company. Assume that each department has their employee lists in different locations of a database at the company. This scenario gets employee information from both locations and combines it in such a way that it looks like one list of employees.

First, author a JSP page to query a database for information about employees. You can start with the task webject similar to the one shown previously in Basic Group Creation and Display on page 36. This task webject looks for those employees defined in the “salesemp” table. Now, you can add another webject that queries for information from the “devemp” table. For example, the page could contain query webjects similar to the following:

```
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
  <ie:param name="CLASS"     data="salesemp"/>
  <ie:param name="WHERE"     data="()"/>
  <ie:param name="GROUP_OUT" data="sales"/>
</ie:webject>

<ie:webject name="Query-Objects" type="OBJ">
```

```
    <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
    <ie:param name="CLASS"     data="devemp"/>
    <ie:param name="WHERE"     data="()"/>
    <ie:param name="GROUP_OUT" data="development"/>
  </ie:webject>
```

The first Query-Objects webject asks for information about employees from the sales employee database table (using the “salesemp” table name in the CLASS parameter). Specifically, it asks for information about all of the employees in the sales department. The results of the query are then placed in the VDB in a group called “sales”. You can think of the sales group in terms of the following table:

ename	phone	department	title
Burton, Jack	873-2302	Sales	Sales Representative
Law, Gracie	873-2200	Sales	Sales Manager
LoPan, David	873-3313	Sales	Administrative Assistant

The second Query-Objects webject also asks for information from the database, but this time the information comes from the development employee table (using the “devemp” table name in the CLASS parameter). The results of this second query are then placed in the VDB in a group called “development”. You can think of the development group in terms of the following table:

ename	phone	department	title
Anderson, Pat	873-2428	Development	Engineer
Stein, Chris	873-2608	Development	Manager
Wong, May	873-2741	Development	Engineer

Now there are two groups in the VDB, one called “sales” and one called “development”. The scenario is not yet complete because we want to combine the two groups into one list of employees. To combine the groups, you can add the following Merge-Groups webject to the JSP page:

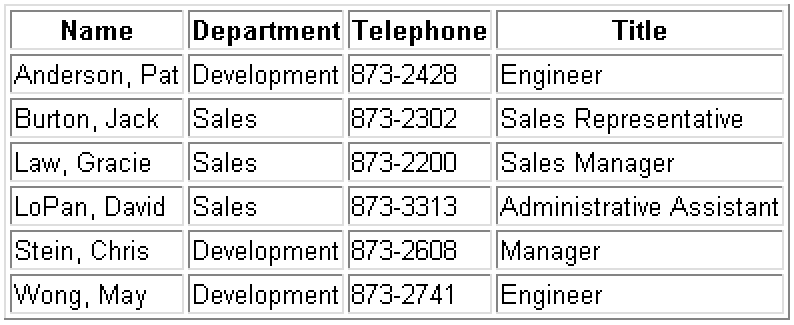
```
  <ie:webject name="Merge-Groups" type="GRP">
    <ie:param name="GROUP_IN"  data="sales"/>
    <ie:param name="GROUP_IN"  data="development"/>
    <ie:param name="SORTED"    data="ASC"/>
    <ie:param name="SORTBY"    data="ename"/>
    <ie:param name="GROUP_OUT" data="employees"/>
  </ie:webject>
```

This webject takes the sales and development groups and merges the information into one large group using the employee name from the “ename” attribute as a guide. All of the information is sorted in ascending order based on the employee name. The results of the merged information are then placed in a third group called “employees”. This final group is then placed in the VDB for later use.

To view the employee information that is stored in the employees group, you can use a display webject that is similar to the scenario used in the Basic Group Creation and Display on page 36:

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"  data="employees"/>
  <ie:param name="BORDER"    data="1"/>
  <ie:param name="ATTRIBUTE"
            data="ename,department,phone,title" delim=","/>
  <ie:param name="HEADER"
            data="Name,Department,Telephone,Title" delim=","/>
</ie:webject>
```

The resulting display is the following table that contains the employees from both the sales and development departments. The display includes the department attribute, which had been left out in the earlier scenario:



Name	Department	Telephone	Title
Anderson, Pat	Development	873-2428	Engineer
Burton, Jack	Sales	873-2302	Sales Representative
Law, Gracie	Sales	873-2200	Sales Manager
LoPan, David	Sales	873-3313	Administrative Assistant
Stein, Chris	Development	873-2608	Manager
Wong, May	Development	873-2741	Engineer

Manipulating and Displaying Multiple Groups in the VDB

When Info*Engine queries databases and data repositories, it places the results of the queries in the VDB. In most cases, the last group in the VDB is the one needed for display in the web browser. However, all groups generated through a JSP page are available to other webjects on the page, and possibly to other webjects called on other pages in the session (depending on the scope set). This includes those standalone tasks that are executed through the JSP engine.

For standalone tasks, Info*Engine always makes this last group created through the task available for manipulation. The other groups retrieved by the task prior to the last group still exist in the VDB, but are not accessible without the use of the Return-Groups webject:

```
<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="group_names"/>
</ie:webject>
```

Using an example similar to the employee example described previously, the following scenario demonstrates how to obtain a separate lists of employees from the sales department and the development department of a company and then, unlike the previous example, displays each employee list, rather than displaying a combined list of employees.

To accomplish this, author an Info*Engine task named “listseparate” that queries a database for information about sales employees and development employees. The two Query-Objects webjects necessary to retrieve these two lists are the same webjects that were used in the previous example:

```
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
  <ie:param name="CLASS"     data="salesemp"/>
  <ie:param name="WHERE"     data="()"/>
  <ie:param name="GROUP_OUT" data="sales"/>
</ie:webject>

<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
  <ie:param name="CLASS"     data="devemp"/>
  <ie:param name="WHERE"     data="()"/>
  <ie:param name="GROUP_OUT" data="development"/>
</ie:webject>
```

As before, the first Query-Objects webject asks for information about employees from the sales employee table and the second Query-Objects webject asks for information from the development employee table. The results of the queries are then placed in the sales and development groups.

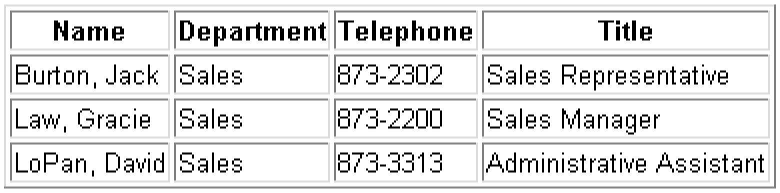
To execute the “listseparate” task and then view the two employee lists that are stored in the sales and development groups, you can create a JSP page. On the page, use the Info*Engine custom **task** tag to execute the task and include two display webjects to display the groups. By default, the task executes in the same JVM as the JSP engine that is processing the JSP page. Therefore, all of the groups created through the task are available to the webjects that execute after the task executes.

The task tag and display webjects that are designed to display both groups might look like this:

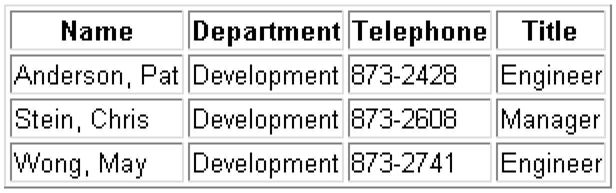
```
<ie:task uri="listseparate.xml"/>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"  data="sales"/>
  <ie:param name="BORDER"    data="1"/>
  <ie:param name="ATTRIBUTE"
            data="ename,department,phone,title" delim=","/>
  <ie:param name="HEADER"
```

```
            data="Name,Department,Telephone,Title" delim=","/>
</ie:webject>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"  data="development"/>
  <ie:param name="BORDER"    data="1"/>
  <ie:param name="ATTRIBUTE"
            data="ename,department,phone,title" delim=","/>
  <ie:param name="HEADER"
            data="Name,Department,Telephone,Title" delim=","/>
</ie:webject>
```

Notice that the task tag executes the “listseparate” task. After the task completes, both groups are available. The first Display-Table webject displays the sales group, which is specified in its GROUP_IN parameter. The second Display-Table webject displays the development group, which is specified in its GROUP_IN parameter. The resulting tables are similar to the following:



Name	Department	Telephone	Title
Burton, Jack	Sales	873-2302	Sales Representative
Law, Gracie	Sales	873-2200	Sales Manager
LoPan, David	Sales	873-3313	Administrative Assistant



Name	Department	Telephone	Title
Anderson, Pat	Development	873-2428	Engineer
Stein, Chris	Development	873-2608	Manager
Wong, May	Development	873-2741	Engineer

Putting the two Query-Objects webjects in a task separates the queries from the display as far as your code is concerned, but in this case, the resulting task execution takes place as if the Query-Objects webjects were on the same JSP page as the display webjects.

If you execute the “listseparate” task outside of the JVM where the JSP engine resides, then not all of the groups generated by the task are automatically available to the JSP page. Only the last group generated is automatically available. You can expand the “listseparate” task so that it returns more than the last group by adding the Return-Groups webject after the two query webjects. The “listseparate” task must end in a Return-Groups webject so that both groups are available to the calling task or JSP. The Return-Groups webject might look like this:

```
<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN"  data="sales"/>
  <ie:param name="GROUP_IN"  data="development"/>
</ie:webject>
```

In the Return-Groups webject, you can include multiple GROUP_IN parameters. Each of these parameters defines a particular group in the VDB that you want to have returned for further processing. A parameter value of “ * ” causes all groups in the task’s VDB to be returned.

To execute the “listseparate” task outside of the JVM where the JSP engine resides, specify the processor attribute on the **task** tag in the JSP page. For example, if the task processor has the default name of “com.myCompany.server.taskProcessor” you can include the processor attribute in the **task** tag as follows:

```
   <ie:task uri="listseparate.xml" processor="com.myCompany.server.
taskProcessor"/>
```

You can choose to execute tasks outside of the JVM where the JSP engine resides for performance reasons or for some other reason. Remember that whenever you do specify the processor, you must ensure that the task named returns all of the groups that are needed by the other webjects on the JSP page.

Sending and Retrieving Data Groups from Adapters

When you are working with data that is targeted for an information system, you get the data to the adapter that interfaces with the information system by specifying the GROUP_IN parameter on adapter webject you want to use. Only those groups specified on the GROUP_IN parameter are delivered to the adapter. All other groups in the VDB are not available to the adapter.

When you are working with data that is coming from an information system, name the group in which the data is returned by specifying the GROUP_OUT parameter on adapter webject you want to use. If for some reason the complete group cannot be returned, the adapter indicates this to Info*Engine and Info*Engine throws an exception. After executing adapter webjects that pull data from an information system, you should always check for possible exceptions.

Context Groups

When a task webject returns information from a data management system or from some other source, the information is returned as a group of objects and stored in the VDB. The VDB also contains special groups that store other information gathered during the request process. These groups are called “context” groups.

Context groups are maintained by Info*Engine and are made available to all webjects. Webjects can be authored to include the substitution syntax which substitutes values from the current context groups and modifies the behavior of the webject for this execution.

Three predefined groups are available within the collection of context groups: @SERVER, @FORM, and @COOKIE. A shorthand notation of SERVER, FORM, and COOKIE (excluding the @ symbol) can be used if no webject has created a group called SERVER, FORM, or COOKIE. The reserved names @SERVER, @FORM, and @COOKIE refer explicitly to the context groups and are the preferred names for the groups.

@SERVER
The @SERVER group contains attributes that are derived from the protocol used to communicate from the web browser to the web server. It can contain attributes such as **Accept-Language** or **Auth-User**. Refer to the current web-browser-to-web-server protocol specification to find more information on the individual attributes found in this group. To determine the exact contents of the current @SERVER group, you can execute the Echo-Request display webject.

@FORM
The @FORM group contains attributes that are obtained from the CGI query specification data that is received with the URL used to access the JSP page. It also contains any HTML form data that was received as the result of a web browser POST request.

@COOKIE
The @COOKIE group contains one element that has an attribute for each cookie that is processed during the connection to the JSP page. The attribute name is the name of the cookie and the value is the value of the cookie.

In addition to these groups, the **Auth-Map** context group is created when an authentication task executes. For more information, see Credentials Mapping on page 125.

Dynamic Parameter Value Substitution

Although most simple Windchill installations do not require them, webject parameter values must often be taken directly from user input in a web-based form or as a group of unknown size or kind from a data repository. Info*Engine provides a mechanism called “dynamic parameter value substitution” that can be used in the following situations:

- When information comes from a web-based form or as URL parameters.
- When information comes from the data objects in a VDB group.

The substitution required in a webject is applied once, immediately before a webject is executed. Therefore, to substitute values from a group that is created through the execution of a task, you must execute the task before you execute the webject that contains the substitution expressions. You cannot execute the task from within the webject.

You can use Info*Engine substitution expressions in data attribute values on the **param** tag used in any webjects.

General Value Substitution Syntax

Info*Engine substitution expressions are always delimited by the characters `$(` and `)`. Within these delimiters, specify the group and attributes for which the substitution is made as follows:

`$(`*`group_name`*`[`*`element_selector`*`]`*`attribute_name`*`[`*`value_selector`*`])`

where:

`group_name`

Names a group that exists in the VDB. This group can be a context group such as @FORM or a data group that results from the execution of a task.

`element_selector`

Identifies the elements (rows) within the group that are selected. The selector can be one of the following:

- An integer, which identifies an element by its index. The first element has an index of 0. Therefore, if a group contains 10 elements, their indices range from 0 to 9.
- The letter `N`, which selects the last element of the group.
- Empty, which selects all elements and concatenates them together into a string using no separators. The empty selector is usually used when there is only one element.
- The word `META`, which selects the metadata from the group instead of selecting elements.
- The asterisk (`*`) symbol, which selects all elements and concatenates them together into a string. Semicolons separate the elements in the string.
- An attribute and value pair specified as `attribute=value`. Info*Engine selects the first element it finds that contains the specified value for the named attribute. For example, assume that the "grp1" group contains "name" and "phone" attributes. To locate the phone number of an employee named "Doyle," use the following substitution expression:

```
$(grp1[name=Doyle]phone[])
```

`attribute_name`

Names an attribute (column) that exists within the selected elements.

If `META` is specified for `element_selector`, then one of the following attributes can be specified for `attribute_name`:

- `COUNT` is the number of elements in the group.
- `STATUS` is the numeric status value currently associated with the group. A status of 0 indicates success. A status of anything but 0 indicates failure.
- `NAME` is the name of the group.

- TYPE is the type of group. The group creator may set the meta value of TYPE but is not required to. Valid group types include:
    - **Object** – The group contains data.
    - **Status** – The group contains the success or failure of a request
    - **Exception** – The group contains the error information produced when an exception has occurred.
    - **Unknown** – The group creator did not set the TYPE information for the group.
- MESSAGE is the current message associated with the group. If there is no message associated with the group, the substitution is an empty string. You can retrieve multiple messages from the MESSAGE metadata. For example, to get all messages associated with "Grp123" you can use the following substitution expression:

  ```
  $(Grp123[META]MESSAGE[*])
  ```

  In addition, you can specify any metadata attribute defined in the group for attribute_name.

value_selector
Identifies the attribute values that are selected. The selector can be one of the following:

- An integer, which identifies a value by its index. The first value has an index of 0. Therefore, if an attribute contains 5 values, their indices range from 0 to 4.
- The letter N, which selects the last value of the attribute.
- Empty, which selects all values and concatenates them together into a string using no separators. The empty selector is usually used when there is only one value.
- The asterisk ("*") symbol, which selects all values and concatenates them together into a string. Commas separate the values in the string.

The following substitution expression selects all XYZ values from the element in the @FORM group and concatenates them together into a string using a comma to separate the values:

```
$(@FORM[]XYZ[*])
```

The following substitution expression selects the first NAME attribute value from the element in the USERS group:

```
$(USERS[]NAME[0])
```

The following substitution expression selects the COUNT value from the metadata in the OUTGRP group:

```
$(OUTGRP[META]COUNT[])
```

Default Values for Substitutions

To cover the possibility that the evaluation of a substitution expression could return no values, you should include a default for the expression. Specify a default by including the following **default** attribute on the **param** tag:

`default="`*`value`*`"`

For example, the following **param** tag sets the default for the ATTRIBUTE parameter to the string `"*"`:

```
<ie:param name="ATTRIBUTE" ... default="*"/>
```

Substitution Expressions for User-Specified Values

When information must be specified by a user rather than by the designer of a task, the substitution expressions that access the @FORM context group makes this information available within a webject even though the exact value is not known at the time the template or task is being designed.

You can use substitution expressions in any data attribute value on a **param** tag.

To specify one parameter value as input by a user in a web-based form or on a URL, the parameter value can be similar to the following:

`data="$(@FORM[]`*`variable`*`[])"`

You use `@FORM[]` in these formats because there is only one element in the @FORM context group. The empty value selector, `[]`, for the form variable is used because there is only one value equated to the variable.

To specify multiple parameter values as input, you include the asterisk (*) as the value selector. To format multiple parameter values correctly, you must also include the **delim** attribute to identify the comma as the separator in the string that is substituted. For multiple values, the parameter value can be similar to the following:

`data="$(@FORM[]`*`variable`*`[*])" delim=","`

The asterisk in the value selector, `[*]`, selects all values and concatenates them together into a string using the comma as a separator. The **delim** attribute tells the webject that multiple values in the data attribute are separated by a comma.

Note

If the form you use includes an INPUT element that has a control type of **file select** for selecting files to upload, all of the form variables you want to use in parameter substitution must be set on the form before the INPUT element. For more information, see Uploading and Downloading BLOBs on page 54.

Example: Display-Object Substitution Using @FORM

The following variables set up the table format and attributes that are visible to the user. The values for the variables are available in the @FORM context group and can be retrieved from this group.

```
border=2
att=ename
att=phone
headers=Name
headers=Telephone
```

Also assume that the following Display-Object webject resides in an Info*Engine JSP page and displays an employee list using the variables from the form:

```
<ie:task uri="listemployees"/>

<ie:webject name="Display-Object" type="DSP">
   <ie:param name="BORDER" data=$(@FORM[]border[])/>
   <ie:param name="ATTRIBUTE" data=$(@FORM[]att[*]) delim=","
                                default="*"/>
   <ie:param name="HEADER" data=$(@FORM[]headers[*]) delim=","/>
</ie:webject>
```

After substitution processing by Info*Engine, the parameters in the webject are as follows:

```
<ie:webject name="Display-Object" type="DSP">
   <ie:param name="BORDER" data="2"/>
   <ie:param name="ATTRIBUTE" data="ename,phone" delim=","
                               default="*"/>
   <ie:param name="HEADER" data="Name,Telephone" delim=","/>
</ie:webject>
```

Example: Query-Objects Substitution Using @FORM

Assume that the following URL has been submitted to “webServer1”:

```
http://webServer1/Windchill/servlet/IE/tasks/QueryObject.xml?CLS=EMPLOYEES&ATTS=
                           NAME&ATTS =EMPNO&ATTS=SALARY&WHR=NAME='&SMITH'
```

Then, the @FORM context group contains the following variables:

```
CLS=EMPLOYEES
ATTS=NAME
ATTS=EMPNO
ATTS=SALARY
WHR=NAME='&SMITH'
```

Also assume that the following Query-Objects webject resides in the `QueryObject.xml` task and queries the information system connected to the "`jdbc`" adapter using the variables from the URL:

```
<ie:webject name="Query-Objects" type="ACT">
   <ie:param name="INSTANCE" data="jdbc"/>
   <ie:param name="ATTRIBUTE" data="$(@FORM[]ATTS[*])" delim=","
                            default="*"/>
   <ie:param name="CLASS" data="$(@FORM[]CLS[0])" default="EMP"/>
   <ie:param name="WHERE" data="$(@FORM[]WHR[0])" default="()"/>
   <ie:param name="GROUP_OUT" data="queryStatus"/>
</ie:webject>
```

After substitution processing by Info*Engine, the parameters in the webject are as follows:

```
<ie:webject name="Query-Objects" type="ACT">
  <ie:param name="INSTANCE" data="jdbc"/>
  <ie:param name="ATTRIBUTE" data="NAME,EMPNO,SALARY" delim=","
                            default="*"/>
  <ie:param name="CLASS" data="EMPLOYEES" default="EMP"/>
  <ie:param name="WHERE" data="NAME='&SMITH'" default="()"/>
  <ie:param name="GROUP_OUT" data="queryStatus"/>
</ie:webject>
```

After all processing is completed by Info*Engine, the parameters in the webject are as follows:

```
<ie:webject name="Query-Objects" type="ACT">
  <ie:param name="INSTANCE" data="jdbc"/>
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE" data="EMPNO"/>
  <ie:param name="ATTRIBUTE" data="SALARY"/>
  <ie:param name="CLASS" data="EMPLOYEES"/>
  <ie:param name="WHERE" data="NAME='&SMITH'"/>
  <ie:param name="GROUP_OUT" data="queryStatus"/>
</ie:webject>
```

Substitution Expressions for VDB Data Group-Specified Values

When information must come from a VDB group that has been generated, the substitution expressions that access a VDB data group makes this information available within a webject even though the exact value is not known at the time the task is being designed.

Note

The VDB group must be available to the webject before the webject executes. You cannot execute the task that creates the group from within the webject.

You can use substitution expressions in any **data** attribute value on a **param** tag.

To specify one parameter value that is substituted from a VDB data group, the parameter value can be similar to the following:

```
data=$(grp_name[]att_name[])
```

Use `grp_name[]` in these formats when there is only one element (row) in the group named "grp_name." Use `att_name[]` in these formats when there is only one value in the **att_name** attribute. If the data group has more than one element, you can use `grp_name[0]` to access the first element in the group and use `grp_name[N]` to access the last element in the group. To access other elements, you can specify the index of the element inside the brackets. Similarly, you can use `0`, `N`, or other indices inside the brackets to access a specific value in the attribute named by `att_name[]`.

To specify multiple parameter values as input, you include the asterisk (`*`) as the value selector. To format multiple parameter values correctly, you must also include the **delim** attribute to identify the comma as the separator in the string that is substituted. For multiple values, the parameter value can be similar to the following:

```
data=$(grp_name[]att_name[*]) delim=","
```

The asterisk in the value selector, `[*]`, selects all values and concatenates them together into a string using the comma as a separator. The **delim** attribute tells the webject that multiple values in the **data** attribute are separated by a comma.

The last set of formats assumes that there is only one element in the named group. If the data group has more than one element, you can use `grp_name[0]` to access the first element in the group and use `grp_name[N]` to access the last element in the group. To access other elements, you can specify the index of the element inside the brackets.

For more information, see General Value Substitution Syntax on page 45.

Example: Create-Group Substitution Using A VDB Data Group

Assume that the following table represents the elements (rows), attributes (column headings) and attribute values (cells in columns) in the “EMP” VDB group:

ename	phone
Burton, Jack	873-2302
Law, Gracie	873-2200
LoPan, David	873-3313

You can create a new group called “salesEmp” by adding the DEPT attribute column to the “EMP” group using substitution in the following Create-Group webject:

```
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=$(EMP[0]ename[]):
              TELEPHONE=$(EMP[0]phone[]):DEPT=Sales"/>
  <ie:param name="ELEMENT" data="NAME=$(EMP[1]ename[]):
              TELEPHONE=$(EMP[1]phone[]):DEPT=Sales"/>
  <ie:param name="ELEMENT" data="NAME=$(EMP[2]ename[]):
              TELEPHONE=$(EMP[2]phone[]):DEPT=Sales"/>
  <ie:param name="CLASS" data="SalesEmployees"/>
  <ie:param name="GROUP_OUT" data="salesEmp"/>
</ie:webject>
```

After substitution processing by Info*Engine, the parameters in the webject are as follows:

```
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Burton, Jack:
              TELEPHONE=873-2302:DEPT=Sales"/>
  <ie:param name="ELEMENT" data="NAME=Law, Gracie:
              TELEPHONE=873-2200:DEPT=Sales"/>
  <ie:param name="ELEMENT" data="NAME=LoPan, David:
              TELEPHONE=873-3313:DEPT=Sales"/>
  <ie:param name="CLASS" data="SalesEmployees"/>
  <ie:param name="GROUP_OUT" data="salesEmp"/>
</ie:webject>
```

Executing the Create-Group webject creates the “salesEmp” group that is represented by the following table:

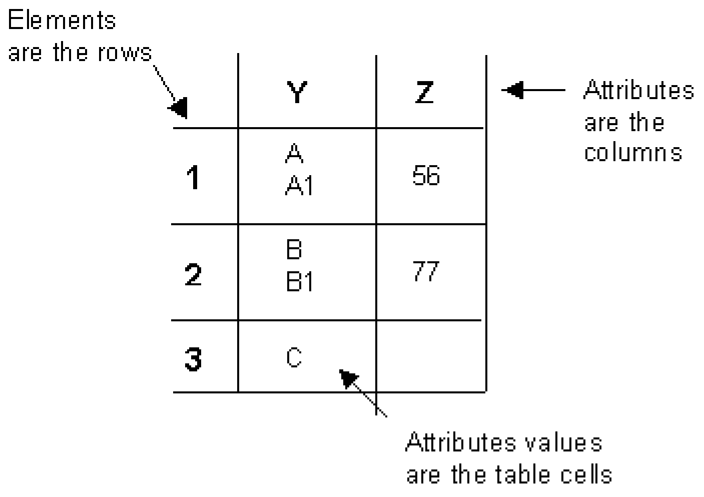
NAME	TELEPHONE	DEPT
Burton, Jack	873-2302	Sales
Law, Gracie	873-2200	Sales
LoPan, David	873-3313	Sales

Returning Multiple Values in Substitution Expressions

Using the asterisk (*) for the element selector or for the value selector in a substitution expression can return a string containing multiple values. The string can contain:

- Multiple attribute values from a single element and attribute pair.
- The attribute values from multiple elements that correspond to a single attribute.

To understand how these combinations are represented in the string, consider the following graphical representation of a group named "tbl":


Elements are the rows
Y
Z
Attributes are the columns
1
A
A1
56
2
B
B1
77
3
C
Attributes values are the table cells

Multiple Attribute Values from One Element

To return a string containing all `Y` attribute values from element 1 (`A` and `A1`), include the following data attribute:

```
data="$(tbl[0]Y[*])"
```

The string returned is:

```
"A,A1"
```

Notice that the comma separates the values.

To specify a different separator, you must include the **valueSeparator** attribute. For example, to return a colon-separated list, use the following:

```
data="$(tbl[0]Y[*])" valueSeparator=":"
```

Then, the string returned is:

```
"A:A1"
```

First Attribute Value From Each Element

To return a string containing the first `Z` attribute value from each element (`56` and `77`), include the following data attribute:

```
data="$(tbl[*]Z[0])"
```

By default, the string returned is:

```
"56;77;"
```

Notice that the semicolon separates values from different cells and that there is no value for the third element.

To specify a different separator, you must include the **elementSeparator** attribute. For example, to return a percent-separated list, use the following:

```
data="$(tbl[*]Y[0])" elementSeparator="%"
```

Then, the string returned is:

```
"56%77%"
```

Multiple Attribute Values From All Elements

To return a string containing all `Y` attribute values from all elements (`A`, `A1`, `B`, `B1`, and `C`), include the following data attribute:

```
data="$(tbl[*]Y[*])"
```

By default, the string returned is:

```
"A,A1;B,B1;C"
```

Notice that the comma separates the values in one cell and the semicolon separates values from different cells.

To specify different separators, you must include the **valueSeparator** and **elementSeparator** attributes. For example, to return a string that uses the colon and percent symbols as separators, use the following:

```
data="$(tbl[*]Y[*])" valueSeparator=":" elementSeparator="%"
```

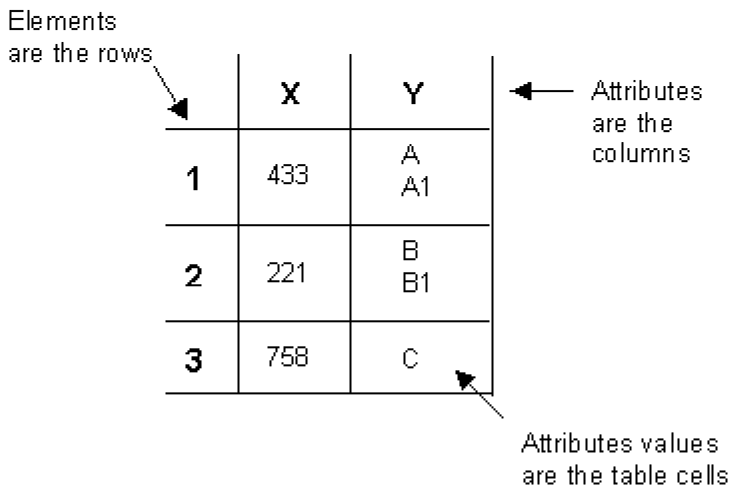
Then, the string returned is:

```
"A:A1%B:B1%C"
```

Selecting an Element by an Attribute-Value Pair in Substitution Expressions

Using an attribute–value pair for the element selector in a substitution expression provides a way to select an element without knowing the index number of the element.

To understand how the selections works, consider the following graphical representation of a group named “tbl”:


Elements are the rows
X
Y
Attributes are the columns
1
433
A
A1
2
221
B
B1
3
758
C
Attributes values are the table cells

Element Selection

To return a string containing all `Y` attribute values from the element where `X= 221`, include the following data attribute:

```
data="$(tbl[X=221]Y[*])"
```

The `X` attribute value of `221` is located in element number 2, therefore the `Y` attribute values returned in the string are:

```
"B,B1"
```

By default, the comma separates the values.

Uploading and Downloading BLOBs

BLOBs are Binary Large Objects. A BLOB can be any random large block of bits such as a Word document, a picture, or sound file.

You can use HTML forms and special adapter webjects to upload BLOBs from a web browser to a database, and download BLOBs from a database to a web browser. The following sections discuss the use of form variables and provide examples for uploading and downloading BLOBs.

For additional information on the adapter webjects that upload and download BLOBs, see the appropriate adapter guide.

For advanced users, Info*Engine also provides methods such as **setOutputStream**, **setInputStream**, and **sendContent** that can manipulate BLOBs and the Java language provides classes such as `java.io.ByteArrayInputStream` and `java.io.ByteArrayOutputStream` to read or write BLOBs to or from memory.

Note

The following examples include JDBC adapter webjects with SQL statements. These are simple examples, and do nothing to preprocess user-supplied input to protect database content from malicious users. Task authors should take precautions to validate user input to protect against malicious user attacks. Such attacks may attempt to exploit tasks using SQL injection or by supplying malformed parameter values, which can cause data corruption or allow access to protected data.

Using Form Variables When Uploading BLOBs

To upload BLOBs, the HTML form element on your JSP page must include the following attributes:

```
method="POST"
action="task_to_execute"
enctype="multipart/form-data"
```

These attributes establish the environment required for streaming the BLOB and storing the form variables.

To understand how to use form variables and how to control BLOB processing, it is helpful to know how the web browser, Info*Engine, adapter, and database interact to transfer BLOB data from the browser to a database. The following interactions identify the major steps that occur when a form is submitted from the browser:

- The browser sends the form variables and file data as a stream of data through the web server to the Info*Engine servlet.
- To maintain the optimum performance, the Info*Engine servlet and Info*Engine server process the stream as it is received rather than reading and storing the entire stream before doing anything.
- The servlet reads any form variables stored at the beginning of the input stream and stores them in the @FORM context group until the first BLOB data is encountered.
- When a BLOB is encountered, the Info*Engine servlet stops storing variables and passes the BLOB data on to its output stream so that the stream continues on to the Info*Engine server.

- By executing the task that is identified in the HTML form **action** attribute, the Info*Engine server connects to an adapter and passes the BLOB on to the adapter.
- By executing the webject in the task, the adapter then connects to a database and sends the BLOB to the database.

The following example code contains explanatory text and a form for selecting a file to store in the column of a database row. This example posts the data directly to an Info*Engine task using the Info*Engine servlet, meaning the client receives the XML output from task execution upon clicking “Submit”:

```
<%@page language="java" session="false" errorPage="../IEError.jsp"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>
<html>
<head>
<title>Upload File</title>
</head>
<body>
This page prompts for a "name" and "file" to store in an database.
The "name" is used to select a row in the table. The value of
"name" and the "file" contents are stored in columns in the selected row.
The table must be created before this example is run. The table could be created
using a SQL statement like 'CREATE TABLE BLOBTEST (NAME VARCHAR(60),FILECONTENT BLOB)'
<h2>Upload File to Oracle BLOB Column</h2>
<form method="POST" action="/Windchill/servlet/IE/tasks/com/company/UploadBlob.xml"
      enctype="multipart/form-data">
<TABLE>
  <tr> <td align=right>
      <B><FONT FACE=arial,helvetica>Adapter Instance:
    </td>
    <td>
      <INPUT name = "instance" type="text" size=50>
    </td>
  </tr>
  <tr> <td align=right>
      <B><FONT FACE=arial,helvetica>Name:
    </td>
    <td>
      <INPUT name = "filename" type="text" size=50>
    </td>
  </tr>
  <tr> <td align=right>
      <B><FONT FACE=arial,helvetica>File:
    </td>
    <td>
      <INPUT name = "file" type="file" size=50>
    </td>
  </tr>
  <tr> <td align=right>
      <INPUT type=submit NAME="submit" VALUE="Submit" id=button>
    </td>
  </tr>
</TABLE>
</form></body></html>
```

The file select control displayed through the third form INPUT element provides the vehicle from which the user selects the file to upload and, when the data stream is created, the BLOB data in the file selected is streamed right after the form variables set from the first two INPUT elements.

The `UploadBlob.xml` task identified in the **action** attribute of the form element is the task that the Info*Engine server executes. This task (which is described in the next section) uses the form variables to identify the adapter instance and specify the name that corresponds to the BLOB in the table row where the BLOB is stored.

To ensure that Info*Engine stores the form variables for use in the task, the form variables must be set in INPUT elements that are before the INPUT element that selects the BLOB file (as is done the previous example). If the order of the INPUT elements was reversed in the previous example, the form variables required by the task would not be stored in the @FORM context group because they would not appear in the stream until after the BLOB data. Any form variables in the stream after a BLOB are lost because the server does not read the entire stream before transferring it on to the adapter. Instead, the server transfers the BLOB directly to the adapter without first buffering the entire thing in memory. This optimizes performance and allows very large files to be sent to adapters without requiring enormous amounts of system memory.

Controlling Which Webjects Get Uploaded BLOBs

When BLOBs are uploaded to Info*Engine from a web page or Info*Engine-based application, Info*Engine cannot determine which adapter webjects consume the BLOBs or how many BLOBs each webject should consume. By default, Info*Engine attempts to deliver all available BLOBs to the first adapter webject (any webject with a type of `ACT` or `OBJ`). Sometimes, the first adapter webject is not a webject that consumes BLOBs, so the default behavior of Info*Engine is not always appropriate.

To control how BLOBs are consumed by webjects, you can include the BLOB_COUNT parameter on any adapter webject. This parameter specifies how many BLOBs should be delivered to the adapter webject. You can specify a value of `0` when no BLOBs should be delivered to the webject. If you omit the BLOB_COUNT parameter, all remaining BLOBs are delivered to the webject.

The following `UploadBlob.xml` example task contains three adapter webjects. The first two webjects (Do-Sql) delete and add rows to a database table and do not use BLOBs. On these webjects, the BLOB_COUNT parameter is set to `0`. The third webject (Put-Blob-Stream) is the webject that stores the BLOB and it has been defined to accept one BLOB. The values for the @FORM variables used in the parameters for the webjects can be supplied through a form like the form described in the previous section.

Note

You must set the BLOB_COUNT parameter to `0` for every webject except for the webject you want the BLOBS to be delivered to.

The example task assumes that the database table contains the following columns:

- The NAME column contains the name of the BLOB.
- The FILECONTENT column contains the BLOB data.

The code for the example task is as follows:

```
<%@page language="java" session="false"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
<!--
  Upload a file from the browser and save in an oracle
  blob column.
--/>

<ie:webject name="Do-Sql" type="ACT">
  <ie:param name="INSTANCE" data="$(FORM[]instance[])"
                            default="jdbcAdapter"/>
  <ie:param name="SQL"
      data="DELETE FROM BLOBTEST WHERE NAME=
           '$(FORM[]filename[0])'"/>
  <ie:param name="CLASS" data="BLOBTEST"/>
  <ie:param name="GROUP_OUT" data="TEMP"/>
  <ie:param name="BLOB_COUNT" data="0"/>
</ie:webject>

<ie:webject name="Do-Sql" type="ACT">
  <ie:param name="INSTANCE" data="$(FORM[]instance[])"
                            default="jdbcAdapter"/>
  <ie:param name="SQL"
      data="INSERT INTO BLOBTEST VALUES
           ('$(FORM[]filename[0])', EMPTY_BLOB())"/>
  <ie:param name="CLASS" data="BLOBTEST"/>
  <ie:param name="GROUP_OUT" data="TEMP"/>
  <ie:param name="BLOB_COUNT" data="0"/>
</ie:webject>

<ie:webject name="Put-Blob-Stream" type="OBJ">
  <ie:param name="INSTANCE" data="$(FORM[]instance[])"
                            default="jdbcAdapter"/>
  <ie:param name="CLASS" data="BLOBTEST"/>
  <ie:param name="ATTRIBUTE" data="FILECONTENT"/>
  <ie:param name="WHERE"
            data="(NAME='$(FORM[]filename[0])')"/>
  <ie:param name="GROUP_OUT" data="TEMP"/>
</ie:webject>
```

Using Form Variables for Downloading BLOBs

The following example code produces explanatory text and a form for selecting a row from a database table containing a BLOB. Clicking "Retrieve" invokes an Info*Engine task that selects the corresponding row from the database table and returns the BLOB it contains:

```
<%@page language="java" session="false" errorPage="../IEError.jsp"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>
<html>
<head><title>Send Blob</title>
</head>
<body>
<P>This page prompts for a "name" of the BLOB file to retrieve from an
database and a "MIME type" to associate with the BLOB file. The
"name" is used to select a row in the table where the BLOB is
stored. The table must be created and BLOBs uploaded to rows in
the table before this example is run. The table could be
 created using a SQL statement like 'CREATE TABLE BLOBTEST
 (NAME VARCHAR(60),FILECONTENT BLOB)'.

<H2>Download File from BLOB Column</H2>
<P>The MIME type determines which application is opened when the BLOB is
received.  The MIME type is not stored in the table. Enter one of the
following MIME types:
<blockquote>'text/plain'<br>
'application/msword'<br>
'application/msexcel'<br>
'application/vnd.ms-excel'<br>
'image/gif'</blockquote>
```

```
   <P>Include the single quotes in your MIME Type entry.</p>

   <form method="POST" action="/Windchill/servlet/IE/tasks/com/company/DownloadBlob.xml"
enctype="multipart/form-data">
   <TABLE>  <tr> <td align=right>
         <B><FONT FACE=arial,helvetica>Adapter Instance:
       </td>
       <td>
         <INPUT name = "instance" type="text" size=50>
       </td>
     </tr>
     <tr> <td align=right>
         <B><FONT FACE=arial,helvetica>Name:
       </td>
       <td>
         <INPUT name = "filename" type="text" size=50>
       </td>
     </tr>
     <tr> <td align=right>
         <B><FONT FACE=arial,helvetica>Mime Type:
       </td>
       <td>
         <INPUT name = "mimetype" type="text" size=50>
       </td>
     </tr>
     <tr> <td align=right>
         <INPUT type=submit NAME="submit" VALUE="Retrieve" id=button>
       </td>
     </tr></TABLE></form></body></html>
```

The `DownloadBlob.xml` task identified in the **action** attribute of the form element is the task that the Info*Engine server executes to download the BLOB. This task (which is described in the next section) uses form variables to identify the adapter instance, specify the name that corresponds to the BLOB in the table row where the BLOB is stored, and set the MIME type.

Example: BLOB Download Task

The following `DownloadBlob.xml` example task contains one adapter webject. This webject downloads one BLOB to the web browser. The MIME type specified in the MIMETYPE parameter is passed back to the browser and determines which application the browser launches to display the BLOB. The values for the @FORM variables used in the parameters for the webject can be supplied through a form like the form used in the previous section.

The example task assumes that the database table contains the following columns:

- The NAME column contains the name of the BLOB.
- The FILECONTENT column contains the BLOB data.

```
<%@page language="java" session="false"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<!--
    Possible MIME Types
      application/msword
      text/plain
      application/msexcel
      application/vnd.ms-excel
--/>

<ie:webject name="Send-Blob-Stream" type="OBJ">
  <ie:param name="INSTANCE" data="$(FORM[]instance[])"
                            default="jdbcAdapter"/>
  <ie:param name="CLASS" data="BLOBTEST"/>
  <ie:param name="ATTRIBUTE" data="FILECONTENT"/>
  <ie:param name="MIMETYPE" data="$(FORM[]mimetype[0])"/>
  <ie:param name="WHERE"
            data="(NAME='$(FORM[]filename[0])')"/>
  <ie:param name="GROUP_OUT" data="STATUS"/>
</ie:webject>
```

Example Federated Search for Parts

This is a basic federated Info*Engine application example that allows you to do a simultaneous basic search for parts by part number across any number of information systems (read-only).

The following actions are allowed in this example application:

- Query—Search a given repository for parts.
- Fetch—Fetch a single part from a given repository.
- Download—Download the primary content of a part.

This example can be located in your Windchill installation at:

`<Windchill>/prog_examples/federation/search_portal`

For an in-depth description of the example, open the `ReadMe` file that is included with this example in the `doc` subdirectory.

3

Info*Engine JSP Pages

The following topics introduce the concept of an Info*Engine JSP page and describe how to use Info*Engine custom tags.

Authoring Info*Engine JavaServer Pages

JavaServer Pages (JSP) is a core technology of the Java Platform, Enterprise Edition (Java EE) and solutions based upon EJB (Enterprise Java Beans). Info*Engine supports the development of enterprise custom Java applications and provides a JSP processor as an extension of the Info*Engine servlet engine. The JSP processor dynamically translates JSP pages into servlets.

Usually, a JSP page is an HTML page with some additional JSP tags and some embedded Java code. However, inclusion of JSP tags or embedded Java is not mandatory, so a page containing only HTML is a legitimate JSP page.

JSP pages that interact with Info*Engine usually contain a simple set of JSP tags and a set of custom Info*Engine tags that define the webjects that are then executed when the page is accessed. For example, the following `DisplayTable.jsp` page creates a group containing one element and displays the table-formatted results:

```
   <%@page language="java"  session="false"
     errorPage="IEError.jsp"%>

   <%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>

   <html>
   <head>
   <title>JSP Display-Table</title>
   </head>

   <body bgcolor="#FFFFFF">
   <h3> I*E Display-Table JSP Using Taglibs <h3>

   <ie:webject name="Create-Group" type="GRP">
     <ie:param name="ELEMENT"
        data="name=myGroup2:email=xxx@xxx.com:address=PTC"/>
     <ie:param name="GROUP_OUT" data="newGroup2"/>
   </ie:webject>

   <ie:webject name="Display-Table" type="DSP"/>

</body>
</html>
```

⚠ **Caution**

Due to the upgrade to the latest JSP specification, JSPs using Info*Engine substitution syntax can no longer use the `${...}` syntax. Info*Engine now supports an alternate syntax, `$(...)` that must be used instead. All Info*Engine tasks now function with either a syntax of `$(...)` or `${...}`. Preexisting tasks continue to function as is and do not have to be changed. Preexisting JSPs must be updated to use the new `$(...)` syntax or they fail to compile and do not function as expected.

For more information, see Using the Info*Engine Task Editor on page 95.

Storing Info*Engine JSP Pages

The root of the Windchill web application is `<Windchill>/codebase`, and therefore all Info*Engine JSPs must be stored somewhere within that codebase directory. Ideally, you should create a directory hierarchy to organize your JSPs, such as `<Windchill>/codebase/com/company/`.

Accessing Info*Engine JSP Pages

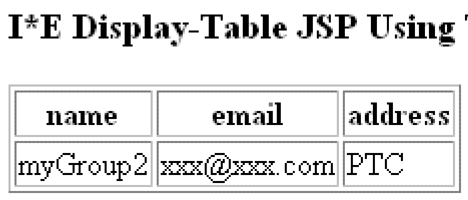
The installer also specifies an application URL that is used as the URL prefix for requesting Info*Engine JSP pages. You can produce the URL that contains the request to execute Info*Engine JSP pages by doing the following:

- Include the host name and port (if the port is not the default), and the application URL prefix specified when Info*Engine was installed. The default application URL is “Windchill.”
- Name the path for the JSP page that is relative to the `codebase` directory.
- Specify any optional values to pass to the page.

Therefore, to execute the `com/company/DisplayTable.jsp` page using the “myServer” host name and the “Windchill” application URL, specify the following URL:

```
http://myServer/Windchill/com/company/DisplayTable.jsp
```

The output displayed in the browser is similar to the following:



I*E Display-Table JSP Using Taglibs

name	email	address
myGroup2	xxx@xxx.com	PTC

Usually, a web server supporting JSP is configured so that it recognizes any file name with the `.jsp` extension as a JSP page. When a URL references this type of file, the web server passes the URL to its JSP processor. The JSP processor then checks to see if it already has a servlet for this page. If not, it automatically translates the page to a servlet and then executes that servlet. If the page contains only HTML, the generated servlet is trivial and consists of one or more Java print statements that simply send the HTML to the browser. If, on the other hand, the JSP page contains some embedded Java code, that code is incorporated directly into the servlet that is generated.

If the JSP processor detects that it already has a servlet for the URL that has been passed to it, then it checks to see if the page has been modified since the last time that the servlet was generated. If it detects that the page has been updated since the last servlet generation, it automatically regenerates the servlet. Otherwise, it reuses the previously generated servlet.

Creating Info*Engine JSP Pages

Each Info*Engine JSP page should contain the following items:

- The standard JSP `page` directive, which defines the general page characteristics.
- The standard JSP `taglib` directive, which identifies a tag library containing Info*Engine custom tags.

  You must put this directive before any lines that use the custom tags in the library.
- Info*Engine custom tags, which provide access to a set of custom actions that encapsulate recurring functionality.

  The custom tags provide the syntax for executing webjects and provide the structure around which you can build a JSP.

To be well-formed and valid, the Info*Engine JSP pages must follow basic JSP rules:

- For Info*Engine custom tags and JSP tags, you must use lowercase. For example, you must specify the **webject** tag as “webject” and not “WEBJECT” or “Webject” or even “webJecT”.
- You can use comments to document what is happening in your page or to cause the compiler to skip a section of the page. If a webject is surrounded by a comment, it is not executed. Comments can be located anywhere in a task except within tags, declarations, or other comments.

  Comments begin with `<!--` and end with `-->`.

- Empty elements must be properly constructed. The trailing `/>` characters (the forward slash followed by the right angle bracket) in the JSP syntax indicates that the element is empty and no matching end tag should be sought. For example, the **param** custom tags make use of the empty element construction.
- Additional general rules for using scriptlets, expressions, declarations, directives, and Info*Engine custom tag elements can be found in the Info*Engine Custom Tag Reference on page 243.

The example `DisplayTable.jsp` page includes the following standard JSP directives:

- `IEError.jsp` as the standard page directive. It contains a reference to the example error page:

```
<%@page language="java"  session="false"
  errorPage="IEError.jsp"%>
```

- The `ie` prefix as the required prefix. The prefix identifies the tag as belonging to the core tag library, which provides the general custom tags that can be used in a JSP page:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
```

The previous `DisplayTable.jsp` page also includes the **webject** and **param** custom tags:

- The **webject** tags identify the Create-Group and Display-Table webjects.
- The **param** tags supply webject parameters and parameter values.

  For the Create-Group webject, the parameters define the name of the output group and the attributes and values of elements in the group.

An Info*Engine JSP page can also contain any supported HTML and JSP tag. This guide does not describe the HTML and JSP tags that you can use. To review the supported tags, you can access the following sites:

http://java.sun.com

http://www.w3.org/

Executing Tasks from JSP Pages

Although you can include any webject in a JSP page, you might want to separate the webjects that generate and manipulate data from those that display data. This is done by creating text-based documents called standalone tasks and executing the tasks through the Info*Engine **task** tag. The **uri** attribute on the **task** tag names the task to execute.

Info*Engine Tasks on page 85 describes how to create and use tasks.

Specifying URIs and URLs

Info*Engine parameters can require that you specify URIs or URLs as their values.

Note

The URIs and URLs use the forward slash as the separator (/) in paths even though the back slash (\) is the directory separator used on Windows systems. Info*Engine correctly identifies all system URIs and URLs when you specify the forward slash. You should only use a back slash in URIs where a back slash is being used outside of the context of use as a path separator.

File Entries

Some custom tags require that you enter specific parameter values as valid URIs to files. Some webjects also require that you enter specific parameter values as valid URLs to files. In both cases, you are entering the location of a file to execute as defined by the tag or webject. The URI or URL can be relative or absolute:

- Relative URIs and URLs reference files that reside under the root file system directory that is defined for the local Info*Engine task processor.
- Absolute URIs and URLs reference files that reside in the local file system, on a remote HTTP server, or are referenced through an accessible LDAP directory.

Absolute file references can be grouped as follows:

- References starting with `file:///` are to the local file system. When specifying drive letters, it is good practice to use all uppercase or all lowercase drive letters. Info*Engine distinguishes between upper and lower case. As a result, if two file references are the same except for the case of the drive letter, then Info*Engine treats those references as two distinct paths.
- References starting with `http://` identify files on a remote system.
- References starting with `ldap://` identify tasks that are stored in attributes of LDAP entries.

  The general format of an LDAP URL is:

  ```
  ldap://hostname:port/search_base?attribute_
  names?scope?filter
  ```

To specify an LDAP URL, replace each part of the URL with the appropriate value:

- *`hostname:port`* locates the LDAP directory server. The port can be omitted when the server answers on the default LDAP port, which is port 389.
- *`search_base`* is the distinguished name identifying the entry that is the root of the subtree that is to be searched.
- *`attribute_names`* is a comma-separated list of the attributes to be returned from the selected directory entries. By default, all attributes are returned.
- *`scope`* determines the scope of the search. Valid scope values are:

  SUB — for a subtree search.

  ONE — for a single-level search.

  BASE — to search only the entry identified by the distinguished name.

By default, the scope is set to BASE.

- *`filter`* defines the search filter. By default, all entries are selected.

To locate an LDAP entry containing a task, you can omit the scope and filter attributes from the format. You specify the distinguished name of LDAP entry containing the task and the attribute name of the attribute in which the task source is stored.

LDAP Search Base Entries

In other cases, including several tags in the directory tag library, you might be required to enter an LDAP URL that identifies a search base LDAP entry. Using the format listed above, this type of entry uses only the **search_base** attribute:

```
ldap://hostname:port/search_base
```

Example File Locations

Example URI locations of a `task1.xml` task file are as follows:

- Assume that the file resides in the `C:\ptc\Windchill\tasks\infoengine\jdbc` directory and that `C:\ptc\Windchill\tasks` is the root file directory of the local task processor. Then use the following relative URI to locate the file:

```
uri="infoengine/jdbc/task1.xml"
```

- Assume that the file resides on the local file system in the `D:\ie\jdbc` directory. Then use the following absolute URI to locate the file:

```
uri="file:///D:/ie/jdbc/task1.xml"
```

- Assume that the file resides in the `opt/ptc/Windchill/infoengine/tasks/jdbc` directory on the svr2 remote host. Then use the following absolute URI to locate the file:

```
uri="http://svr2/opt/ptc/Windchill/infoengine/tasks/jdbc/task1.xml"
```

- Assume that the task resides in an LDAP server on the `cn=IEtasks` node and that the name of the attribute where the task is stored is `ptcCommandDelegateCode`. Also assume that the server host name is srv3 and the **dc** attributes are `myHost` and `com`. Then use the following absolute URI to locate the task:

```
uri="ldap://srv3/cn=IEtasks,dc=myHost,dc=com?
ptc.CommandDelegateSource"
```

Info*Engine Custom Tags and JSP Pages

Info*Engine tags provide access to a set of custom Info*Engine actions that encapsulate recurring functionality. Using these tags, you can easily code JSP pages that do these actions.

Info*Engine provides you with the following tag libraries that contain custom tags for your use:

- The core library contains general Info*Engine tags that make coding an Info*Engine JSP page simpler. For more information, see Core Library Tags on page 256.
- The directory library contains Info*Engine tags that can be used when you are manipulating information that is in a directory. For more information, see Directory Library Tags on page 283.
- The supplied library contains Info*Engine tags that are designed for authoring Info*Engine Tasks. For more information, see Supplied Library Tags (JSTL) on page 290, Supplied Library Tags (Logging) on page 295, and Supplied Library Tags (XSL) on page 296.

As described previously, before using these tags on a JSP page, you must identify the library and its prefix using a JSP `taglib` directive. For all examples shown in this guide, the `ie` prefix is specified for the core tag library as follows:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
```

In addition, the `iedir` prefix is specified for the directory tag library as follows:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>
```

For details on the specified prefixes for the supplied tag libraries, refer to the Info*Engine Custom Tag Reference on page 243. However, you can use a prefix of your choice as long as it uniquely identifies the tags on the page.

Common Uses for Custom Tags

Action	Custom Tag
Define webjects and their parameters	**webject** **param**
Execute Info*Engine tasks and define their parameters	**task** **param**
Authenticate users who access the JSP page	**authenticate**
Group a sequence of webjects, tasks, or Java code so that the group is executed as a unit and optionally include success and failure processing These tags are supported in JSPs but their use is strongly discouraged from within JSPs due to how taglibs must be processed. Ideally these tags should only be used from Info*Engine tasks.	**unit** **init** **success** **failure**
Iterate through an existing Info*Engine group one element at a time and retrieve the value	**forEach** **getValue**
Concurrently execute tasks or webjects	**parallel**

The following sections include examples that use some of these custom tags. In addition, there are other tags in the Info*Engine tag libraries that may be useful to you. The complete list of tags, their syntax, and descriptions can be found in the Info*Engine Custom Tag Reference on page 243.

For the general rules that apply to both JSP pages and Info*Engine tasks, see Info*Engine Custom Tag Reference on page 243.

Adapter Webjects and JSP Pages

Action and query webjects are defined for the Info*Engine adapters installed at your site. Each adapter is designed to access data from a specific type of information system. For detailed adapter webject descriptions and syntax, see the adapter guide for the information system you want to access.

Even though each adapter provides a unique set of webjects (each with their own set of parameters), Info*Engine has standardized a way to access the adapter through each webject. To do this, Info*Engine requires that you include the INSTANCE parameter in each adapter webject. This parameter identifies the adapter that is to be used. Although the name used to identify the adapter can take on many different forms, the most common name used is the service name defined for the adapter when it is configured.

A common way to name adapters is to include a domain name as part of the service name so that the adapter service name is unique across your entire Info*Engine environment. For example, if the domain name of the computer used to define the service name is “myHost.myCompany.com,” then a JDBC adapter service name might be:

```
com.myCompany.myHost.jdbcAdapter
```

Your site can also use simpler service names. For example, if there are only two JDBC adapters installed at your site, they can be named “JDBCadapter1” and “JDBCadapter2.” Your site administrator determines what the adapter service names are when the adapters are installed and configured. Generally, all service names must be unique, and maintaining uniqueness is the responsibility of the Windchill administrator.

Specify the INSTANCE parameter on each adapter webject. For example, it is specified on the following Query-Objects webject, which queries a database table using the JDBC adapter:

```
<%@page language="java" session="false"
                                      errorPage="IEError.jsp"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                  prefix="ie"%>

<html>
<head>
<title> JSP TableList </title>
</head>
<body bgcolor="#FFFFFF">
<h2> I*E JSP Test - TableList </h2>
<%
    String where = request.getParameter ("where");
    if ( where == null )
       where = "()";

    String table = request.getParameter ("table");
    if ( table == null )
       table = "EMP";

    String instance = request.getParameter ("instance");
    if ( instance == null )
       instance = "jdbcAdapter";
%>

<p><b>
Searching table <%= table%> with where clause <%= where%>.
</b></p>
```

```
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE" data="<%=instance%>"/>
  <ie:param name="ATTRIBUTE" data="*"/>
  <ie:param name="CLASS" data="<%=table%>"/>
  <ie:param name="WHERE" data="<%=where%>"/>
  <ie:param name="GROUP_OUT" data="myGroupOut"/>
</ie:webject>

<p>
<ie:webject name="Display-Table" type="DSP"/>
</body>
</html>
```

Note

In most cases, the preferable and standard way to enter parameter values is to use Info*Engine syntax. Instead of using HTML syntax:

```
   <ie:param name="INSTANCE" data="<%=instance%>"/
```

You should use Info*Engine syntax:

```
   <ie:param name="INSTANCE" data="$(@FORM[]instance[])" default=
"jdbcAdapter"/>
```

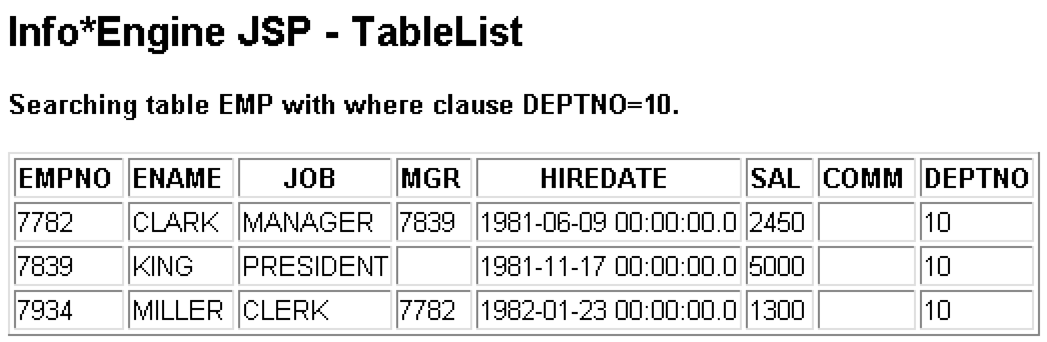
The INSTANCE parameter is just one of the required parameters. To execute the example page, you must specify the following information on the URL:

- An adapter name, such as “jdbcAdapter1”
- A table name, such as “EMP”
- A `where` clause, such as `where=DEPTNO=10`

You use this to create the following URL:

```
http://localhost/Windchill/infoengine/com/company/
TableList.jsp?...
```

In this case, `jdbcAdapter1` becomes the value of the INSTANCE parameter, `EMP` becomes the value for CLASS, and the `where` clause is `where=DEPTNO=10`. The output displayed in the browser could be similar to the following:



Info*Engine JSP - TableList

Searching table EMP with where clause DEPTNO=10.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.0	2450		10
7839	KING	PRESIDENT		1981-11-17 00:00:00.0	5000		10
7934	MILLER	CLERK	7782	1982-01-23 00:00:00.0	1300		10

If the adapter named “jdbcAdapter1” had not been available, an error would have been returned. To minimize the effect of an adapter being unavailable, you can specify multiple adapters if you know that there is more than one adapter that can process a specific webject.

To specify multiple adapters, include multiple INSTANCE parameter values on the webject. Info*Engine attempts to connect to the adapters in the order specified on the webject. For example, if you know that both adapters “jdbcAdapter1” and “jdbcAdapter2” are defined in your Info*Engine environment, you could include both in the Query-Objects webject as follows:

```
<%
    String where = request.getParameter ("where");
    if ( where == null )
       where = "()";

    String table = request.getParameter ("table");
    if ( table == null )
       table = "EMP";

    String instance1 = request.getParameter ("instance1");
    if ( instance1 == null )
       instance1 = "jdbcAdapter1";
    String instance2 = request.getParameter ("instance2");
    if ( instance2 == null )
       instance2 = "jdbcAdapter2";
%>

<p><b>
Searching table <%= table%> with where clause <%= where%>.
</b></p>

<ie:webject name="Query-Objects" type="OBJ">
```

```
    <ie:param name="INSTANCE" data="<%=instance1%>"/>
    <ie:param name="INSTANCE" data="<%=instance2%>"/>
    <ie:param name="ATTRIBUTE" data="*"/>
    <ie:param name="CLASS" data="<%=table%>"/>
    <ie:param name="WHERE" data="<%=where%>"/>
    <ie:param name="GROUP_OUT" data="myGroupOut"/>
  </ie:webject>
```

The URL to execute the updated `TableList.jsp` page is as follows:

`http://localhost/Windchill/infoengine/com/company/TableList.jsp?...`

If "jdbcAdapter1" is available, it is used; if it is not available, "jdbcAdapter2" is used. If neither is available, an error is returned.

Info*Engine also provides the following parameters that can be added to each adapter webject and may be useful when attempting to connect to an adapter:

CONNECTION_ATTEMPTS
Defines the maximum number of times to attempt establishing a connection to an adapter before returning an error. If multiple INSTANCE parameter values are specified, the value of CONNECTION_ATTEMPTS defines the maximum number of times to iterate through the list of adapter instances.

CONNECTION_ATTEMPT_INTERVAL
Defines the amount of time, in seconds, to delay between connection attempts. If multiple INSTANCE parameter values are specified, the value of CONNECTION_ATTEMPT_INTERVAL defines the number of seconds to wait between the attempts to iterate through the entire list of adapter instances.

For example, these parameters could also be added to the previous Query-Objects webject example as follows:

```
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE" data="<%=instance1%>"/>
  <ie:param name="INSTANCE" data="<%=instance2%>"/>
  <ie:param name="CONNECTION_ATTEMPTS" data="3"/>
  <ie:param name="CONNECTION_ATTEMPT_INTERVAL" data="30"/>
  <ie:param name="ATTRIBUTE" data="*"/>
  <ie:param name="CLASS" data="<%=table%>"/>
  <ie:param name="WHERE" data="<%=where%>"/>
  <ie:param name="GROUP_OUT" data="myGroupOut"/>
</ie:webject>
```

Adding the parameters sets up three attempts to connect to either adapter and provides a 30 second delay after attempting both connections and before trying again.

For additional information about the using adapter webjects, see the adapter guides for the adapters you are using.

Task Webjects and JSP Pages

The Info*Engine task webjects give you a quick way to generate and manipulate information, as well as to manage your JSP pages. Info*Engine maintains the information that is generated in groups within its VDB, as discussed in Info*Engine Data Management on page 31. Task webjects do not return data to a displayable device such as a web browser. Any data generated by a task webject is returned as a group to the VDB.

Use task webjects along with display webjects to produce the data solution that your site requires. Both task and display webjects can be used on the same JSP page to generate and then display data.

Info*Engine provides the following types of task webjects:

Type	Description
GRP	Group webjects can create, compare, combine, sort, or otherwise manipulate groups.
MGT	Management webjects provide common functions that you can use to manage your JSP pages or tasks.
MSG	Message webjects can be used in conjunction with a third-party MOM for generic messaging functions and task queuing functions.
WES	Web Event Service (WES) webjects can be used in conjunction with a third-party MOM for handling Info*Engine events.
ACT	Action webjects are provided by adapters to perform actions such as creating, copying, and updating data in a data repository.
ADM	Administrative webject perform specific administrative functions, such as gathering simple statistics, or causing service properties to reload at runtime.
OBJ	Query webjects are provided by adapters to search external databases for objects that match specified criteria.

You can use group and management webjects on any JSP page or in any task with just the basic Info*Engine environment set up. To use message or Web Event Service webjects, your site administrator must have installed and configured a MOM and implemented the Info*Engine features that provide the environment required by the webjects.

In addition, you can create external custom webjects (`EXT` type) that provide custom solutions in either a JSP page or a standalone task.

The `DisplayTable.jsp` page example previously provided contains the following group webject:

```
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT"
     data="name=myGroup2:email=xxx@xxx.com:address=PTC"/>
  <ie:param name="GROUP_OUT" data="newGroup2"/>
```

```
</ie:webject>
```

The result of executing the Create-Group webject is the “newGroup2” group, which has one element (row), with three attributes (columns):

name	email	address
myGroup2	xxx@xxx.com	PTC

Use the **webject** tag to code task webjects, as well as display webjects. In this example, the **webject** tag syntax includes both the beginning and ending **webject** tags. In the previous display webject example, only the beginning tag was needed. The ending **webject** tag is needed here because the Create-Group webject has two parameters that are nested inside the **webject** tag: ELEMENT and GROUP_OUT. These parameters are defined using **param** tags.

Task Webject Reference on page 357 describes all group, management, message, and Web Event Service webjects. There are also task webjects used in the examples shown throughout the remaining sections on this topic.

The action and query webjects available to you are determined by which adapters have been installed at your site. For more information, see Adapter Webjects and JSP Pages on page 71.

Display Webjects and JSP Pages

The Info*Engine display webjects give you a quick way to display information that has been gathered and manipulated by Info*Engine and its adapters. A display webject is specified by `DSP` in the **type** attribute on the **webject** tag. For example, the `DisplayTable.jsp` page described previously contains the following **webject** tag:

```
<ie:webject name="Display-Table" type="DSP"/>
```

Notice that the **type** attribute on the **webject** tag is set to “`DSP`”, which indicates that the Display-Table webject is a display webject. In this case, the webject requires no parameters and therefore, the **webject** tag syntax used requires only the /> characters to end the tag.

To display information in a web browser, you can combine HTML display elements with the display webjects that are specifically tailored for use with HTML. These webjects provide the ability to display generated data in the following formats:

Display Format	Webject
HTML tables	Display-Table
HTML form elements	Display-Selection
HTML elements such as a check box, radio button, or text box	Display-Value
Author-specified HTML coding	Display-Object

When coding webjects that utilize HTML, you should nest all of your HTML tags and webjects inside the following set of HTML tags:

```
<html>
<body>
  :
  :
</body>
</html>
```

In addition to the general display webjects listed in the previous table, there are display webjects available for use with HTML that provide the ability to do the following:

- Apply an XSL stylesheet to data
- Display data coded in XML tags
- Provide localized text

The `DisplayTable.jsp` example page described earlier shows one use of the Display-Table webject. Display Webject Reference on page 299 describes all display webjects. There are also display webjects used in the examples shown in the following sections.

Multiple Webject Execution

By using the **parallel** custom tag, you can execute more than one webject at the same time.

For example, the following JSP page executes two queries at the same time. When both queries complete, the resulting groups are joined and then displayed:

```
<html>
<body>

<ie:parallel>

  <ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="EMP"/>
    <ie:param name="WHERE" data="()"/>
    <ie:param name="GROUP_OUT" data="employees"/>
  </ie:webject>

  <ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="DEPT"/>
    <ie:param name="WHERE" data="()"/>
    <ie:param name="GROUP_OUT" data="departments"/>
  </ie:webject>

</ie:parallel>

<ie:webject name="Join-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="employees"/>
  <ie:param name="GROUP_IN" data="departments"/>
  <ie:param name="JOINBY" data="DEPTNO"/>
  <ie:param name="JOIN_TYPE" data="MAX"/>
  <ie:param name="SORTED" data="ASC"/>
  <ie:param name="GROUP_OUT" data="join-results"/>
</ie:webject>

<ie:webject name="Display-Object" type="DSP">
  <ie:param name="BORDER" data="1"/>
  <ie:param name="CAPTION" data="taglib query table"/>
  <ie:param name="FOOTER" data="done."/>
  <ie:param name="UNDEFINED" data="-"/>
  <ie:param name="ALIGN" data="left"/>
  <ie:param name="VALIGN" data="top"/>
  <ie:param name="CELLPADDING" data="5"/>
  <ie:param name="CELLSPACING" data="1"/>
</ie:webject>

</body>
</html>
```

For the details on the custom tags used, see the tag descriptions in the Info*Engine Custom Tag Reference on page 243.

Success and Failure Processing for Webjects

Info*Engine provides the following set of custom tags that you can use to group the execution of webjects and provide success and failure processing:

- **unit**
- **init**
- **success**
- **failure**

These tags are supported in JSPs, but their use is strongly discouraged from within JSPs due to how a **taglib** must be processed. Ideally these tags should only be used from Info*Engine tasks.

For example, the following JSP page has the main body of a unit create one group and then execute three queries concurrently, one of which replaces the first group created. In the `success` block, it creates a group named “success”. In the `failure` block, it creates a group named “failure”. In either success or failure, it displays the XML of the last group created:

```
   <%@page language="java"  session="false"
     errorPage="../IEError.jsp"%>

   <%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
prefix="ie" %>

   <%response.setContentType ("text/xml");%>

   <ie:unit>

     <ie:webject name="Create-Group" type="GRP">
       <ie:param name="ELEMENT"    data="dbOut2=out"/>
       <ie:param name="DELIMITER"  data=":"/>
       <ie:param name="GROUP_OUT"  data="dbOut2"/>
     </ie:webject>

   <ie:parallel>

     <ie:webject name="Query-Objects" type="OBJ">
       <ie:param name="INSTANCE"  data="jdbcAdapter"/>
       <ie:param name="CLASS"     data="EMP"/>
       <ie:param name="DBUSER"    data="jdm"/>
       <ie:param name="PASSWD"    data="jdm"/>
       <ie:param name="WHERE"     data="ENAME='SMITH'"/>
       <ie:param name="GROUP_OUT" data="dbOut1"/>
     </ie:webject>

     <ie:webject name="Query-Objects" type="OBJ">
       <ie:param name="INSTANCE" data="jdbcAdapter"/>
       <ie:param name="CLASS"    data="EMP"/>
       <ie:param name="DBUSER"   data="jdm"/>
       <ie:param name="PASSWD"   data="jdm"/>
       <ie:param name="WHERE"    data="ENAME='SMITH'"/>
       <ie:param name="GROUP_OUT" data="dbOut2"/>
     </ie:webject>

     <ie:webject name="Query-Objects" type="OBJ">
       <ie:param name="INSTANCE" data="jdbcAdapter"/>
       <ie:param name="CLASS"    data="EMP"/>
       <ie:param name="DBUSER"   data="jdm"/>
       <ie:param name="PASSWD"   data="jdm"/>
```

```
    <ie:param name="WHERE"    data="ENAME='SMITH'"/>
    <ie:param name="GROUP_OUT" data="dbOut3"/>
  </ie:webject>

  </ie:parallel>

  <ie:success>

    <ie:webject name="Create-Group" type="GRP">
      <ie:param name="ELEMENT"    data="SUCCESS=success"/>
      <ie:param name="DELIMITER"  data=":"/>
      <ie:param name="GROUP_OUT"  data="success"/>
    </ie:webject>

    <ie:webject name="Object-XML" type="DSP"/>

  </ie:success>

  <ie:failure>

    <ie:webject name="Create-Group" type="GRP">
      <ie:param name="ELEMENT"    data="FAILURE=failure"/>
      <ie:param name="DELIMITER"  data=":"/>
      <ie:param name="GROUP_OUT"  data="failure"/>
    </ie:webject>

    <ie:webject name="Object-XML" type="DSP"/>

  </ie:failure>

</ie:unit>
```

For the details on the custom tags used, see the tag descriptions in the Info*Engine Custom Tag Reference on page 243.

Catching Exceptions

There are three general ways that you can code your JSP pages to catch exceptions:

- Use an error page.
- Use a `try/catch` block.
- Use the **unit** tag and nested **success** and **failure** tags (not suggested in JSPs).

Using an Error Page

By creating an error page and specifying it in the `page` directive on your JSP pages, you can catch exceptions from the page. You can designate that a page is an error page on the `page` directive by setting the **isErrorPage** attribute to `true`. For example:

```
<%@page language="java" isErrorPage="true"
    import="java.io.*,com.infoengine.util.*,com.infoengine.exception.*,
    com.infoengine.exception.fatal.*,com.infoengine.exception.nonfatal.*,
    java.util.Hashtable"%>
```

Use your error page by specifying it as the error page in the `page` directive on other pages. For example the following `page` directive sets the **errorPage** attribute to `IEError.jsp`:

```
<%@page language="java" session="false" errorPage="IEError.jsp"%>
```

An error page can be very useful because it can act as a common location for error processing to give a consistent experience when errors occur. An error page can also be written to deal with exceptions based on their type, presenting more meaningful information to the user.

Using try/catch Blocks

You can catch exceptions within specific parts of a JSP by using `try/catch` blocks within scriptlets. This is similar to handling exceptions using **unit** and **failure** tags, but allows you to more programmatically deal with exceptions. For example:

```
<%
try {
%>
<!-- webject or task invocations -->
<%
// catch statement appropriate given the contents of this block
} catch ( AdapterException ae ) {
  //handle the adapter exception
} catch ( IDPartialResultsException partial ) {
  // extract and display the partial results
}
// etc.
%>
```

Using unit Tags

You can catch exceptions within specific parts of a page by using **unit** tags on the page and including one or more `failure` blocks within the unit. In the previous section, the `failure` block provided was for all failures. In addition to having a general `failure` block, you can have `failure` blocks that catch a specific exception. The following `failure` blocks catch four different exceptions:

```
<ie:failure exception="AdapterException">
  <ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT"  data="FAILURE=AdapterError"/>
    <ie:param name="DELIMITER"  data=":"/>
    <ie:param name="GROUP_OUT"  data="failure"/>
  </ie:webject>
   <ie:webject name="Object-XML" type="DSP"/>
</ie:failure>

<ie:failure exception="IEPartialResultsException">
  <ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT"  data="FAILURE=PartialResults"/>
    <ie:param name="DELIMITER"  data=":"/>
    <ie:param name="GROUP_OUT"  data="failure"/>
  </ie:webject>
  <ie:webject name="Object-XML" type="DSP"/>
</ie:failure>

<ie:failure exception="IEInternalServiceException">
  <ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT"
              data="FAILURE=InternalServiceError"/>
    <ie:param name="DELIMITER"  data=":"/>
    <ie:param name="GROUP_OUT"  data="failure"/>
  </ie:webject>
  <ie:webject name="Object-XML" type="DSP"/>
</ie:failure>

<ie:failure exception="IEFatalException">
  <ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT"  data="FAILURE=FatalException"/>
    <ie:param name="DELIMITER"  data=":"/>
    <ie:param name="GROUP_OUT"  data="failure"/>
  </ie:webject>
  <ie:webject name="Object-XML" type="DSP"/>
</ie:failure>
```

You can also use the Throw-Exception webject to throw your own exceptions or rethrow an exception from within a page or a task. All exceptions thrown during task webject execution are automatically entered in the SERVER context group as the attributes named **exception-class** and **exception-message**. Therefore, you can use the Throw-Exception webject to rethrow and exception and its message without knowing what the exception is.

4

Info*Engine Tasks

The following topics introduce the concept of an Info*Engine task, describe how to create and use tasks, and describe the XML output that Info*Engine produces when displaying groups.

About Info*Engine Tasks and Task Rules

Info*Engine text-based documents are called "standalone tasks," and can control the retrieval and manipulation of data within your Info*Engine environment. Instead of using a custom Java application or JSP pages to perform all operations on the data (including retrieving and displaying your data), you can separate out the data retrieval and manipulation operations from the display operations using tasks.

One of the advantages of using tasks is that you can organize your code so that data operations can be done once and then used many times. The Info*Engine task compiler parses Info*Engine tasks and produces executable Java classes from them. This improves the performance of executing Info*Engine tasks by eliminating the need to parse and interpret a task each time it is called. It also facilitates embedding Info*Engine tasks in standalone Java applications and JSP pages.

The task compiler produces the executable Java classes in three basic steps:

1. It parses the source of a task and generates Java source code that implements the task.
2. It calls a Java compiler to produce an executable class from the generated Java source.
3. It calls a class loader to load and instantiate the classes produced by the Java compiler.

As a performance optimization, the task compiler retains compiled classes in a cache and avoids the first two steps whenever it can determine that a cached class is up to date. A cached class is discarded and regenerated whenever the compiler discovers that the task has been updated since the last time that the task was compiled.

Note

Due to the upgrade to the latest JSP specification, JSPs using Info*Engine substitution syntax can no longer use the `${...}` syntax. Info*Engine now supports an alternate syntax, `$(...)` that must be used instead.

All Info*Engine tasks function with either syntax of `$(...)` or `${...}`. Preexisting tasks continue to function as is and do not have to be changed.

Preexisting JSPs must be updated to use the new `$(...)` syntax or they fail to compile and do not function as expected.

Specific Task Rules

When creating tasks, you must adhere to the following rules:

- An Info*Engine task should start with a JSP `page` directive similar to the following:

  ```
  <%@page language="java" session="false"%>
  ```

  The **import** attribute on the `page` directive should also be included if additional classes are required.

- The task must specify the custom tag library used in the file. Info*Engine has three custom libraries: core, directory, and supplied libraries; only the core library and the supplied libraries are supported in tasks. To identify the library, include `taglib` directives similar to the following:

  ```
  <%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
             prefix="ie" %>
  ```

  Note

  You must specify the **uri** attribute value exactly as it is shown here. You can choose a different prefix value. However, you must use whatever value you specify as the prefix for the tags from the corresponding library.

- The task must have at least one task webject. You use the **webject** and **param** tags to specify the webject.
- Tasks cannot contain any display elements such as display webjects. If HTML tags are in a task, they are ignored.

General Task Rules

To be well-formed and valid, the Info*Engine task must follow these basic rules:

- For Info*Engine custom tags and JSP tags, you must use lowercase. For example, you must specify the **webject** tag as “webject” and not “WEBJECT” or “Webject” or even “webJecT”.
- You can use comments to document what is happening in your task or to cause the complier to skip a section of the task. If a webject is surrounded by a comment, it is not executed. Comments can be located anywhere in a task except within tags, declarations, or other comments.

  Comments begin with <!– and end with –>.

- Empty elements must be properly constructed. The trailing `/>` characters (the forward slash followed by the right angle bracket) in the task syntax indicates to the task compiler that the element is empty and no matching end-tag should be sought. For example, the **param** custom tags make use of the empty element construction.
- Additional general rules for using scriptlets, expressions, declarations, directives, and Info*Engine custom tag elements can be found in the Info*Engine Custom Tag Reference on page 243.

Authoring Info*Engine Tasks

Best Practice

When authoring an Info*Engine task it is very important to consider how you intend that task to be invoked. Info*Engine tasks might be executed in several different ways, including from other Java code using the SAK (for example, from a Windchill user interface or workflow), from a SOAP client, or from a raw HTTP client through the IE servlet. For example, if you are authoring a task that is intended to be called from a Windchill workflow you probably do not also intend for someone to be able to run that task through the IE servlet. To control how a task is executed, you can use the **access** attribute. For more information, see page Directive on page 248.

All webjects that do not directly deal with the display of information can be placed in Info*Engine tasks. That is, any webject with that has the `DSP` type exists only in a JSP page or custom application. The following standard webject types can be included in Info*Engine tasks and are known as “task webjects”:

- The GRP type, or group webjects.
- The OBJ type, or query webjects.
- The ACT type, or action webjects.
- The MGT type, or management webjects.
- The MSG type, or message webjects.
- The WES type, or Web Event Service webjects.
- The ADM type, or administrative webjects.

In addition, you can create external custom webjects (EXT) that provide custom solutions in either a JSP page or a standalone task. For more information, see Creating an External Custom Webject on page 148.

Think of a task as a script of commands executed by Info*Engine. Each command in a task is a webject. The webjects are executed in the sequence defined within the task. By default, the sequence is from the top to the bottom. The webjects within a task perform operations such as querying databases, combining and integrating data in interesting ways, performing schema translations, and creating and updating database information. For a task to be successful, it must contain at least one properly constructed webject.

The following `CreateGroup.xml` task creates a group consisting of names, street addresses, and email addresses:

```
<%@page language="java" session="false"%>

<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Create an internal Group -->

<ie:webject name="Create-Group" type="GRP">

     <ie:param name="ELEMENT"
data="NAME=Sam Johnson:ADDRESS=1234 Main St.:EMAIL=sjohnson@somewhere.com"/>

     <ie:param name="ELEMENT"
data="NAME=Harvy Anderson:ADDRESS=1234 Amber St.:EMAIL=handerson@somewhere.com"/>

     <ie:param name="ELEMENT"
data="NAME=James O'Connor:ADDRESS=775 Main St.:EMAIL="/>

     <ie:param name="ELEMENT"
data="NAME=Harvey Hampton:ADDRESS=775 Main St.:EMAIL=hhampton@somewhere.com"/>

     <ie:param name="CLASS" data="EmployeeData"/>

     <ie:param name="GROUP_OUT" data="createdgroup"/>

</ie:webject>
```

The resulting “createdgroup” group generated by this task is shown as the following table:

NAME	ADDRESS	EMAIL
Sam Johnson	1234 Main St.	sjohnson@somewhere.com
Harvy Anderson	1234 Amber St.	handerson@somewhere.com
James O’Connor	775 Main St.	
Harvey Hampton	775 Main St.	hhampton@somewhere.com

Notice that there was no `EMAIL` value specified for James O’Connor, so the corresponding table cell is empty.

Parts of a Task

Each Info*Engine standalone task should contain the following items:

- The standard JSP `page` directive.

  Although the Info*Engine task compiler only uses the **import** attribute from the `page` directive, it is good practice to include it as the first line in your task file even when no additional classes are required. You should use fully qualified class names in scriptlets or specify the classes used in scriptlets on the import statement.
- The standard JSP `taglib` directive, which declares that the Info*Engine standalone task uses custom tags defined in a tag library.

  You must put this directive before any lines that use the custom tags in the library.
- Info*Engine custom tags, which provide access to a set of custom actions that encapsulate recurring functionality.

  The custom tags provide the syntax for executing webjects and provide the structure around which you can build a task.

The previous `CreateGroup.xml` example task includes the following standard JSP directives:

```
<%@page language="java" session="false"%>
```

This is the standard `page` directive. No additional classes are required for this task.

To identify the “core” Info*Engine tag library you must include the following:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
```

This provides the general tags that can be used in the task and specifies `ie` as the required prefix that identifies the tag as belonging to the core tag library.

The previous `CreateGroup.xml` example task also includes the **webject** and **param** tags:

- The **webject** tag identifies the Create-Group webject.
- The **param** tags supply webject parameters and parameter values.

  For the Create-Group webject, the parameters define both name of the output group and the attributes and values of elements in the group.

There are many more custom tags that you can use in a task. For the description and syntax of all of the custom tags, see Info*Engine Tags on page 251.

In addition to directives and custom tags, you can include JSP scriptlets, expressions, and declarations in your Info*Engine tasks. For the details on how the Info*Engine task compiler processes scriptlets, expressions, and declarations, see their corresponding descriptions in Scriptlets on page 244, Expressions on page 245, and Declarations on page 247.

Task Creation

You can create task files using any standard editing tool. Using a tool that provides you with a Graphical User Interface (GUI) for inserting custom tags and JSP elements is helpful, but not required.

For additional information on creating tasks, see Using the Info*Engine Task Editor on page 95.

Task File Extension

You should save your task files using the `.xml` extension. Info*Engine recognizes files with this extension as tasks.

Task File Location

Info*Engine provides a directory structure under which you should save your task files. Using this directory structure allows you to execute tasks from a web browser URL and to specify a relative path to the task when executing it from within another task or from a JSP page.

Windchill is configured to look for Info*Engine tasks within the `<`*`Windchill`*`>/tasks` directory. You must save your tasks somewhere within the tasks directory. Ideally, you should create a directory hierarchy to organize your tasks, such as `<`*`Windchill`*`>/tasks/com/company/CreateGroup.xml`.

Task Execution

You can execute a standalone task in the following ways:

- Specify the task on page 92 in the Info*Engine custom **task** tag.

  You can include the **task** tag in JSP pages and in other tasks.
- Enter the task on page 93 URL in a web browser.
- Submit the task on page 94 so that it is queued for execution.
- Emit an event on page 94 that results in a task being executed.

Note

The URLs and URIs shown in this guide use the forward slash as the separator (/) in paths. Info*Engine correctly identifies all system URLs and URIs when you specify the forward slash. If you prefer to use the back slash for NT URLs and URIs, you must escape the back slash. This means that you enter two back slashes (\\) for each slash (\).

Specifying a Task in a task Tag

The **uri** attribute on the Info*Engine **task** tag names the task to execute. For example, to execute the `CreateGroup.xml` task described previously, you could include the following elements in a JSP page:

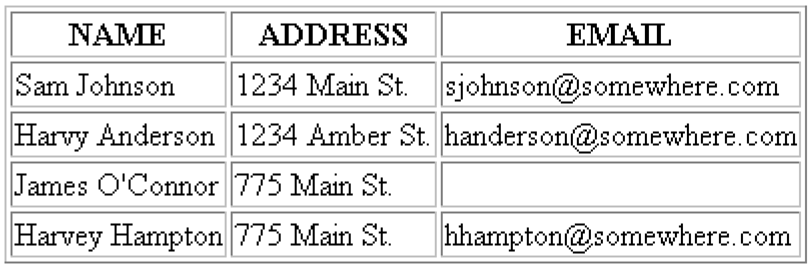
```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:task uri="/com/company/CreateGroup.xml"/>
```

In this example, the `taglib` directive identifies the Info*Engine core tag library and defines `ie` as the prefix that the tags use. The **task** tag specifies the relative path to the `CreateGroup.xml` task.

The group created by the example task is automatically available to the rest of the elements on the JSP page. These elements could further manipulate the data in the group or could display the group back to the user who initiated the execution of the JSP page. For example, assume that a user executed the following JSP page:

```
<%@page language="java" session="false"
        errorPage="IEError.jsp%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:task uri="/com/company/CreateGroup.xml"/>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"  data="createdgroup"/>
</ie:webject>
```

The `CreateGroup.xml` task executed through the **task** tag is the same task described earlier. The output from this task is the “createdgroup” group, which is then used as input to the Display-Table webject. Executing this page results in the following display:



NAME	ADDRESS	EMAIL
Sam Johnson	1234 Main St.	sjohnson@somewhere.com
Harvy Anderson	1234 Amber St.	handerson@somewhere.com
James O'Connor	775 Main St.	
Harvey Hampton	775 Main St.	hhampton@somewhere.com

Entering a Web Browser URL

Entering a task URL in the web browser is an easy way to execute the task without providing the additional coding required to execute the task through an application or JSP page.

The installer also specifies an application URL that is used as the URL prefix for requesting Info*Engine tasks. Produce the URL that contains the request to execute Info*Engine JSP pages by doing the following:

- Including the host name and application URL prefix specified when Windchill was installed. The default application URL is `Windchill`.
- Including the `/servlet/IE/tasks` prefix, which directs the servlet to the task processor.
- Specifying the path for the task that is relative to the `tasks` directory.
- Specifying any optional parameters to pass to the task.

Note

If your site is using form-based authentication, programmatic clients attempting to access the IE servlet must use the `/protocolAuth` URL prefix. For example:

```
http://<host>/Windchill/protocolAuth/servlet/IE
```

For more information about form-based authentication, see the PTC Windchill Advanced Deployment Guide.

Therefore, to execute the `com/company/CreateGroup.xml` task using the `myServer` host name and the `Windchill` application URL, specify the following URL:

```
http://myServer/Windchill/servlet/IE/tasks/com/company/CreateGroup.xml
```

The XML output displayed in the browser is similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="createdgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL>hhampton@somewhere.com</EMAIL>
  </wc:INSTANCE>
</EmployeeData>
</wc:COLLECTION>
```

Submitting Tasks and Emitting Events

If your Info*Engine environment includes the implementation of Info*Engine task queuing or the Web Event Service with a Message-Oriented Middleware (MOM) software product such as IBM MQSeries, you can execute tasks through specialized webjects that are provided for these environments. Using these webjects requires the installation and configuration of additional third-party products and also requires additional Info*Engine configuration.

Your site administrator should know if your environment includes any of these optional features. For implementation information, see the *Info*Engine Implementation Guide*. For more webject information, see Queue-Task on page 472 and Subscribe-Event on page 492.

Using the Info*Engine Task Editor

The Info*Engine Task Editor provides a streamlined method for creating tasks and JSPs without having to do a lot of typing. By selecting menu options in a Windows interface and entering information in data fields, or even by typing partial webject names, you can easily create formatted scripts that take advantage of all the current Info*Engine webjects.

Starting and Stopping the Task Editor

With Windchill running, start the Info*Engine Task Editor. You can start the Task Editor in one of the following ways:

- For Windows, if you accepted the defaults during installation, you can select **InfoEngine Task Editor** from the **Start** menu.
- You can also use the platform-appropriate startup script:
  - Windows: `<Windchill>/taskeditor/bin/modeler.bat`
  - Unix: `<Windchill>/taskeditor/bin/modeler`

Once started, the Info*Engine Task Editor attempts to look up service information.

If an Info*Engine Server and Naming Service are available, they are used to remotely invoke tasks on available task processors. The task editor can invoke tasks to supply more advanced functionality such as introspection of JDBC database tables and storage of task source in LDAP.

If more than one Info*Engine Server or Naming Service is found, you are prompted to select the ones you want to use.

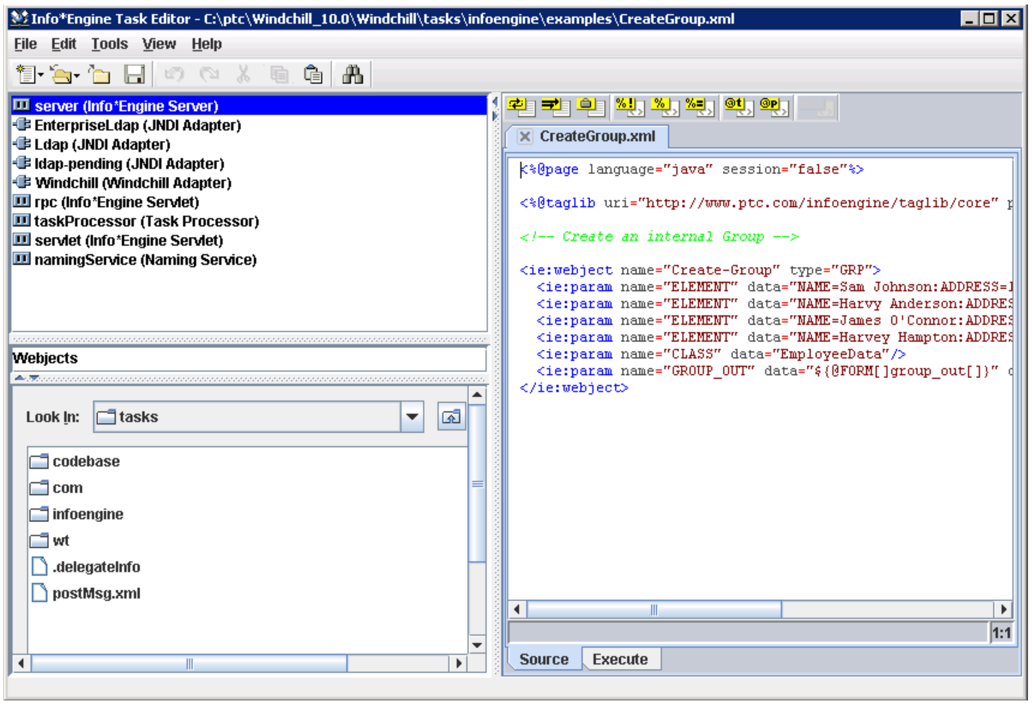
To stop Info*Engine Task Editor, simply close its window.

Navigating the Task Editor

The following sections are brief descriptions of the user interface in the Info*Engine Task Editor.

Panels

The Info*Engine Task Editor consists of four panels: editor, browse, services, and webjects.


Info*Engine Task Editor - C:\ptc\Windchill_10.0\Windchill\tasks\infoengine\examples\CreateGroup.xml
File
Edit
Tools
View
Help
Webjects
Look In:
tasks
CreateGroup.xml
1:1
Source
Execute

server (Info*Engine Server)
EnterpriseLdap (JNDI Adapter)
Ldap (JNDI Adapter)
ldap-pending (JNDI Adapter)
Windchill (Windchill Adapter)
rpc (Info*Engine Servlet)
taskProcessor (Task Processor)
servlet (Info*Engine Servlet)
namingService (Naming Service)

codebase
com
infoengine
wt
.delegateInfo
postMsg.xml

```
<%@page language="java" session="false"%>

<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" p

<!-- Create an internal Group -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Sam Johnson:ADDRESS=I
  <ie:param name="ELEMENT" data="NAME=Harvy Anderson:ADDRES
  <ie:param name="ELEMENT" data="NAME=James O'Connor:ADDRES
  <ie:param name="ELEMENT" data="NAME=Harvey Hampton:ADDRES
  <ie:param name="CLASS" data="EmployeeData"/>
  <ie:param name="GROUP_OUT" data="${@FORM[]group_out[]}" c
</ie:webject>
```

Editor

The editor is the large panel at the right, and contains the source for the active task or JSP. The tabs along the top of the panel indicate the open editor for either a task or JSP. You can switch from one editor to another by clicking the corresponding tab. The tabs at the bottom of the panel allow you either to view the source code or run the task or JSP.

Services

Webjects

The services and webject panels are to the left of the editor panel, with the services on top and their corresponding webjects below.

The services panel displays the services and adapters that are available to the current configuration you chose during setup. Services are labeled with the receptacle icon ▥. Adapters are labeled with the plug-in icon ⊣▤.

Browse

The browse panel is located in the lower left, and helps you navigate to saved tasks.

Note

You can customize the size and availability of panels by clicking and dragging the panel borders.

Menus

The following menus appear:

File Menu	
New	Create a new Info*Engine task or JSP. The new file appears in the editor panel. (Keyboard shortcuts: press CTRL+T for a new task or CTRL+J for a new JSP.)
Open	Open an Info*Engine task or JSP from the local file system. The file appears in the editor panel. (Keyboard shortcut: press CTRL+O.)
Close	Close the Info*Engine task or JSP currently being edited. (Keyboard shortcut: press CTRL+F4.)
Close all	Close all Info*Engine tasks or JSPs currently being edited.
Save	Save the Info*Engine task or JSP currently being edited to their source location. (Keyboard shortcut: press CTRL+S.)
Save All	Save all Info*Engine tasks or JSPs currently being edited to their source location. (Keyboard shortcut: press CTRL+SHIFT+S.)
Save as	Save the active Info*Engine task or JSP in the editor panel to a new location on the local file system.
Exit	Exit the Info*Engine Task Editor

Edit Menu	
Undo	Undo the last edit action. (Keyboard shortcut: press CTRL+Z.)
Redo	Redo the last undone edit action. (Keyboard shortcut: press CTRL+Y.)
Cut	Remove the currently selected text and place it in the system clipboard. (Keyboard shortcut: press CTRL +X.)
Copy	Copy the currently selected text to the system clipboard. (Keyboard shortcut: press CTRL+C.)

Edit Menu	
Paste	Paste information from the system clipboard into the active Info*Engine task or JSP in the editor panel. (Keyboard shortcut: press CTRL+V.)
Select all	Select all the text of the active Info*Engine task or JSP in the editor panel. (Keyboard shortcut: press CTRL+A.)
Find	Open the **Find/Replace** window to locate text within the active Info*Engine task or JSP in the editor panel. (Keyboard shortcut: press CTRL+F.)
Go to	Open a **Go To Line** window to move the cursor to a specific line within the active Info*Engine task or JSP in the editor panel. (Keyboard shortcut: press CTRL +G.)
Match braces	If the cursor is adjacent to a brace character, this action moves the cursor to the corresponding open or close brace character. (Keyboard shortcut: press CTRL +].) If no corresponding open or close brace is found, then a beep sounds. If the cursor is not adjacent to a brace character, then this action does nothing. The following are considered brace characters: (), [], {}, and <>.

Tools Menu	
Edit tag	Open the **Editor** window for the tag in which the cursor is placed in the editor panel.
Preferences	Open the **Preferences** window, where you can modify how text appears in the editor and control tag insight preferences.

View Menu	
Toolbars	Show or hide the editor, **File**, **Edit**, or **Find** toolbars.
Services/Webjects	Show or hide the services and webjects panel.

Help Menu	
Contents	Open the **Help** window. (Keyboard shortcut: press F1.)
About	Open the **About** window.

Toolbar Icons

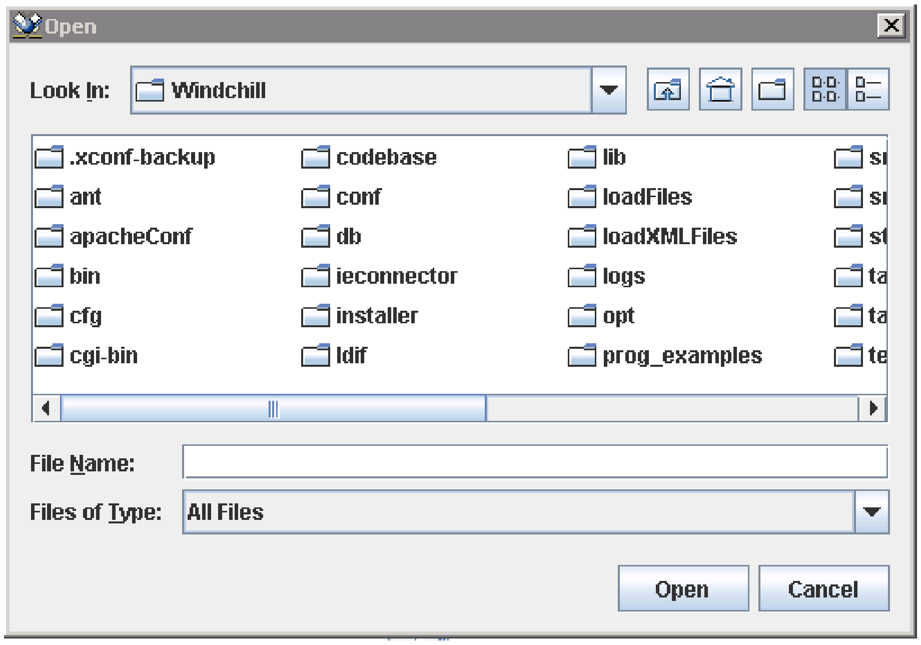
Icon	Action	Description
	Insert a `forEach` block	Click this icon to insert a `forEach` block into your Info*Engine task or JSP source. When dropping a `forEach` block into your source, a window opens allowing you to enter values for the **groupIn** and **groupOut** attributes.
	Insert a `parallel` block	Click this icon to insert a `parallel` block into your Info*Engine task or JSP source.
	Insert a **task** tag	Click this icon to insert a **task** tag into your Info*Engine task or JSP source. When dropping a task into your source, a window opens allowing you to enter values for the URI and processor attributes. You are also able to add parameters to be sent along with the request to execute the task.
%!	Insert a declaration block	Click this icon to insert a declaration block into your Info*Engine task or JSP source.
%	Insert a scriptlet block	Click this icon to insert a scriptlet block into your Info*Engine task or JSP source.
%=	Insert a JSP expression block	Click this icon to insert an expression block into your Info*Engine task or JSP source.
@t	Insert JSP `taglib` directive	Click this icon to insert a `taglib` directive into your Info*Engine task or JSP source.
@P	Insert JSP `page` directive	Click this icon and then click in the editor panel to insert a `page` directive into your Info*Engine task or JSP source.
edit	Edit tag	Clicking the edit icon edit with the cursor in a tag displays the associated tag editor for you to make the desired changes. When finished, click **OK** to apply the changes. Tag editors are available for several different types of tags, such as **webject**, **task**, **param**, **forEach**, JSP `page` directive, and JSP `taglib` directive.

Opening an Existing Info*Engine Task or JSP

You can use the Info*Engine Task Editor to open and edit existing task and JSP files, regardless of whether they were created with the Task Editor.

Use the browse panel to navigate to an existing task. If necessary, click and drag the panel borders to view the browsing toolbar options.

Alternatively, select **File ▸ Open ▸ Open File**. The **Open** window opens, where you can browse for task and JSP files:



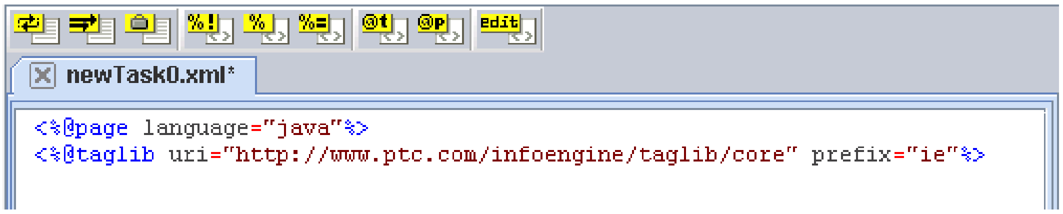
Select the file you want and click **Open**, and the selected file appears in the editor panel.

Creating an Info*Engine Task or JSP

With the Info*Engine Task Editor, you can create tasks and JSPs that meet the same criteria as explained in Info*Engine JSP Pages on page 63 with minimal code entry.

New Task

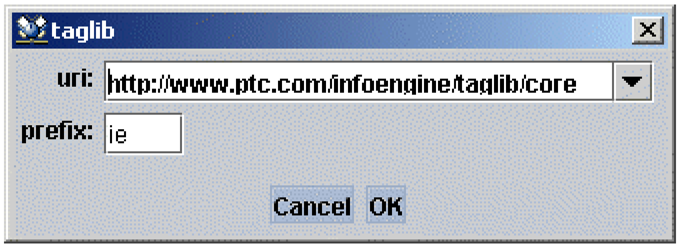
To create a new task, select **File** ▸ **New** ▸ **New task**. This opens an editor panel for a new task. Scriptlets for `page` and `taglib` directives are already added according to settings chosen in **Preferences**.



```
<%@page language="java"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
```

You can change the content of editable tags or directives by placing the cursor anywhere in the scriptlet and clicking the edit icon Edit.

For example, placing the cursor in the `taglib` directive and clicking Edit opens the **taglib** window:

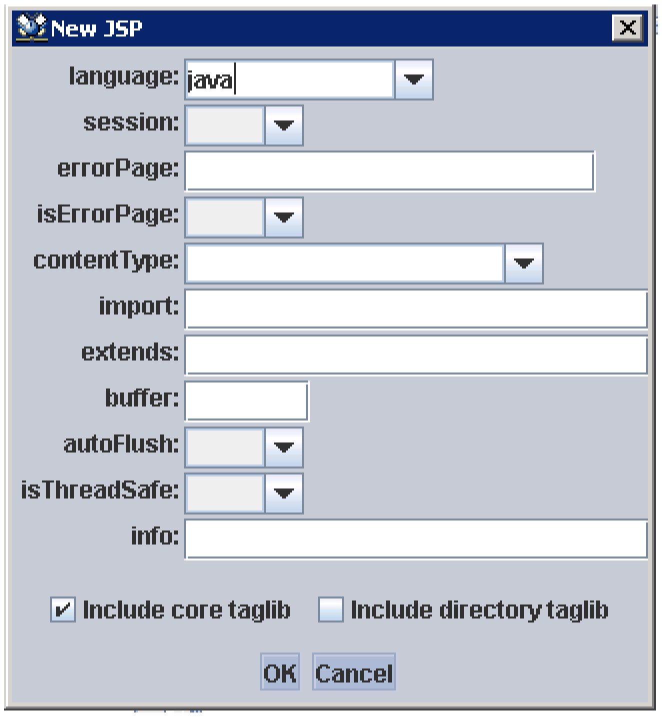


uri: http://www.ptc.com/infoengine/taglib/core

prefix: ie

New JSP

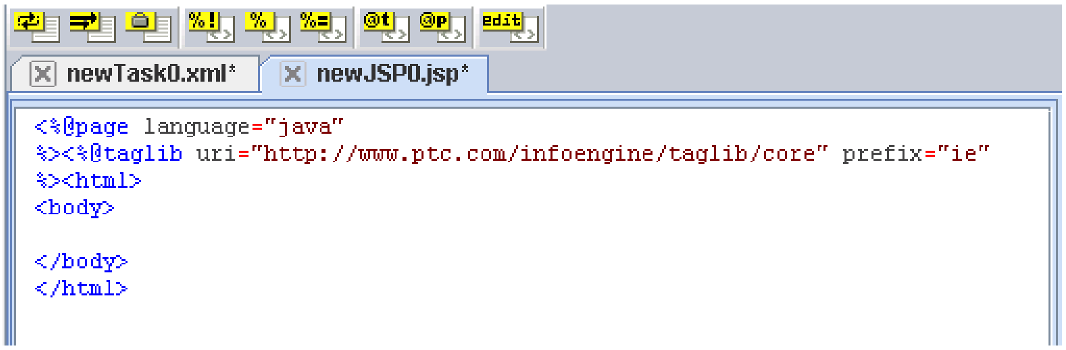
To create a new JSP, select **File ▸ New ▸ New JSP**. This opens the **New JSP** window, with fields and drop-down menus for you to enter and select `page` and `taglib` directives:



language:	java
session:	
errorPage:	
isErrorPage:	
contentType:	
import:	
extends:	
buffer:	
autoFlush:	
isThreadSafe:	
info:	

[✓] Include core taglib
[ ] Include directory taglib

Click **OK** and the appropriate coding is placed at the top of a new JSP editor:



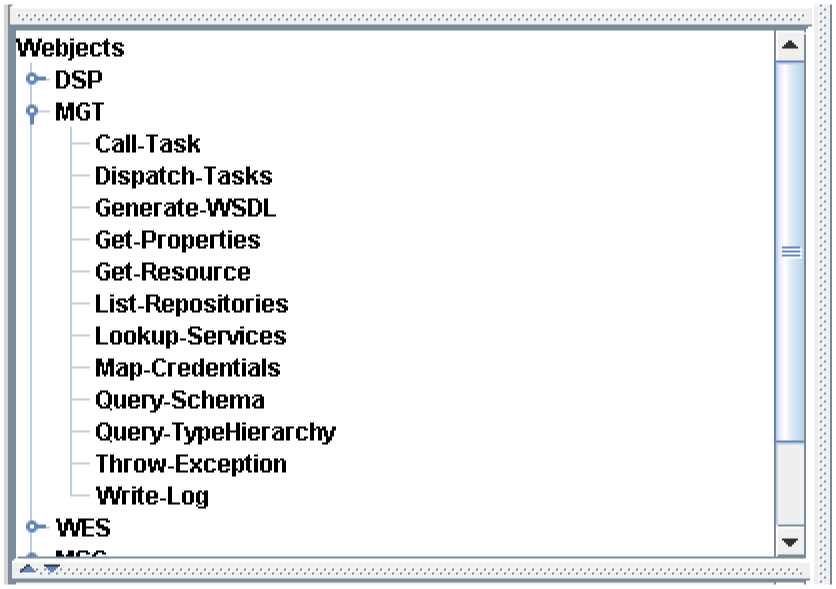
```
<%@page language="java"
%><%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><html>
<body>

</body>
</html>
```

Entering and Editing Info*Engine and JSP Tags

The Info*Engine Task Editor also helps you to customize several aspects of your Info*Engine tasks and JSP tags.

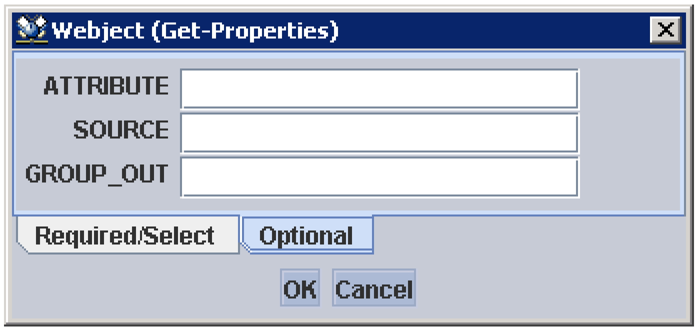
Webjects and the Webject Editor

When a service or adapter is selected in the top panel, the lower panel displays the webjects it supports (if any). The webjects are organized by type. If an editor panel is open, you can browse to the desired webject, select it, and then click in the editor panel to insert that webject.



Webjects
- DSP
- MGT
  - Call-Task
  - Dispatch-Tasks
  - Generate-WSDL
  - Get-Properties
  - Get-Resource
  - List-Repositories
  - Lookup-Services
  - Map-Credentials
  - Query-Schema
  - Query-TypeHierarchy
  - Throw-Exception
  - Write-Log
- WES

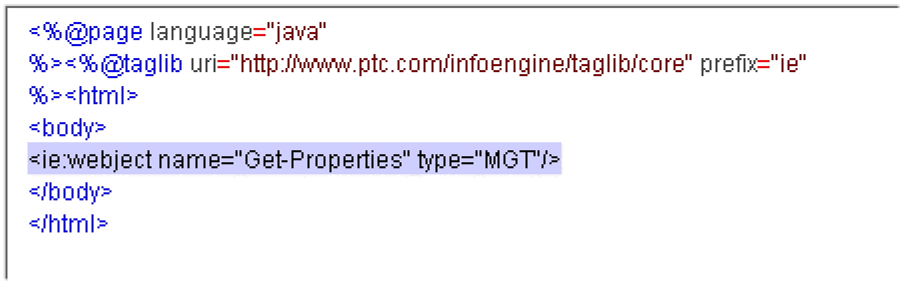
A window specific to that webject opens and displays the parameters supported by that webject. The parameters are organized on two tabs: **Required/Select** and **Optional**:



Webject (Get-Properties)

ATTRIBUTE	
SOURCE	
GROUP_OUT	

Fill in the desired parameter values and click **OK** to insert the webject into your Info*Engine task or JSP source.



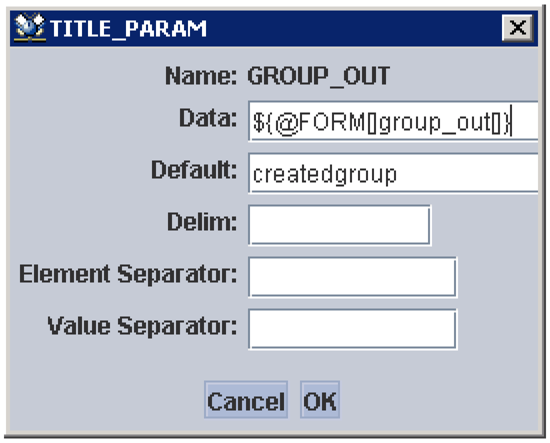
```
<%@page language="java"
%><%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><html>
<body>
<ie:webject name="Get-Properties" type="MGT"/>
</body>
</html>
```

Tag and Webject Editors

There are three ways to open a tag editor window:

- Right-click anywhere within a tag, webject, or directive. If the selected element has editing functionality available, an **Edit** message appears. Click the message to open an editor window.
- Click anywhere within a tag, webject, or directive, and then open the **Tools** menu. If the selected element has editing functionality available, the **Edit tag** option is clickable and opens an editor window.
- Click anywhere within a tag, webject, or directive, and click the edit icon .

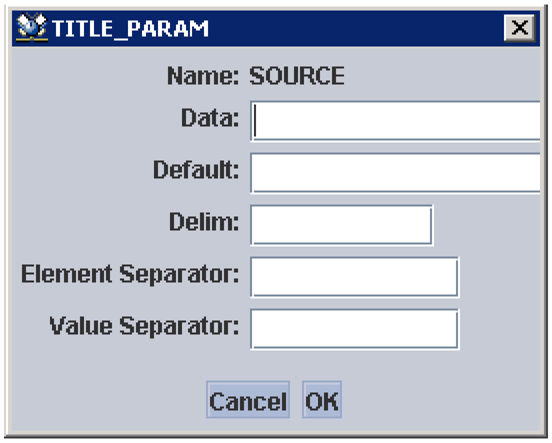
For some tags and webjects, there is a second level of editing available. When working with webjects, you can double-click the name of a parameter and a window opens, allowing you to edit some parameter attributes. Doing so displays a parameter editor where you can edit other parameter attributes such as **default**, **delim**, **elementSeparator** and **valueSeparator**:



TITLE_PARAM

Name:	GROUP_OUT
Data:	${@FORM[]group_out[]}
Default:	createdgroup
Delim:	
Element Separator:	
Value Separator:	

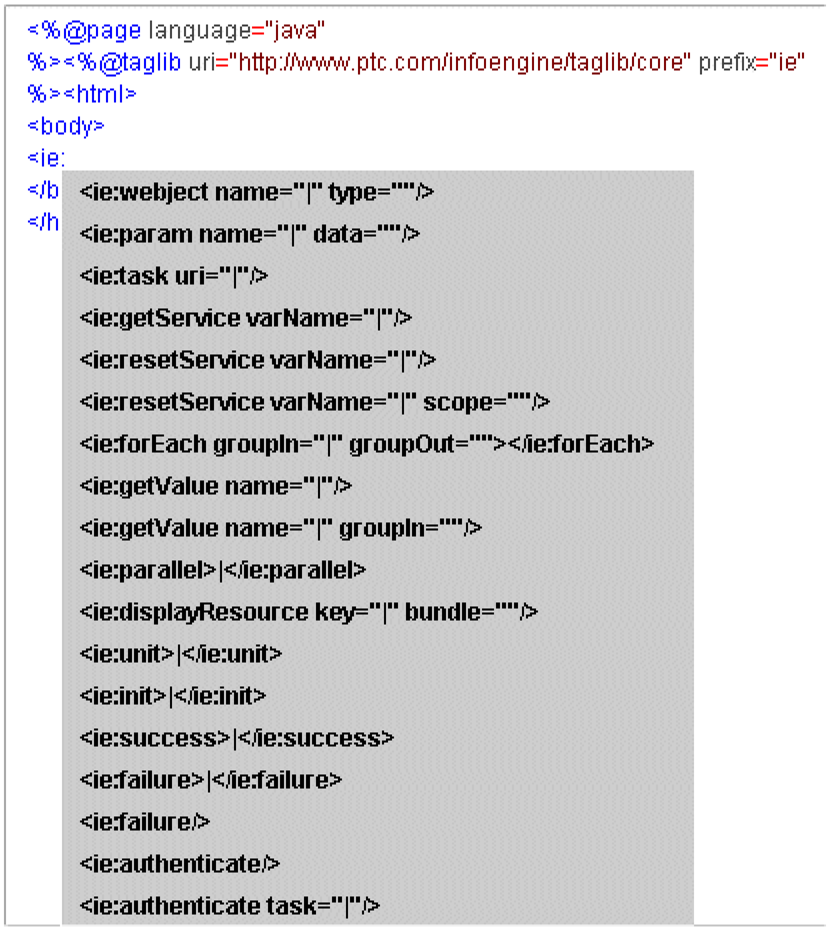
Similarly, when working with a **task** tag editor window, you can double-click within the **Name** field and a new window opens to edit parameters:



Editor windows are available for several different types of tags, such as: **webject**, **task**, **param**, **forEach**, the JSP `page` directive, and the JSP `taglib` directive. They are also available for most webjects.

Tag Insight

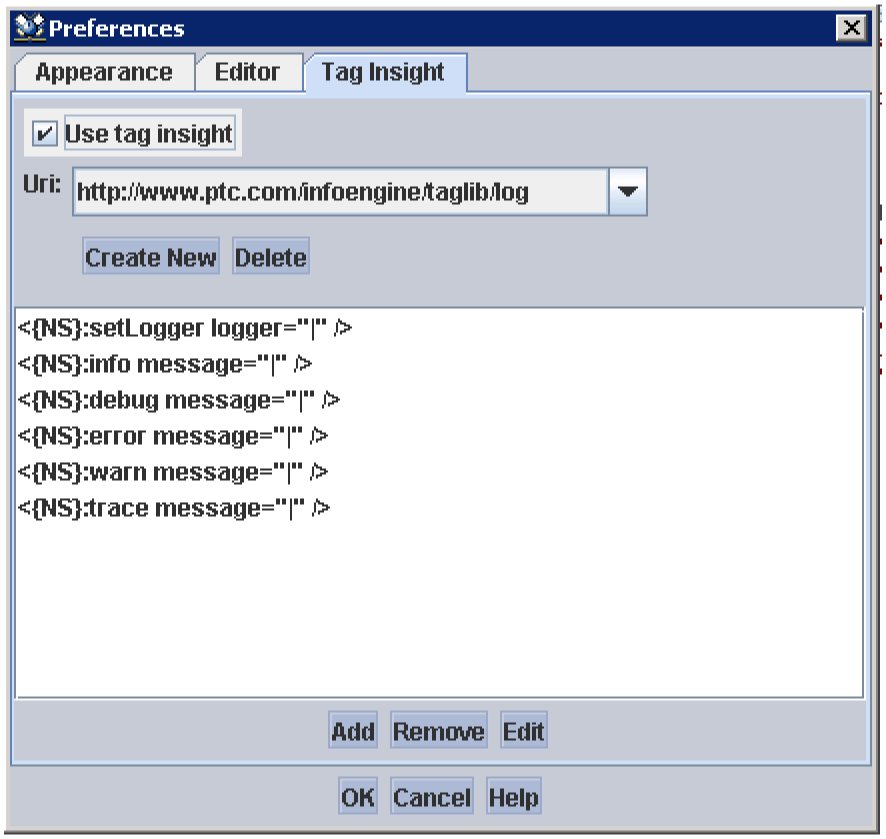
The text editor can make suggestions relative to the text you are typing. If you are typing the start of a tag that it recognizes, it displays a drop-down list of possible tags from which you can select.



```
<%@page language="java"
%><%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><html>
<body>
<ie:
</b
</h
```

```
<ie:webject name="|" type=""/>
<ie:param name="|" data=""/>
<ie:task uri="|"/>
<ie:getService varName="|"/>
<ie:resetService varName="|"/>
<ie:resetService varName="|" scope=""/>
<ie:forEach groupIn="|" groupOut=""></ie:forEach>
<ie:getValue name="|"/>
<ie:getValue name="|" groupIn=""/>
<ie:parallel>|</ie:parallel>
<ie:displayResource key="|" bundle=""/>
<ie:unit>|</ie:unit>
<ie:init>|</ie:init>
<ie:success>|</ie:success>
<ie:failure>|</ie:failure>
<ie:failure/>
<ie:authenticate/>
<ie:authenticate task="|"/>
```

By default, Tag Insight is available for several types of tags, such as JSP tags and Info*Engine core, supplied, and directory tags. You can create your own Tag Insight on the **Tag Insight** tab of the **Preferences** window:



Uri: http://www.ptc.com/infoengine/taglib/log

```
<{NS}:setLogger logger="|" />
<{NS}:info message="|" />
<{NS}:debug message="|" />
<{NS}:error message="|" />
<{NS}:warn message="|" />
<{NS}:trace message="|" />
```

Understanding XML Output for Info*Engine Groups

Info*Engine maintains all generated groups as serializable Java objects. These objects can be easily manipulated by custom Java applications, JSP pages, and tasks. Then, when there is a request to display a group, Info*Engine generates an XML document containing the group.

By default, the Info*Engine generated XML document contains the following parts:

- The document declaration at the beginning of the document.
- The `wc:COLLECTION` element, which is the document root for all Info*Engine generated XML documents.
- The root group element, which is named through the CLASS parameter on webjects that produce output groups. For example, if the CLASS parameter is set to “Employees,” then the root group element start tag includes

`<Employees>`. In addition to the element name, the group element start tag always includes the NAME attribute. The value of the NAME attribute comes from the group name specified on the GROUP_OUT parameter of the webject that created the group.

- The `wc:INSTANCE` elements, which identify Info*Engine objects (rows) within a group. Each Info*Engine object contains attributes that are coded as sub-elements of a `wc:INSTANCE` element.

These parts are described in detail in the following sections. For examples of Info*Engine XML output, see the example XML tasks documented in Group Webjects on page 360.

In addition, Info*Engine can also generate XML output that includes the metadata that is part of the group, element, or attribute data.

Document Declaration

XML documents must begin with an XML declaration that specifies the version of XML being used and the character encoding for this XML document. The current version of XML defined by the W3C, and used by Windchill, is 1.0. In addition, Info*Engine supports UTF8 for the character encoding. All XML generated by Info*Engine therefore begins with the following XML declaration:

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

An XML “namespace” is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. A namespace allows an XML author to uniquely specify a vocabulary to prevent naming collisions. Info*Engine defines the namespace `http://www.ptc.com/infoengine/1.0`, and associates the namespace prefix `wc:` to all Info*Engine defined elements and attributes.

```
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
```

The document entity, also known as the document root, serves as the root of the entity tree and a starting-point for an XML processor. Info*Engine defines the `wc:COLLECTION` element as the document root for all Info*Engine generated XML documents.

Info*Engine Groups

The serializable objects used for manipulating Info*Engine groups stores the values of the webject CLASS and GROUP_OUT parameters that were specified when the group was created. The value of the CLASS parameter is used to define the element name for the root element of the group data, and the value of the GROUP_OUT parameter is used as the NAME attribute for this element. For example, if a CLASS parameter is set to “DemoGrp” and the GROUP_OUT parameter is set to “salesGrp,” then Info*Engine generates the following XML document elements:

```
<?xml version="1.0" encoding="UTF-8" ?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<DemoGrp NAME="salesGrp" TYPE="Object" STATUS="0">
</DemoGrp>
</wc:COLLECTION>
```

If the webject CLASS parameter is undefined or omitted, then Info*Engine uses “Unknown-Class-Name” as the element name. Though it is not technically an error to omit the CLASS parameter, the task writer should always include the CLASS parameter. Omitting the CLASS parameter makes the XML document vague. For example the following XML elements define two groups of information: “employees” and an unnamed group. In the example, it is more difficult to understand what data the unnamed group represents:

```
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<Employees NAME="group1" TYPE="Object" STATUS="0"/>
</Employees>
<Unknown-Class-Name NAME="group2" TYPE="Object" STATUS="0"/>
</Unknown-Class-Name>
</wc:COLLECTION>
```

The group element contains two other attributes: TYPE and STATUS. The TYPE attribute indicates the group type. For data groups, this value is always `Object`. The STATUS attribute indicates the completion status for the generation of this group. A status of zero (0) indicates that no errors occurred. A non-zero value indicates that data is incomplete due to some error. Although the status values for groups no longer need to be checked, the STATUS attribute remains for backward compatibility. Info*Engine currently throws an exception whenever a webject cannot generate the requested group.

Info*Engine Objects

An Info*Engine group can contain one or more objects. In the XML document, each object is identified by the `<wc:INSTANCE>` start tag and `</wc:INSTANCE>` end tag. Objects can contain zero or more attributes, where each attribute can contain data. Each Info*Engine attribute corresponds to a XML sub-element that appears within the `wc:INSTANCE` element.

For example, assume that the following XML document represents a single Info*Engine group delimited by the `<EMP>` and `</EMP>` tags, which contains a single object. The object is delimited by the `<wc:INSTANCE>` and `</wc:INSTANCE>` tags:

```
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EMP NAME="empGrp" TYPE="Object" STATUS="0">
```

```
<wc:INSTANCE>
<EMPNO>7934</EMPNO>
<ENAME>MILLER</ENAME>
<JOB>CLERK</JOB>
<MGR>7782</MGR>
<HIREDATE>1982-01-23 00:00:00.0</HIREDATE>
<SAL>1300</SAL>
<COMM/>
<DEPTNO>10</DEPTNO>
</wc:INSTANCE>
</EMP>
</wc:COLLECTION">
```

In the example, you can see the sub-element tags for the Info*Engine attributes and can determine that the object has the following attributes:

```
EMPNO
ENAME
JOB
MGR
HIREDATE
SAL
COMM
DEPTNO
```

Metadata in XML Output

Metadata is information about normal application data that is contained in a group, element, or attribute. For example, metadata can provide additional qualifying information such as data type information. It could also provide information about relationships between the elements in a group or between the attributes in an element.

Software that interfaces with Windchill can generate metadata. For example, Windchill attribute type information is stored as metadata. This metadata is automatically passed along with the Info*Engine groups that are created. In addition, you can set metadata using the Set-Metadata webject.

By default, metadata is not included when XML output is generated through the Display-XML webject. To include metadata in your XML output, you can use the FULL mode on this webject.

In the XML output that includes metadata, metadata is nested between `<wc:Meta>` start tags and `</wc:Meta>` end tags.

Nesting Tasks

Although any number of webjects can be included within a task, there are times when repetitive tasks are performed. Candidates for task nesting include any general task that might be used under multiple conditions. By using dynamic substitution (which is described in Dynamic Parameter Value Substitution on page 44), you can easily construct tasks that can be used multiple times.

You can nest tasks by specifying the Info*Engine custom **task** tag within a task.

task Tag Rules for Nesting Tasks

When working with the Info*Engine **task** tag, the following rules apply:

- You can nest any number of tasks within another task, and nested tasks can contain other nested tasks.
- The task file being referenced can also contain any of the task webjects and any of the custom tags that are available to tasks.
- You should always provide exception processing within your task to catch errors when nested tasks fail.

The syntax and complete description of the **task** tag is described in task Tag on page 275.

Simple Nested Task Example

The following task includes a **task** tag that executes another task named `QueryTask.xml`:

```
<%@page language="java" session="false"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<process name="Embedded Task Example with exception handle">

<ie:unit>

  <ie:task uri="infoengine/examples/QueryTask.xml"/>

  <!-- If task fails, print message to stderr and
       throw exception-->

 <ie:failure>
    <%
    System.err.println("FAILURE when calling QueryTask.xml");
    %>
```

```
      <ie:webject name="ThrowException" type="MGT"/>
    </ie:failure>

  </ie:unit>
  </process>
```

Assume the `QueryTask.xml` file contains the following:

```
<%@page language="java" session="false"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE"  data="com.myHost.Adapter"/>
  <ie:param name="CLASS"     data="salesemp"/>
  <ie:param name="WHERE"     data="()"/>
  <ie:param name="GROUP_OUT" data="sales"/>
</ie:webject>
```

For additional examples of webjects that can be used in nested tasks, see Dynamic Parameter Value Substitution on page 44.

5

Info*Engine Packages

The following topics discuss creating, installing, and managing packages to help organize your LDAP directory, as well as how to ascribe properties to your packages.

About Info*Engine Packages

A package is a set of files to be delivered into your task root, and (if necessary) associated LDAP infrastructure entries required for an application to function. Utilizing packages allows these files and directory entries to be installed, updated, and uninstalled from a system in sets rather than individually. Using packages also greatly simplifies the installation and maintenance of federated applications requiring updates to the LDAP.

Task packages may contain any files to be delivered into your task root, most commonly XML tasks, XSL style sheets, and tag library descriptor files. Task package filenames end with the extension `.ptctar`.

Providing LDAP Directory Information

Some Info*Engine-based applications like web services, Windchill JCA applications, or other federated applications may require LDAP directory entries. While this kind of information can be provided using LDIF files, the LDIF files may get very complicated and must be maintained and updated as an application expands or changes.

With packages, this LDAP information can be provided by specifying some basic properties either directly in the task source or within `.delegateInfo` properties files.

Property Definitions

The following properties can be specified in either a `.delegateInfo` properties file or a task. For details on the specification format for either method, refer to the appropriate section below:

repositoryType
: Specifies the repository type that type identifiers and delegates are created under. This property should not be specified unless the application belongs to a product that has defined a repository type other than “com.ptc.windchill.” Installation tools determine the appropriate repository type based on the repository the package is being installed under. Installations typically have a repository type of “com.ptc.windchill.”

typeId
: Specifies the type identifier under which delegates are created. If this property is not specified, then the installation tool determines the type identifier by the task directory hierarchy. For example, a task residing in the “org/myOrg/Person” directory is assumed to have a type identifier of “org.myOrg.Person.”

delegateName

Specifies the name of a task delegate. This property should only be specified in a task, and then only if the task filename (minus the `.xml` extension) differs from the name of the command delegate being created.

installDelegate

Specifies whether a single delegate or group of delegates should be installed in the LDAP directory. The value of this property is boolean. This property is useful for utility tasks that are never directly called by an application, and thus requires no delegate entry in the LDAP directory. While creating a delegate for a task that is never externally invoked does not interfere with the directory structure, avoiding the creation of such delegates keeps the LDAP directory contents cleaner. Specify `FALSE` to prevent delegates from being created. The default for this property is `TRUE`.

Property Definitions in a Properties File

Properties can be defined in a special Java properties file named `.delegateInfo`. Any properties specified in a `.delegateInfo` file are inherited by all entries below that level in the directory tree. Subdirectories can also have their own `.delegateInfo` files that would supersede the properties specified in a higher level directory.

Properties can be specified in the `.delegateInfo` file as in any standard Java properties file. For example:

```
typeId=WCTYPE|wt.part.WTPart
```

Property Definitions in a Task

Properties defined within a task supersede properties specified in a properties file. Tasks with these defined properties are sometimes referred to as “self-describing” tasks.

Since Info*Engine tasks are not Java properties files, the properties must be specified in a different manner:

- Properties within Info*Engine tasks must be specified inside of a special comment section that begins with the `com.infoengine.delegate.def` identifier. This identifier informs the Package Manager and installation tools that properties are specified.
- Each property definition must be preceded by an at symbol (`@`). Any lines in the comment section that do not begin with `@` are registered as a comment and used in the description of an installed delegate.
- The property name and value must be separated by a single space. The following is an example of a self-describing task comment section:

```
<!--com.infoengine.delegate.def
```

```
this is the delegate description
@delegateName get-name
-->
```

Using the Package Manager

The Package Manager utility is a simple Java application that creates, installs, and uninstalls task packages. The main Java class for the Package Manager is `com.infoengine.administration.packaging.UI`. The Package Manager can be started from a windchill shell. For example:

```
windchill "--javaargs=-DpropFile=<Windchill>/codebase/wt.properties"
com.infoengine.administration.packaging.UI
```

The classpath can be specified either as the CLASSPATH environment variable, or on the Java command line. An example of the classpath specified on the Java command line is:

```
java -classpath classpath com.infoengine.administration.packaging.UI
```

where *classpath* is the classpath noted above.

Starting the Package Manager

To start the Package Manager, use the following procedure:

1. Execute one of the following commands:

   ```
   java com.infoengine.administration.packaging.UI
   ```

   or

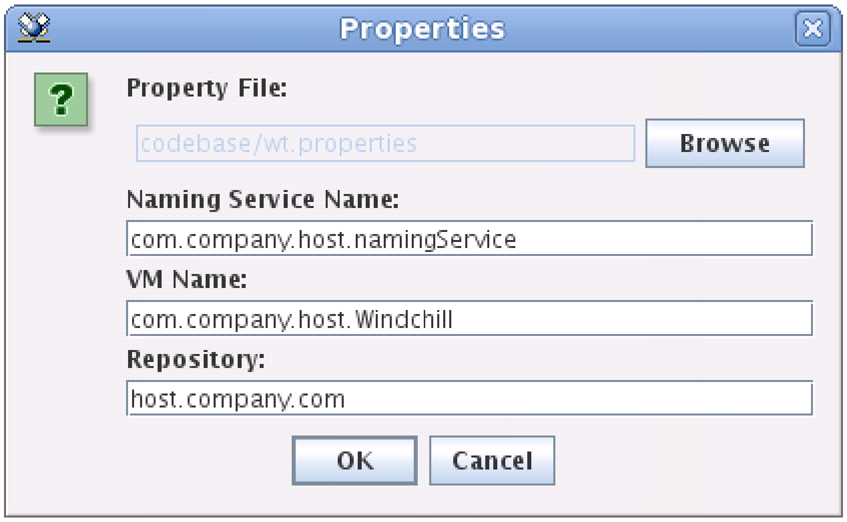
   ```
   java -DpropFile=(<Windchill>)/codebase/wt.properties
   com.infoengine.administration.packaging.UI
   ```

The **Properties** window opens.

Note

Specifying the `propFile` property on the command line avoids requiring you to browse to the location of your property file in the following step.

2. Verify or enter the information in the **Properties** window fields as needed.



Properties

Property File:

codebase/wt.properties

Naming Service Name:

com.company.host.namingService

VM Name:

com.company.host.Windchill

Repository:

host.company.com

- **Property File**—If you did not specify the `propFile` property on the command line, click **Browse** and browse to your properties file.
- **Naming Service Name**—The runtime service name of your naming service. This service must exist in the LDAP directory, and must be properly configured.
- **VM Name**—The name of your Windchill virtual machine. This value must be the runtime service name of your Windchill service.
- **Repository** — The name of your installation repository. The repository must exist in our directory relative to your naming service search base. Typically this value is your fully qualified host name.

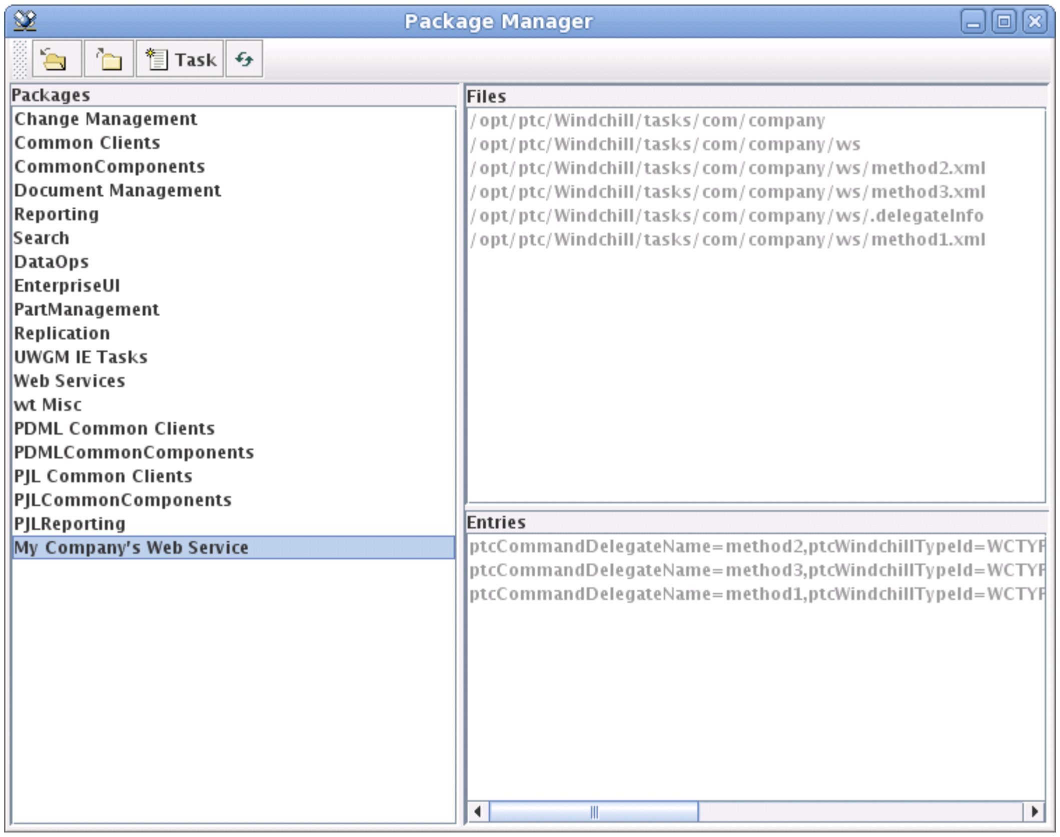
3. Click **OK**.

If there is a problem with any of the values, the **Properties** window is presented again with the problem value displayed in red.

If all supplied values are acceptable, the Package Manager opens.

Navigating the Package Manager

The Package Manager provides a simple interface, allowing you to select the files that are part of the package, and to supply basic package information.



Change Management
Common Clients
CommonComponents
Document Management
Reporting
Search
DataOps
EnterpriseUI
PartManagement
Replication
UWGM IE Tasks
Web Services
wt Misc
PDML Common Clients
PDMLCommonComponents
PJL Common Clients
PJLCommonComponents
PJLReporting
My Company's Web Service

/opt/ptc/Windchill/tasks/com/company
/opt/ptc/Windchill/tasks/com/company/ws
/opt/ptc/Windchill/tasks/com/company/ws/method2.xml
/opt/ptc/Windchill/tasks/com/company/ws/method3.xml
/opt/ptc/Windchill/tasks/com/company/ws/.delegateInfo
/opt/ptc/Windchill/tasks/com/company/ws/method1.xml

ptcCommandDelegateName=method2,ptcWindchillTypeId=WCTYP
ptcCommandDelegateName=method3,ptcWindchillTypeId=WCTYP
ptcCommandDelegateName=method1,ptcWindchillTypeId=WCTYP

The **Package Manager** window consists of a toolbar and three panels.

The panels list the following information:

- **Packages**—Displays any packages installed on your system. Hovering your cursor over the package name displays the package description (if one is available).
- **Files**—Displays all files installed as a part of the selected package. All file names in this list are absolute.
- **Entries**—Displays all of the LDAP directory entries created as a result of the package installation.

Typically, the first time you start the Package Manager all three panels are empty, as no packages have been installed.

The icons on the toolbar include:

- **Install Package**
- **Uninstall Package**

Note

If you have not selected an installed package from the **Packages** list, then the uninstall package icon is not enabled.

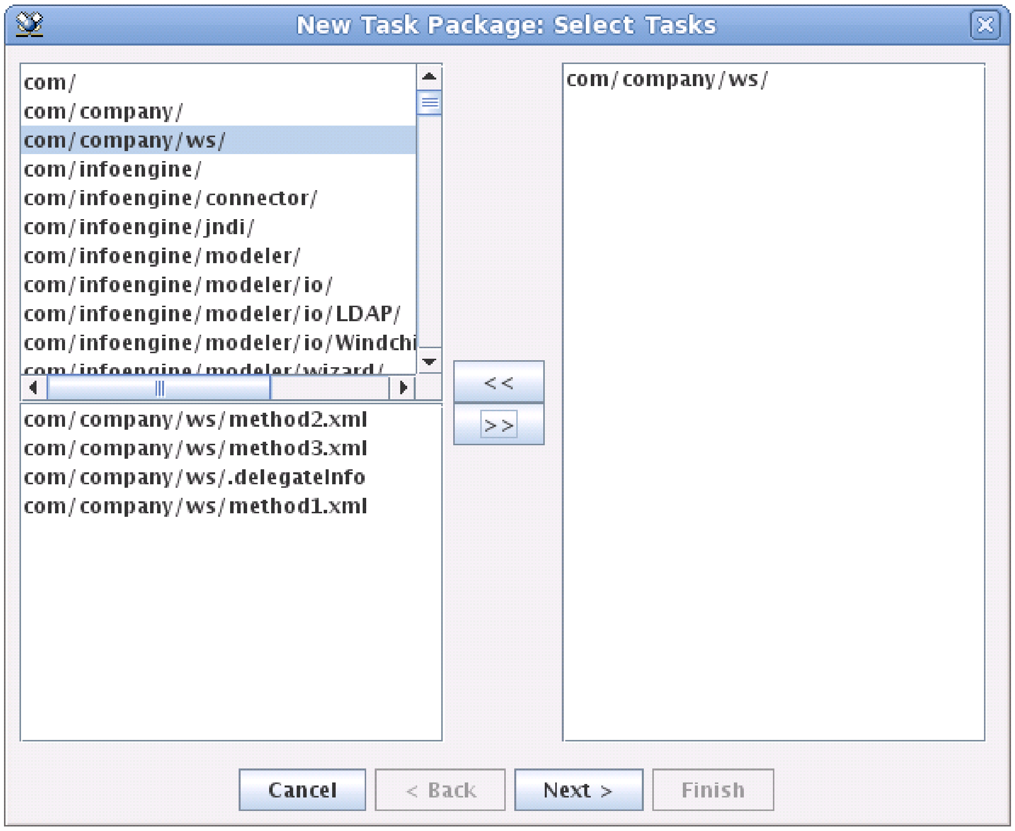
- **New Task Package**

Creating Packages

To create a package, use the following procedure:

1. Start the package manager. For more information, see Starting the Package Manager on page 116.
2. To create a task package, click **New Task Package**.

   The **New Task Package: Select Tasks** window opens.



com/
com/company/
com/company/ws/
com/infoengine/
com/infoengine/connector/
com/infoengine/jndi/
com/infoengine/modeler/
com/infoengine/modeler/io/
com/infoengine/modeler/io/LDAP/
com/infoengine/modeler/io/Windchi
com/infoengine/modeler/wizard/

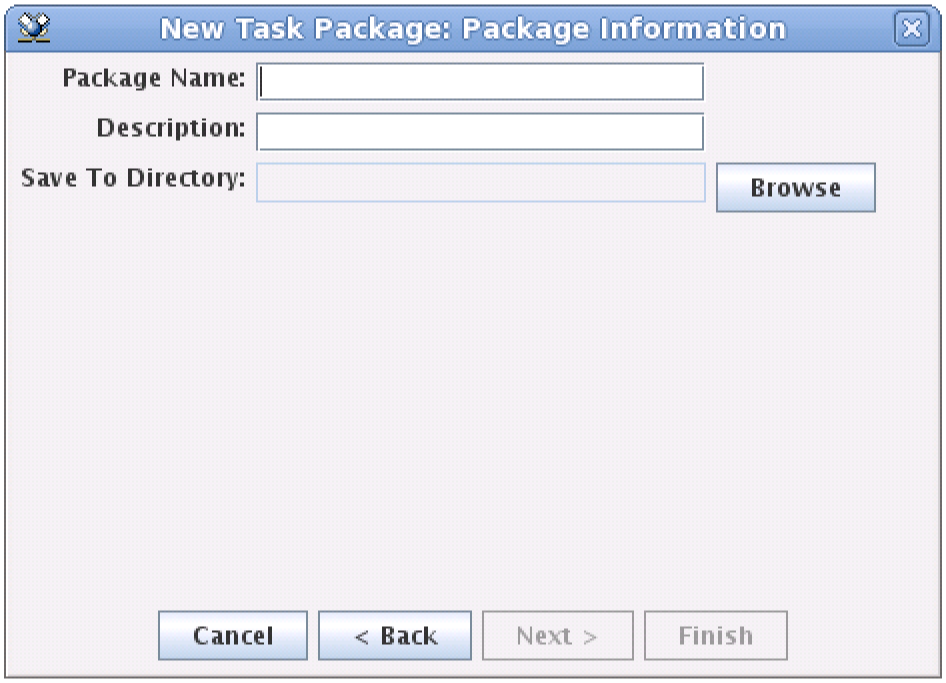
com/company/ws/method2.xml
com/company/ws/method3.xml
com/company/ws/.delegateInfo
com/company/ws/method1.xml

com/company/ws/

The **New Task Package: Select Tasks** window contains three panels:

- The upper left panel lists all directories that can be selected from the root of the appropriate file system (either the task root or the root of the configuration specification hierarchy).
- The lower left panel lists files that can be found within the directory selected in the upper left list.
- The panel on the right lists the files or directories to be packaged.

3. Select the directories or files to be packaged:
   - To package an entire directory and its contents, select the directory from the list on the upper left and click the add button >> to move the directory name to the list on the right.
   - To package a specific file and not an entire directory, select the appropriate directory in the list on the upper left, then select the appropriate file in the list on the lower left. Click the add button >> to move the file to the list on the right.
   - If you have selected a file or directory that you do not want packaged, select it in the list on the right, and click the remove button << to remove it from the package list.
4. Click **Next** to open the **New Task Package: Package Information** window.



5. In the **New Task Package: Package Information** window, specify the identification and location information for the package:
   - In the **Package Name** field, enter the package name. The package name should be descriptive, and can contain spaces and any characters. The value entered in this field is used for the installed label of the package as displayed in the Package Manager, and, in a compressed form, for the package filename. For the filename, any characters that are non-alphanumeric are removed. For example, if you enter the following value in the **Package Name** field when creating a task package:

     ```
     My Company's Web Service
     ```

     then the package filename is:

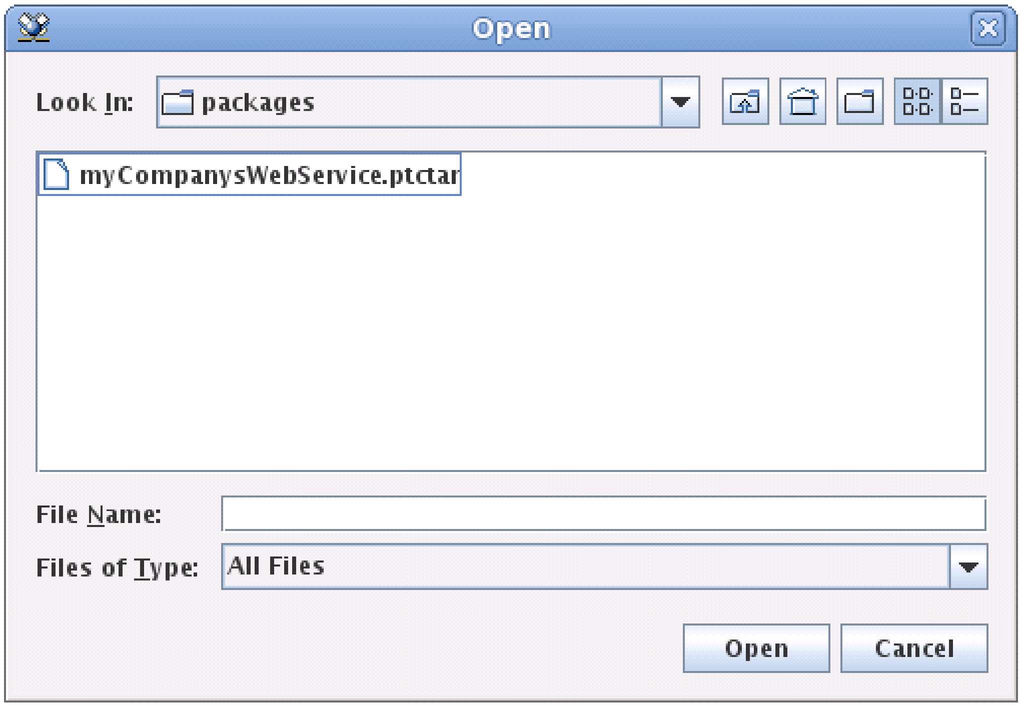
     ```
     myCompanysWebService.ptctar
     ```
   - In the **Description** field, enter a description of the package. Any value entered in this field displays when you hover your mouse pointer over the package name on the Package Manager.
   - Next to the **Save To Directory** field, click **Browse** to select the directory where the package is to be saved.
6. Click **Finish** to create the package.

Installing Packages

To install a package, use the following procedure:

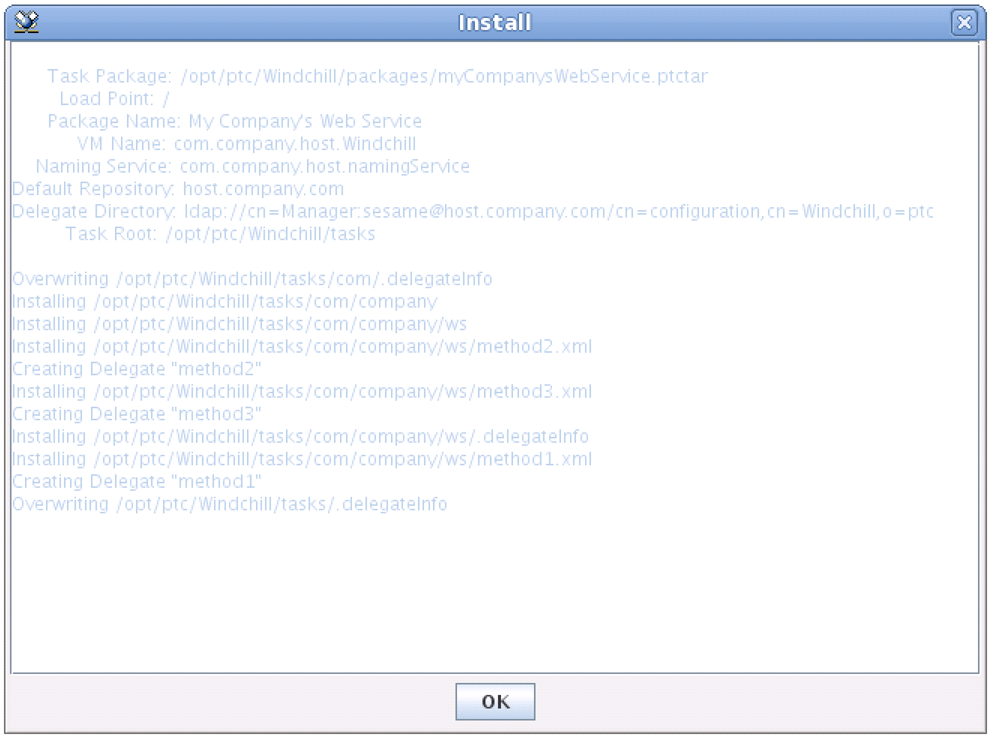
1. Start the Package Manager. For more information, see Starting the Package Manager on page 116.
2. Click **Install Package** .

The **Open** window opens.



3. Browse to the package you want to install.
4. Select the package and click **Open**.

The **Install** window opens, reporting on the progress of the installation. Any problems that occur during installation are registered in the progress report.



```
      Task Package: /opt/ptc/Windchill/packages/myCompanysWebService.ptctar
        Load Point: /
      Package Name: My Company's Web Service
           VM Name: com.company.host.Windchill
    Naming Service: com.company.host.namingService
Default Repository: host.company.com
Delegate Directory: ldap://cn=Manager:sesame@host.company.com/cn=configuration,cn=Windchill,o=ptc
         Task Root: /opt/ptc/Windchill/tasks

Overwriting /opt/ptc/Windchill/tasks/com/.delegateInfo
Installing /opt/ptc/Windchill/tasks/com/company
Installing /opt/ptc/Windchill/tasks/com/company/ws
Installing /opt/ptc/Windchill/tasks/com/company/ws/method2.xml
Creating Delegate "method2"
Installing /opt/ptc/Windchill/tasks/com/company/ws/method3.xml
Creating Delegate "method3"
Installing /opt/ptc/Windchill/tasks/com/company/ws/.delegateInfo
Installing /opt/ptc/Windchill/tasks/com/company/ws/method1.xml
Creating Delegate "method1"
Overwriting /opt/ptc/Windchill/tasks/.delegateInfo
```

5. When the installation is complete, click **OK**.

Uninstalling Packages

To uninstall a package, use the following procedure:

1. Start the Package Manager. For more information, see Starting the Package Manager on page 116.
2. Click **Uninstall Package** .

   The **Uninstall** window opens, reporting on the progress of the uninstall. Success or failure of the uninstall is indicated in the progress report. If the package is not successfully uninstalled, then the package name remains listed in the **Package Manager** and contains only the contents that could not be successfully removed from the system.
3. When the uninstall is complete, click **OK**.

6

Credentials Mapping

The following topics describe how to use the credentials mapping function to authenticate and then manage users.

Authentication through Credentials Mapping

Credentials mapping (also known as authentication mapping) relies upon the fact that the web server or servlet has authenticated the user already. Then, given an authenticated username, Info*Engine obtains a map that defines the usernames and credentials that are to be sent to adapters or sent to the JMS MOM (if implemented at your site) on behalf of the authenticated user.

With a credentials mapping mechanism in place, Info*Engine can dynamically add authentication parameters to these webjects through a site-defined credentials mapping task or a set of credentials files.

The following rules apply to credentials mapping:

- If the author of a task or JSP page explicitly specifies DBUSER and PASSWD parameters on a webject, those parameters take precedence over any other authentication information that might be available.
- If DBUSER and PASSWD parameters are not explicitly specified, Info*Engine attempts to provide values for them from the credentials mapping information.

If no credentials mapping information is available, or if the credentials mapping does not provide valid DBUSER and PASSWD values for the JMS MOM or the adapter to which a webject is being routed, Info*Engine does not send any DBUSER or PASSWD values. In this case, the adapter obtains default values from its configuration properties, and the Info*Engine messaging software and the Web Event Service obtains default values from their configuration properties. If no default values are set, Info*Engine attempts anonymous access.

Credentials Mapping for Adapters

Credentials mapping for adapters works using the following process:

- Windchill is pre-configured to include a credentials mapping task. This task is configured through the Windchill adapter using the `wt.federation.task.mapCredentials` property value, which is set to `/wt/federation/MapCredentials.xml`. The contents of this task are dynamically generated, starting with the following file:

  ```
  <Windchill>/tasks/wt/federation/
  MapCredentials.xml.template
  ```

  using the property values found in `site.xconf`.

  Windchill's LDAP access for user and group information relies on this credentials mapping task, and can provide credentials to adapters based on whether the current user is an administrator or a regular user. If you would like to modify mapped credentials, you can set properties in `site.xconf`. For more information, read the contents of `MapCredentials.xml.template` and see Managing the Credentials Task on page 129.

Note

If you need to customize this task you should do so by modifying the `MapCredentials.xml.template` file. You should also keep a backup copy, because patch or maintenance release installation may overwrite your customization.

- Your Info*Engine administrator sets the configuration property named `.credentialsMapper` (which defines the mapping task and enables credentials mapping), and optionally sets the `.credentialsTimeToLive` and `.credentialsFiles` properties (which indicate how long the information should remain cached and if there are additional files that have mapping information in them).
- When Info*Engine is called to parse and execute a JSP page or a task that accesses an adapter, it checks to see if a credentials mapping task has been defined. If it discovers that one has been defined, it executes the specified task before executing the JSP page or task originally passed to it. The output group produced by the credentials mapping task is saved as a context group named **Auth-Map**.
- When Info*Engine encounters a webject that must be routed to an adapter, it checks the webject to see if DBUSER and PASSWD parameters have been explicitly specified. If they have not been specified, it uses the value of the webject INSTANCE parameter as a key to find DBUSER and PASSWD values in the **Auth-Map** context group. If values are found, it adds them to the webject as if they had been specified explicitly by the author of the task. Otherwise, the webject is routed to the adapter unmodified.

Creating a Credentials Mapping Task

Each element of the **Auth-Map** context group produced as the output group from the credentials mapping task must contain three attributes named INSTANCE, DBUSER, and PASSWD. The value of INSTANCE identifies the adapter to which the element applies. The values of DBUSER and PASSWD provide the authentication information to be passed to the adapter in cases where explicit DBUSER and PASSWD parameters are not specified on an adapter webject.

When a credentials mapping task executes, all normal context information is available to it. For example, the SERVER context group is available. If a request sent by a browser to Info*Engine has been authenticated by the web server, then the SERVER group contains an attribute named **Auth-User**. This attribute specifies the username that was authenticated by the web server. The credentials mapping task can use that username as a key to obtain user-specific authentication information from one or more adapter-accessible information systems. It can then create an output group from that authentication information.

The following credentials mapping task example creates a group that contains all valid username and INSTANCE name combinations with corresponding DBUSER and PASSWD values. From this group, it then selects only those elements that contain the authenticated username. These elements then make up the **Auth-Map** group:

```
<%@page language="java" session="false"%>

<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Create a group from specified parameters -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="AuthGroup"/>
  <ie:param name="CLASS"     data="AuthorizationRecord"/>
  <ie:param name="DELIMITER" data=":"/>
  <ie:param name="ELEMENT"   data="USERNAME=abc:INSTANCE=com.myHost.jdbcScott:
DBUSER=scott:PASSWD=tiger"/>
  <ie:param name="ELEMENT"   data="USERNAME=abc:INSTANCE=com.myHost.jdbcAdapter:
DBUSER=abc:PASSWD=abc123"/>
  <ie:param name="ELEMENT"   data="USERNAME=xyz:INSTANCE=com.myHost.myGateway:
DBUSER=xyz:PASSWD=xyz123"/>
  <ie:param name="ELEMENT"   data="USERNAME=test:INSTANCE=com.myHost.myGateway:
DBUSER=mno:PASSWD=mno123"/>
</ie:webject>

<!-- Select subset for actual Auth-Map group based on authenticated user -->

<ie:webject name="Subset-Group" type="GRP">
  <ie:param name="GROUP_IN"    data="AuthGroup"/>
  <ie:param name="FILTER"      data="USERNAME='$(SERVER[]AUTH-USER[0])'"/>
  <ie:param name="CASE_IGNORE" data="TRUE"/>
  <ie:param name="GROUP_OUT"   data="Auth-Map"/>
</ie:webject>
```

If this task runs when the authenticated user is “abc,” then the resulting elements in the **Auth-Map** group would be:

USER-NAME	INSTANCE	DBUSER	PASSWD
abc	com.myHost.jdbcScott	scott	tiger
abc	com.myHost.jdbcAdapter	abc	abc123

Then, using the webject INSTANCE parameter, Info*Engine determines which DBUSER and PASSWD parameter values to add.

Creating Credential Mapping Files

Credential mapping files provide a way of setting an initial DBUSER and PASSWD parameter for individual users. Each file name is the name of a user. In the file for a specific user, enter one or more lines, where each line has the following format:

`instance`:*`dbuser`*:*`passwd`*

where:

`instance` specifies the name of an Info*Engine adapter.

`dbuser` specifies the username to be set in the DBUSER parameter of the webject.

`passwd` specifies the password that corresponds with the username. It is the value to be set in the PASSWD parameter of the webject.

The resulting file contains the usernames and passwords that can be used to access information systems.

The path to a directory containing credentials files can be configured using the `wt.property` of `wt.federation.mapCredentials.files`. If this property is not set, credentials mapping files are not used.

To activate credentials mapping through credentials mapping files, you must validate the request information by setting a secret in the task processor `.secret.text` or `.secret.text2` property.

If both the `wt.federation.mapCredentials.files` and the `wt.federation.task.mapCredentials` properties are set, file-based mapping is performed first, then the credentials mapping task is executed. This allows some base or default mapping information to be specified using files then augmented or overridden by the task.

Managing the Credentials Task

The `MapCredentials.xml` file is used to specify a username and password for a specific Info*Engine adapter. This file is updated and managed using the xconfmanager utility, and should not be manually edited. If manually edited, when the propagation operation is performed the xconfmanager overwrites those values using values obtained from the `site.xconf` file. For more information about the xconfmanager utility, see the *PTC Windchill Specialized Administration Guide*.

There are two variations of the property used to set username and password credentials. The `mapcredentials.admin.adapters` property is used when Windchill accesses the adapter on behalf of a Windchill administrator (a user with administrative privileges). And `mapcredentials.nonprivileged.adapters` is used when Windchill accesses an adapter on behalf of a user who does not have administrative privileges.

Using the xconfmanager utility, you can commands similar to the following to set these properties:

```
xconfmanager -p -s "mapcredentials.admin.adapters=<adapter name>^
                 <privileged User>^<privileged user password>"
 -t "codebase/WEB-INF/mapCredentials.txt"
```

To set credentials for a Windchill administrator, use the following command:

```
mapcredentials.admin.adapters=<unique name of the adapter>^<username>^<password>
```

To set credentials for a user without Windchill administration privileges, use the following command:

```
mapcredentials.nonprivileged.adapters=<unique name of the adapter>^<username>^<password>
```

Both variations of this property are multivalued, and no default is defined. Use the `-add` command in the xconfmanager to add additional adapter definitions, and use —`set` to remove them.

For example, during installation the following xconfmanager commands are executed to define the credentials for the various JNDI adapters:

```
xconfmanager -p --set "mapcredentials.admin.adapters=com.ptc.ptcnet.LDAP^cn=manager,
                 0=ptc^admin"
-t "codebase/WEB-INF/mapCredentials.txt"

xconfmanager -p --add "mapcredentials.admin.adapters=com.ptc.ptcnet.EnterpriseLdap^
             cn=manager^admin"
-t "codebase/WEB-INF/mapCredentials.txt"

xconfmanager -p --add "mapcredentials.admin.adapters=com.ptc.ptcnet.Ldap-Pending^
                     cn=manager^admin"
-t "codebase/WEB-INF/mapCredentials.txt"
```

To add non-privileged adapters:

```
xconfmanager -p --set "mapcredentials.nonprivileged.adapters=enterpriseAdapter1^
                   cn=nonprivuser1,o=ptc^password1"
-t "codebase/WEB-INF/mapCredentials.txt"
```

To remove an adapter:

```
xconfmanager -p --remove "mapcredentials.nonprivileged.adapters=enterpriseAdapter1^
           cn=nonprivuser1,o=ptc^encrypted.enterpriseAdapter1.cn=nonprivuser1,o=ptc"
-t "codebase/WEB-INF/mapCredentials.txt"
```

To reset the **MapCredentials** task, clearing all definitions of a specified type:

```
xconfmanager -p --reset "mapcredentials.nonprivileged.adapters"
-t "codebase/WEB-INF/mapCredentials.txt"

xconfmanager -p --reset "mapcredentials.admin.adapters"
-t "codebase/WEB-INF/mapCredentials.txt"
```

Credentials Mapping for MOMs

If your site has installed and configured one or more MOMs, the credentials mapping that is set up for accessing information systems can also be used for accessing MOMs. However, the message and Web Event Service webjects that access the MOM do not use INSTANCE parameters. When the SERVICE parameter is supplied on a webject, its value is used as a key into the **Auth-Map** group. If you have configured a default MOM, then Info*Engine provides the following as substitute INSTANCE parameter values:

Pseudo INSTANCE	Description
`com.infoengine.msg`	Value for message webjects (type=MSG)
`com.infoengine.wes`	Value for Web Event Service webjects (type=WES)
`com.infoengine.jms`	Value for either message or Web Event Service webjects

By setting up three pseudo INSTANCE parameter values, Info*Engine allows your site the flexibility of setting up the following accesses to your MOM through credentials mapping:

- Use the `com.infoengine.jms` pseudo INSTANCE parameter value to set a username and password for general access to the MOM.
- Use the `com.infoengine.msg` and `com.infoengine.wes` pseudo INSTANCE parameter values to override the general access for specific access through either message or Web Event Service webjects.

As is also true with the adapter webjects, specifying DBUSER and PASSWD parameter values on message and Web Event Service webjects overrides any settings provided through credentials mapping or through property settings for default usernames and passwords.

If a username and password is not provided on a webject and one cannot be determined through the credentials mapping set up at your site or through default values that can be set in MSG and WES properties, then an anonymous connection is attempted.

The following example credentials mapping task expands on the previous mapping task to create a group that contains all valid username and INSTANCE name combinations, including one pseudo INSTANCE name for general MOM access. From this group, it selects only those elements that contain the authenticated username. These elements then make up the **Auth-Map** group.

```
<%@page language="java" session="false"%>

<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Create a group from specified parameters -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="AuthGroup"/>
  <ie:param name="CLASS"     data="AuthorizationRecord"/>
  <ie:param name="DELIMITER" data=":"/>
  <ie:param name="ELEMENT"   data="USERNAME=abc:INSTANCE=com.myHost.jdbcScott:
DBUSER=scott:PASSWD=tiger"/>
  <ie:param name="ELEMENT"   data="USERNAME=abc:INSTANCE=com.myHost.jdbcAdapter:
DBUSER=abc:PASSWD=abc123"/>
  <ie:param name="ELEMENT"   data="USERNAME=xyz:INSTANCE=com.myHost.myGateway:
DBUSER=xyz:PASSWD=xyz123"/>
  <ie:param name="ELEMENT"   data="USERNAME=test:INSTANCE=com.myHost.myGateway:
DBUSER=mno:PASSWD=mno123"/>
  <ie:param name="ELEMENT"   data="USERNAME=abc:INSTANCE=com.infoengine.jms:
DBUSER=mom1:PASSWD=gen456"/>
<ie:/webject>

<!-- Select subset for actual Auth-Map group based on authenticated user -->

<ie:webject name="Subset-Group" type="GRP">
  <ie:param name="GROUP_IN"    data="AuthGroup"/>
  <ie:param name="FILTER"      data="USERNAME='$(SERVER[]AUTH-USER[0])'"/>
  <ie:param name="CASE_IGNORE" data="TRUE"/>
  <ie:param name="GROUP_OUT"   data="Auth-Map"/>
<ie:/webject>
```

If this task runs when the authenticated user is “abc,” then the resulting elements in the **Auth-Map** group would be:

USER-NAME	INSTANCE	DBUSER	PASSWD
abc	com.myHost.jdbcScott	scott	tiger
abc	com.myHost.jdbcAdapter	abc	abc123
abc	com.infoengine.jms	mom1	gen456

Then, user “abc” has the username “mom1” and password “gen456” to gain access to the MOM whenever the user tries executing a message or Web Event Service webject.

7

Custom Tag Libraries

The following topics discuss implementing and using an Info*Engine custom tag library, and provides information on using expression language within custom tags.

About the Custom Tag Library

The Info*Engine task compiler supports the concept of custom tag libraries. Custom tag libraries for Info*Engine tasks are similar to those supported by JSP. When you implement a custom tag library, you are using sets of Info*Engine tasks, which can be organized in two ways: either within a common subdirectory, or organized as federated task delegates associated with a type identifier. This allows you to easily write reusable tag functionality without requiring you to write or compile Java code. If more advanced functionality is necessary, then you can implement your custom tag library as a set of Java classes instead. These Java classes are described by an XML document called a Tag Library Descriptor (TLD), which associates the classes with the appropriate tags for use by Info*Engine tasks.

Both task and Java tag library implementations allow you to specify whether an attribute is required, and whether or not a given attribute supports expressions or not. It is important to consider what is appropriate for each attribute exposed when writing a tag library. If an attribute is marked as `required`, then your tag implementation does not need to spend time validating its presence. Info*Engine does not compile a task that uses a tag which does not supply a required attribute value.

If an attribute supports expressions, the value supplied can still be a value other than null. More importantly, however, is whether or not an attribute should support embedded expressions. While expression evaluation is not overly expensive, logically you should mark attributes that are hard-coded to not support expressions. Even where an attribute is marked as supporting expressions, the Info*Engine task compiler examines the attribute value to decide whether or not it needs to evaluate expressions at runtime to avoid unnecessary overhead. For more information, see Expression Language Support on page 142.

Task-Based Tag Library Implementation

Tag libraries can be written using rules and procedures similar to those used when authoring Info*Engine tasks. Each task (or command delegate if using a federated type identifier) can be exposed as a single tag. The parameters documented using the taskdoc comments of a task are exposed as the attributes of the tag. Metadata on each parameter can be used to control whether a given attribute is required, and whether or not it supports expressions. By default, attributes support expressions and are not required. Parameters that override the defaults may appear as follows:

```
<!--com.infoengine.soap.rpc.def
This is the tag description.

@param boolean hardCoded No Expressions {rtexp:false}
@param string required Is required {required:true}
@return INFOENGINE_GROUP ${output}
```

```
-->
```

In this example, metadata on task comments is supplied in curly braces (`{ }`), and the name is separated from the value by a colon (:). In tags, the `@return` comment is not used explicitly, but should be included for completeness and to enhance the usefulness of the task.

Specifying a parameter with a type of INFOENGINE_GROUP provides a group from the VDB of the parent task to the VDB of the tag. As such, a tag has access to the contents of the calling task's VDB, but only to those portions of it that are explicitly required. Array input is added to the tag's @FORM input as a multivalued parameter. Upon input, the tag inherits all of the calling task's @SERVER and @AUTH-MAP context information, but not @FORM. Following tag invocation, the tag's VDB is merged back into the calling task's VDB.

The calling task's input and output streams are also associated with whichever tags it calls, so it is possible for a task-based tag to consume input and produce output.

A calling task and the tags it calls can share other information through the "tasklet context." The tasklet context refers to a pair of maps containing data. Each task gets its own local map to store variable data, and that map is backed by the **IeContext** object of the overarching request. Each Info*Engine-based request has its own instance of **IeContext** (`com.infoengine.util.IeContext`) that can hold contextual information specific to a given request. However, locally-scoped variables in the context are only visible to the task in which they are defined. Sharing information between tasks (and also possibly tags) within the same scope requires request-scoped variables. For more information on the tasklet context, see Expression Language Support on page 142.

For example:

```
<c:set var="shared" value="${value}" requestScope="true" />
```

Each tag based on an Info*Engine task implicitly supports a child tag that has the same prefix, is named "param," and has **name** and **data** attributes. This tag can be used to supply undocumented parameters to a tag (for example, supplying parameter for a supporting adapter to an existing adapter-based SOAP task). The exception to this rule would be a situation in which the task-based tag libraries already contain a tag named "param". In this situation the **param** task takes precedence, and nested **param** tags are not allowed within that tag library.

Tag Library Organization in the Task Root

There are two primary ways to organize a task-based tag library. The first is to simply create a subdirectory within the task root to hold your tags, and then specify the path of that directory (relative to the task root) as the URI when using your tag library. For example:

```
<%@taglib uri="/ext/myorg/tags" prefix="my"%>
```

```
...
<my:myTag hardCoded="true" required="${expression}" />
...
```

Each tag within the subdirectory is available as its own tag, and each of its parameters are exposed as its attributes.

Tag Library Organization Using a Federated Type Identifier

The second method of organizing an Info*Engine task tag library is using a federated type identifier. In this situation, the value you specify on the URI attribute is the type identifier with which your tags are associated. This method of organization has the advantage of allowing tags to support federation and hierarchy climbing. If you choose to use hierarchy climbing, a given tag within a tag library can map to multiple tasks, depending on the input given at task delegation. When using this mechanism, every tag implicitly supports an **input** attribute that allows the caller to supply the input group used to drive task delegation (the exception being tags that already explicitly define an **input** attribute). Both the unique federation identifier (UFID) and federated type identifier on the input group are implicitly qualified if not explicitly supplied.

If necessary, you can override the type hierarchy climber implementation within your task's context. Prior to task delegation, you can create a key by replacing "`.`" with "`_`" in the federated type identifier, appending `_climber` (for example, `wt_fc_Persistable_climber`), and by placing the climber class name in the tasklet context. As a result, the new value for the climber implementation is discovered in the tasklet context and then used during task delegation. The default is to use a Windchill type hierarchy climber, which should be suitable for the vast majority of usages. For an example, see Dispatch-Tasks on page 417.

As an example, if you wrote a **create** method associated with `wt.part.WTPart`, it could be called as follows:

```
<%@taglib uri="wt.part.WTPart" uri="prt" %>
...
<prt:create name="myPart" />
...
```

As previously mentioned, however, this mechanism can be more effective if used with type hierarchies, which would allow for not only dynamic discovery of the appropriate delegate for a task, but also routing to an appropriate system. For example:

```
<%@page language="java"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%@taglib uri="/org/myorg/util/" prefix="util"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/log" prefix="log"%>
<%@taglib uri="/com/infoengine/tlds/iejstl.tld" prefix="c"%>
<%@taglib uri="WCTYPE|wt.fc.Persistable" prefix="obj"%>
...
```

```
<!-- create a document in two systems (assumed tag implementations) -->
<util:query type="wt.org.WTPrincipal"
      where="uid='${@SERVER[0]auth-user[0]}'"
      attribute="name,cabinetRef" groupOut="me" />

<!-- create the task delegate input -->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT"

data="CLASS=wt.doc.WTDocument~name=${@FORM[]docName[]}~folderingInfo.cabinet=
${me[0]cabinetRef[0]}"/>
  <ie:param name="DELIMITER" data="~" />
  <ie:param name="GROUP_OUT" data="doc"/>
</ie:webject>

<!-- create the document in this system -->
<log:debug message="creating document ${doc[0]name[0]} locally" />
<obj:Create input="${doc}" />
<!-- create the document in another system -->
<log:debug message="creating document ${doc[0]name[0]} remotely" />
<c:set var="${doc[0]obid}" value="@otherdomain.myorg.org" />
<obj:Create input="${doc}" />
...
```

Note

Some of the tags above do not actually exist, but are used here for illustration purposes only.

Java-Based Tag Library Implementation

In some cases, writing a Java-based tag library is more appropriate. For example, if you need to integrate with other Java-based APIs that are currently unavailable through Info*Engine, and as a result would require scriptlet code to be embedded within your tasks (or task-based tags). When writing a Java-based tag library implementation, there is a set of interfaces and support base classes that you can implement, and which can then be reused as tags within Info*Engine tasks. Refer to the com.infoengine.task.tagext package within the Javadoc released with your installation (`/codebase/infoengine/docs/apidocs`) to understand the classes and interfaces.

In this case, tags are simple event-driven Java classes, that you can group together into libraries of tags. You can do this using a simple XML document referred to as a Tag Library Descriptor (TLD) that defines the library and what tags it supports, what attributes each tag supports, and information about the tags and their attributes. A TLD consists of a root `taglib` node, an optional `description` node, and an arbitrary number of child `tag` nodes. Each `tag` node defines a tag within your tag library and associates that tag with its implementation class. The `tag` node must contain the `name` and `tagclass` child nodes (specifying the tag name and tag implementation class, respectively), and can optionally contain a `description` node and several `attribute` nodes.

If added, any `attribute` nodes specify the attributes a tag supports, whether or not an attribute is required, and if its values can be an expression. For these purposes an `attribute` node must have `name`, `required`, and `rtexprvalue` child nodes to supply this information. A fragment of a TLD file might look like the following:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib>
 <description>My taglibrary.</description>
 <tag>
   <name>myTag</name>
   <description>what this taglibrary does</description>
   <tagclass>org.myorg.tags.MyTag</tagclass>
   <attribute>
     <name>myAttribute<name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
 </tag>
...
</taglib>
```

In this case, the tag library contains a tag named “myTag,” which is implemented by the “org.myorg.tags.MyTag” class. The tag requires a single attribute, named “myAttribute,” which can be an expression. To include your tags within an Info*Engine task requires that you specify an `@taglib` directive in your task that defines a prefix for use with your tags within the task, and also specifies the location of your TLD file (located in the Info*Engine task root). For example, this tag could be used within an Info*Engine task as follows:

```
<%@taglib uri=”/org/myorg/myTags.tld” uri=”pfx”%>
...
<pfx:myTag myAttribute=”${@FORM[]myAttributeValue[]}” />
```

Tag attributes are supplied to your tag using standard Java bean-like set accessor methods. When loading a tag library, Info*Engine verifies all of these implementation details. An error is produced if your tag cannot be properly loaded because of an implementation detail, such as a missing setter for a supported attribute. In the example above, the tag might have a setter method as follows:

```
public void setMyAttribute ( String value )
```

```
{
  // perhaps set an instance variable or other work
  // related to the attribute value specified in the value
  // parameter.
}
```

Tag attributes can have arbitrary data types, and at runtime values are validated or coerced, if necessary. The supported data types of parameter values are:

String

int

boolean

float

com.infoengine.*object.factory.Group*

Object

Coercion is only supported to the primitive types **String**, **int**, **boolean**, and **float**. Because **Object** is included, you can use attributes of any possible type, however in this case your setter method must supply any required type validation or coercion of the attribute value.

Individual tags (implementations of “com.infoengine.task.tagext.Tag”) can be simple, or they can contain child elements (implementations of “com.infoengine. task.tagext.BodyTag”). Instances of **BodyTag** can also support iteration. A **Tag** or **BodyTag** also have the option to implement **TryCatchFinally** as a means to handle exceptions.

Typically a tag extends either the **TagSupport** or **BodyTagSupport** base classes and completes the implementation to add attribute setters and functionality in the necessary **do*Tag** methods.

In addition, tags can be designed to implicitly set variables within tasks for reuse from scriplet code or from expression language. A tag can support this by implementing the following method defined on “com.infoengine.task.tagext.Tag”:

```
Tag.VariableInfo[] getVariableInfo ( Map<String,String>atts );
```

This method is only called at task compilation time to decide whether or not implicit variables need to be defined in the compiled task source. See the Javadoc for a complete description of the **getVariableInfo** method and how to use this functionality.

At runtime, tags are pooled to avoid performance issues related to constructing large numbers of short-lived objects, which may require excess garbage collection. Therefore, you must be careful that your tag uses optional attributes and stores attribute values in instance variables. You should then implement the **doEndTag** method and reinitialize instance variables within a `finally` block. If this is not done, subsequent usages of your tag may not perform as predicted if it carries instance variable values from previous uses.

Expression Language Support

Expression language (EL) is a mechanism for dereferencing dynamic variable content from several possible sources using a simple string syntax. EL can only be used within custom tags, and cannot be used within core library tags, such as **webject**, **param**, and **task**. Furthermore, the EL syntax discussed here cannot be used with JSP pages. However, it is similar to the EL supported by JSTL 1.0/JSP 2.0, which can be used from within JSP pages.

Note

For performance reasons, custom tag attributes that do not require expression language should avoid supporting it.

Using custom tag attributes that support EL, task authors can embed expressions that are evaluated at runtime against the following:

- The VDB
- The context collection, such as @FORM and @SERVER
- The tasklet context
- The **IeContext** object for the current request (which holds request-scoped context variables)

These locations provide the sources of the information fetched by expressions, allowing the desired information to be further refined within an expression.

An expression can also support accessors, which allow you to use brackets (`[ ]`) to index information into arrays, lists, and maps as well as use the `.` character to dereference Java bean properties. For example, an expression indexing into a list might be written as `${listName[1]}`, which fetches the second element in the list named “listName.” Indexing into a map might be written as `${mapName['key']}`. Fetching a property from a Java bean might be written as `${employee.name}`, which returns the result of invoking the **getName()** method on the bean referenced by “employee.”

Expressions can also be arbitrarily complex, for example the following:

```
${employees['123'].name}
```

fetches a bean from a map using the key “123,” and then fetches the “name” property from that bean by invoking the **getName()** method on the bean. More examples are provided throughout this section.

Since it is common to use the `.` character, it is special-cased within Info*Engine data structure attribute values, specifically when the next part of the expression being evaluated is an Info*Engine element object. In this case, the next text in the expression being evaluated specifies an attribute name. For example, in the following expression:

```
${document[0]thePersistInfo.createStamp[0]}
```

if `document[0]` resolves to the first element in a group named “document,” then the `.` character found within “thePersistInfo.createStamp” would be special-cased to avoid breaking the intent of the expression. In other cases, however, the `.` character specifically dereferences a Java bean property, breaking the above expression. Other characters, such as the single quote (‘) and the brace characters (`{ }`), also have special meaning and may need to be escaped. Preceding those characters with a back slash (\) escapes them.

Aside from fetching data from the VDB and context groups, expression evaluation resolves variable values the tasklet context, which refers to a pair of maps containing data. Each task has a unique local map to store variable data, and that map is backed by the **IeContext** object of the overarching request. Each Info*Engine-based request has its own instance of **IeContext** (`com.infoengine.util.IeContext`), which are able to hold contextual information specific to a given request. For example, this can be used for information that needs to be shared by disjointed pieces of source code.

A tasklet context operates similar to the **IeContext**, but is specific to a single task and has a lifespan that carries only through the execution of that task. However, if you want a task to call a sub-task and to share data, then you can place a request-scoped variable in the **IeContext**, which is visible to all other code running within the same request. The **set** tag supplied as part of the `/com/infoengine/tlds/iejstl.tld` tag library has an argument named **requestScope**, which you can use to differentiate between locally-scoped variables and request-scoped variables. During expression evaluation, the local tasklet context is located, and if the variable is not found the **IeContext** is consulted. This means that local variable data overrides request-scoped variable data. For more information, see set Tag on page 293.

Note

Since you cannot differentiate EL from standard Info*Engine substitution syntax, EL is not supported within core Info*Engine tags (such as the **ie: param** data attribute). However, you can programmatically include expressions in attributes using standard JSP-like expressions such as `<%=evaluateExpression("${x}")%>`.

For example:

```
<c:set var="a" value="${array[0]}" />
<c:set var="a" value="${map['a']}" />
```

and

```
<ie:param name="COUNT" data="<%=evaluateExpression("${group.elementCount}")%>" />
```

An expression can have varying degrees of complexity, as long as its individual parts correspond upon evaluation of the expression. For example:

```
<!-- fetch the first element within the variable 'list' which
     is a list of Map objects and then fetch the value of 'a'
     within that Map. -->
<c:set var="a" value="${list[0]['a']}" />
<!-- I have a group with elements containing an attribute named
     'mapAtt' which is a map that I can index into -->
<c:set var="a" value="${myGroup[0]mapAtt[0]['a']}" />
```

Expressions can invoke no-arg getters on variable objects, which are accessed like Java bean properties. For example:

```
<!-- populate the eCount variable with the number of elements
     in "myGroup" -->
<c:set var="eCount" value="${myGroup.elementCount}">
```

A string can have multiple arbitrary expressions embedded within it. For example:

```
<c:if test="${myGroup==null || myGroup.elementCount!=10}">
  <c:choose>
    <c:when test="${myGroup==null}">
      <c:set var="err" value="Group myGroup is null" />
    </c:when>
    <c:otherwise>
      <c:set var="err"
             value="Group myGroup must have 10 elements but it
```

```
 has ${myGroup.elementCount}" />
    </c:otherwise>
  </c:choose>
  <ie:webject name="Throw-Exception" type="MGT">
    <ie:param name="MESSAGE" data="<%=getContextValue("err")%>" />
  </ie:webject>
</c:if>
```

Unlike JSP expressions, Info*Engine EL is integrated with Info*Engine runtime concepts. During the expression evaluation, the task context collections and VDB are consulted before the tasklet and request context. For example:

```
<c:set var="a" value="${@FORM[]a[]}" />
<c:forEach var="elm" list="${myGroup}">
  <!-- executes once for each element in myGroup -->
</c:forEach>
<c:forEach var="a" list="${myGroup[0]a[*]}">
  <!-- executes once for each value of the a attribute
       within the first element of myGroup -->
</c:forEach>
```

Unlike JSP, Info*Engine EL does not support the use of operators within expressions. The core custom tag library extends EL support somewhat specifically with the **c:set**, **c:if/c:when** tags. These tags internally use the `tasklet.evaluateExpression(String)` method (discussed briefly below) to extend EL to achieve their goals. For example:

```
<!-- the test attribute is broken into separate expressions
     that are evaluated independently in order based on the
     conditional expression to be evaluated.
     in this case ${@FORM[]doSomething[]} ${myGroup} and
     ${myGroup.elementCount}. -->
<c:if test="${@FORM[]doSomething[]==true &amp;&amp;  myGroup!=null &amp;&amp;
     myGroup.elementCount>0}">
  <c:set var="idx" value="0" />
  <c:forEach var="elm" list="${myGroup}">
    <c:set var="mod" value="${idx%2}">
    <c:choose>
      <c:when test="${mod==0}"><!-- even -->/c:when>
      <c:otherwise>!<-- odd -->/c:when>
    </c:choose>
    <!-- similar to if/when set may evaluate expressions independently
         as necessary, in this case ${idx} to get what to add 1 to. -->
    <c:set var="idx" value="${idx+1}" />
  <c:forEach>
</c:if>
```

To support combining EL with Java scriplets, `com.infoengine.SAK.Tasklet` has methods defined on it to support interaction with context variables and to allow for programmatic evaluation of expressions (for reuse from within custom tag libraries).

For example, the following sets a variable within the tasklet context with task scope:

```
public final void setContextValue ( String name, Object value );
```

The following sets a variable within the tasklet context with either task or request scope. If `request` is set to `true`, then the variable is available to other tasks within the same request:

```
public final void setContextValue ( String name, Object value, boolean request );
```

The following fetches the names of all variables defined within the tasklet and request context:

```
public final Set<String> getContextKeys();
```

The following fetches a variable by name from the context:

```
public final Object getContextValue ( String value );
```

The following evaluates an expression and returns the resulting object or null. Exceptions related to syntax problems within the expression or runtime evaluation of the expression are thrown:

```
public Object evaluateExpression ( final String expression ) throws IEException;
```

For example:

```
<%
// programmatically sets a context variable that is an array of strings
setContextValue ( “myArray”, new String [] { “a”, “b”, “c” } );
// fetches a group named myGroup from the VDB
// ONLY for illustration
Group g = (Group)evaluateExpression ( “${myGroup}” );
// wouldn’t do this but fetches the above programmatically populated
// myArray variable
String [] myArray = (String [])getContextValue ( “myArray” );
%>
<c:forEach var=”one” list=”myArray”>
  <!-- will run three times, a, b and c -->
  <c:set var=”message” value=”one is ${one}” />
</c:forEach>
```

8 Custom Webjects

The following topics describe custom webjects, which allow an application developer to extend the functionality of the Info*Engine server by writing custom Java code. Custom webjects are simply user-defined webjects that get dynamically loaded by the Info*Engine server. These webjects are then available to any task or template, and have access to all internal classes of the Info*Engine server.

There are two kinds of custom webjects:

- Custom display webjects are used to format group information. A custom display webject has all the capabilities of standard display webjects.
- Custom group webjects are used to manipulate and transform group data into other groups. A custom group webject has all the capabilities of the standard Info*Engine group webjects.

Required Attributes for Custom Webjects

The following attributes and attribute values must be included when you call custom display and group webjects:

Attribute	Description
`TYPE=EXT`	Indicates that a user-defined class is to be used.
`USE=ClassName`	Identifies the user-defined class (including the package) in which the method implementing the webject resides.

Creating an External Custom Webject

You can create Info*Engine external custom webjects by coding a public class within a unique Java package. The class name must be unique and should follow the standard Java naming rules.

The Info*Engine webject name is derived from the class name by separating each word in the name with a hyphen. For example, if the class name is "averageColumn" then the webject name is "Average-Column."

Webject names are not case sensitive.

You must code the class so the custom webjects in the class:

- Are implemented as static methods.
- Take a `com.infoengine.object.factory.Task` object as their only argument.
- Return a `com.infoengine.object.factory.Task` object.
- Throw `IEException` on error.

For example, the signature for the method that implements the Average-Column custom webject is the following:

```
import com.infoengine.object.factory.Task;
public static Task averageColumn ( Task task ) throws IEException {
    :
    :
    Task response = new Task();
            response.addVdb ( group_out );
            return response;
    :
    :
}
```

You can put the code for multiple custom webjects in the same class. For example, if you have private methods that are used by multiple custom webject methods, the private methods and the custom webject methods can be in one class in a package.

When compiling the `NumericColumnWebjects.java` file, you must add the `ieWeb.jar` and `servlet.jar` files to the classpath. To make your compiled class files automatically available to Info*Engine through the JSP engine, you can put the files in the `WEB-INF/classes` directory where Info*Engine is installed. If this directory does not exist you can create it.

External Custom Webject Examples

The following sections include:

- Source code for the **NumericColumnWebjects** class.
- An example JSP page that uses the external custom webjects.

You can find the source code and JSP page in the `prog_examples/customwebjects` directory where Info*Engine is installed.

NumericColumnWebjects Class

The following example package contains the **NumericColumnWebjects** class. This class contains the following methods:

- **getColumnElements** — A private method used by the public methods to get the elements in a column.
- **averageColumn** — Averages the elements in a numeric column of data.
- **totalColumn** — Computes a total using the elements in a numeric column of data.

```
package examples.customwebjects;

import com.infoengine.util.IEException;
import com.infoengine.exception.fatal.IEInternalServiceException;
import com.infoengine.procunit.webject.GroupWebjectException;
import com.infoengine.object.factory.*;

import java.util.Enumeration;
import java.util.Vector;

/**
 * NumericColumnWebjects
 *
 * supplies implementation for two simple webjects:<br>
 * <li><b>totalColumn</b> - return the numeric total of a column
 * <li><b>averageColumn</b> - return the numeric average of a column
 * The implementation is simple minded using a 'double' to calculate
 * totals and averages.  No formating of the results is performed.
 * <br>
 * Example:<br>
 * <ie:webject name="Query-Objects" type="OBJ">
 *   <ie:param name="INSTANCE" data="jdbcAdapter"/>
 *   <ie:param name="CLASS" data="EMP"/>
 *   <ie:param name="WHERE" data="()"/>
 *   <ie:param name="GROUP_OUT" data="employees"/>
 * </ie:webject>
 *
 * <ie:webject name="Total-Column" type="EXT" use="examples.custom
webjects.NumericColumnWebjects">
 *   <ie:param name="COLUMN" data="SAL"/>
 *   <ie:param name="GROUP_IN" data="employees"/>
 *   <ie:param name="GROUP_OUT" data="total_salary"/>
 * </ie:webject>
 *
 * <ie:webject name="Average-Column" type="EXT" use="examples.custom
webjects.NumericColumnWebjects">
 *   <ie:param name="COLUMN" data="SAL"/>
 *   <ie:param name="GROUP_IN" data="employees"/>
 *   <ie:param name="GROUP_OUT" data="average_salary"/>
 * </ie:webject>
 **/

public class NumericColumnWebjects
{

/**
     * return a list of column elements
     *
     * @return Vector
     * @exception IEException if required values aren't found within
the webject
     **/

    private static Vector getColumnElements ( Task task ) throws IE
Exception {
        Webject w = task.getWebject();
        String grp_out = null;
        String column = null;
        String grp_in = null;
        Group group_in = null;

        // get the column name to retrieve
        Param param = w.getParam ( "COLUMN" );
        if ( param == null )
           throw new GroupWebjectException ( "NumericColumnWebjects:
no COLUMN" );
```

```
        column = param.getData();

        // if GROUP_IN not supplied take the default group off the VDB
        param = w.getParam ( "GROUP_IN" );
        if ( param == null )
            group_in = task.getGroupIn();
        else
            group_in = task.getVdb ( param.getData() );
        if ( group_in == null )
            throw new GroupWebjectException
                      ( "NumericColumnWebjects: no GROUP_IN" );

        // build the Vector of column contents

        int element_count = group_in.getElementCount();
        Vector elements = new Vector ();
        for ( int i = 0; i < element_count; i++ ) {
            Element e = group_in.getElementAt ( i );

               Vector ev = e.getValues ( column, true );

               if ( ev == null ) continue;

               for ( int j = 0; j < ev.size (); ++j )
                   elements.addElement ( ev.elementAt ( j ) );
        }
        if ( !(elements.size() > 0) )
            throw new GroupWebjectException
                      ( "NumericColumnWebjects: no COLUMN \"" + column
\"" );

        return elements;
    }
 /**
     * given a input group, column name, and group out name
     * generate the numeric average of the column's contents
     * return the value in a group.
     *
     * @return Task - response
     * @exception IEException - if required parameters are
missing or column
     *                          is not numeric.
     **/

    public static Task averageColumn ( Task task ) throws IE
Exception {
```

```
try {
    String grp_out = null;
    String column = null;
    Webject w = task.getWebject();
    // get the name of the group to create
    Param param = w.getParam ( "GROUP_OUT" );
    if ( param == null )
        throw new GroupWebjectException
                 ( "NumericColumnWebjects: no GROUP_OUT" );
    grp_out = param.getData();
    // get the name of the column to average
    param = w.getParam ( "COLUMN" );
    if ( param == null )
        throw new GroupWebjectException
                  ( "NumericColumnWebjects: no COLUMN" );

    column = param.getData();

    Vector elements = getColumnElements ( task );

    // total the values
    Object val;
    Double dval;
    double average = 0;
    for ( Enumeration en = elements.elements();
        en.hasMoreElements (); ) {
        val = en.nextElement();
        dval = new Double ( val.toString() );
        average += dval.doubleValue();
    }
    // calculate the average
    average = average / elements.size();

    Element elem = new Element ();
    Att att = new Att ( column + " average" );
    att.addValue ( "" + average );
    elem.addAtt ( att );
    Group group_out = new Group ( grp_out );
    group_out.setElement ( elem );

    Task response = new Task();
    response.addVdb ( group_out );

    return response;
} catch ( Exception exc ) {
    exc.printStackTrace ( System.err );
```

```
            if ( exc instanceof IEException )
                throw (IEException)exc;
            throw new IEInternalServiceException ( exc );
        }
    }
 /**
     * given a input group, column name, and group out name
     * generate the numeric total of the column's contents
     * return the value in a group.
     *
     * @return Task - response
     * @exception IEException - if required parameters are
missing or column
     *                          is not numeric.
     **/

    public static Task totalColumn ( Task task ) throws IE
Exception {
        try {
            String grp_out = null;
            String column = null;

            Webject w = task.getWebject();

            // get the name of the group to create
            Param param = w.getParam ( "GROUP_OUT" );
            if ( param == null )
                throw new GroupWebjectException
                          ( "NumericColumnWebjects: no GROUP_OUT" );

            grp_out = param.getData();

            // get the name of the column to total
            param = w.getParam ( "COLUMN" );
            if ( param == null )
                throw new GroupWebjectException
                          ( "NumericColumnWebjects: no COLUMN" );

            column = param.getData();

            Vector elements = getColumnElements ( task );

            // calculate the total
            Object val;
            Double dval;
            double total = 0;
            for ( Enumeration en = elements.elements();
                en.hasMoreElements (); ) {
                val = en.nextElement();
                dval = new Double ( val.toString() );
                total += dval.doubleValue();
            }

            // build GROUP_OUT and response
            Element elem = new Element ();
            Att att = new Att ( column + " total" );
            att.addValue ( "" + total );
            elem.addAtt ( att );
            Group group_out = new Group ( grp_out );
            group_out.setElement ( elem );

            Task response = new Task();
            response.addVdb ( group_out );

            return response;
        } catch ( Exception exc ) {
            exc.printStackTrace ( System.err );
            if ( exc instanceof IEException )
                throw (IEException)exc;
            throw new IEInternalServiceException ( exc );
        }
    }
}
```

You can execute the **averageColumn** and **totalColumn** methods using the Average-Column and Total-Column external webjects.

Example Average-Column and Total-Column External Webjects

The following `NumericColumnWebjects.jsp` executes a Query-Objects webject that creates the “employees” group, which is then used by the Average-Column and Total-Column external webjects:

```
<%@page language="java" session="false" errorPage="IEError.jsp"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>

<!-- Creates the employees group to be used by Average-Column and
Total-Column extended webjects -->

<ie:webject name="Query-Objects" type="obj">
  <ie:param name="INSTANCE"   data="jdbcAdapter"/>
  <ie:param name="CLASS"   data="EMP"/>
  <ie:param name="WHERE"   data="()"/>
  <ie:param name="GROUP_OUT"   data="employees"/>
</ie:webject>

<!-- This custom webject computes the total of the SAL column -->

<ie:webject name="Total-Column" type="EXT"
                               use="example.customwebjects.NumericColumnWebjects"/>
  <ie:param name="COLUMN"   data="SAL"/>
  <ie:param name="GROUP_IN"   data="employees"/>
  <ie:param name="GROUP_OUT"   data="total_salary"/>
</ie:webject>

<!-- This custom webject computes the average of the SAL column -->

<ie:webject name="Average-Column" type="EXT" use="examples.custom
webjects.NumericColumnWebjects">
  <ie:param name="COLUMN"   data="SAL"/>
  <ie:param name="GROUP_IN"   data="employees"/>
  <ie:param name="GROUP_OUT"   data="average_salary"/>
</ie:webject>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"   data="total_salary"/>
</ie:webject>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="GROUP_IN"   data="average_salary"/>
</ie:webject>
```

9 Web Services Framework

The following topics describe the web services tools available with Info*Engine, and provide examples of how to write and deploy a web service for use with Windchill.

About Web Service Implementation

Windchill offers a toolset that facilitates deployment of the same Info*Engine-based web service that leverages JAX-WS (Java API for XML Web Services) in order to handle the SOAP/XML and security-related web service details. This toolset is available in addition to the Info*Engine-based web services that are supplied by the SimpleTaskDispatcher servlet. The web service implementation, which is based on Info*Engine tasks and task delegation, remains mostly consistent. However, the XML representation of the corresponding SOAP interactions differ based upon the manner in which JAX-WS uses JAXB (Java Architecture for XML Binding) to bind Java objects and Info*Engine data structures to XML. For backward compatibility, access to the same services is available using the SimpleTaskDispatcher.

Aside from leveraging JAX-WS for the SOAP-related details, a major advantage to using this framework is the addition of configurable and efficient security. Legacy Info*Engine web services rely solely upon web server authentication for security. This means that SOAP clients must have complete access to user credentials, and must supply those credentials to the web server per its authentication requirements (typically HTTP Basic Authentication). However, this presents a fairly significant roadblock to developing applications that implement SSO (Single Sign On). This is when a separate application or application server has independently authenticated a user, and a trust relationship exists between the client and Windchill. Relying solely upon web server authentication means that an SSO application or application server must have access to the user's password to reauthenticate that user within an Info*Engine web service.

A downside to leveraging more advanced security policies is the complexity of the mechanisms involved. The web service client must be capable of complying with the configured security policy, which often involves encryption and a complex XML representation of the security information embedded within SOAP requests. If the client in question is Java, then the tooling supplied with Info*Engine hides these complexities so that you can write your client without complex coding for security purposes. If you use a separate third-party SOAP framework to connect to Windchill web services, then it is that client's responsibility to comply with the security policy chosen for a given web service.

With web service security support in Windchill, you can establish a trust relationship between an application (such as a portal server) and an Info*Engine web service by exchanging public key information, requiring that SOAP message exchanges be properly signed and encrypted, and requiring that they contain a valid username token that the web service should trust. As long as the incoming SOAP message meets the security requirements and can be decrypted using a public key found in the server's truststore, the web service then executes a request that has been authenticated with the username. Similarly, depending on the requirements of the security policy in use, the client may be required to have access to the server's public key to decrypt responses.

Each web service endpoint is represented by an instance of a Java servlet. As a result, a JAX-WS-based Info*Engine web service can be deployed multiple times if each time the web service is secured by a different security policy and uses a unique servlet name for each deployment.

Optionally, given additional configuration, it is possible to place a JAX-WS-based Info*Engine web service behind web server authentication and forgo the cryptography-related requirements of selecting another security policy.

Note

This is the default security policy for JAX-WS-based web services released with Windchill.

Using Info*Engine Web Services Tooling

This is a basic overview of how the Info*Engine tooling for web services is organized. After reviewing this section, you should be able to locate tools and understand their purpose. The actual use of the tools is discussed in greater depth later.

The supplied tooling is enabled using Apache Ant, located in:

`<Windchill>/ant`

(where *<Windchill>* is the Info*Engine installation directory).

The tooling is intended to be run from a windchill shell, in which case the `ant` command is already in your path.

The directory that contains the Apache Ant build tools for deploying a web service is located in `<Windchill>/bin/adminTools/WebServices` and contains the following:

- `build.xml`

  The primary Apache Ant build script used to deploy web services and associate those services with security policies.
- `security.properties`

  A properties file containing the security-related configuration information used when deploying a web service. This file contains the security policy to use, the server-side truststore and keystore configurations, and any additional policy-specific configuration (if necessary) that is to be used at deployment time.
- `new-project.xml`

An Apache Ant build script used to generate new Apache Ant projects, which are then used to build or deploy new web services or web service clients.

- `xslt/`

  A subdirectory containing XSL documents used during the web service deployment by `build.xml`.

- `client/`

  A subdirectory containing client-side security configuration properties.

- `client/security.properties`

  The client-side security configuration (truststore and keystore configuration information).

  Note

  The paths specified to the truststore and keystore must be specific to the corresponding location where those files reside on the client machine. These paths are included in security deployment descriptors and are not configurable.

- `common/`

  A subdirectory containing the common Apache Ant framework for web service projects.

Of these build tools, you should edit the `security.properties` files to specify your security configuration for client and server. If desired, you can make copies of these files for each client/server pair you decide to write.

To identify the changes needed, you can use the examples provided in the `<Windchill>/prog_examples/jws` directory, the root of the web service example projects. Each project is generated and developed using the `new-project.xml` script. This directory also contains the `jws-stores.xml` Apache Ant build script that can be used to generate client/server keystore and truststore pairs. The `jws-stores.xml` script uses OpenSSO to generate self-signed certificates for the client and server, and is intended as an example of how you can build your own truststores and keystores.

Note

You should use your own certificates rather than the generated self-signed certificates.

Prerequisites

Windchill releases Java libraries that are duplicated by Java 1.6, but are newer than those packaged with Java. This presents no runtime issues, but before you can use the web service tooling to generate and deploy new services or clients, you must update your Java installation to include the necessary updated classes.

To do this, copy the following JAR file:

```
<Windchill>/srclib/webservices-api.jar
```

to

```
${JAVA_HOME}/jre/lib/endorsed
```

Where *${JAVA_HOME}* is the location of the Java used by Windchill. If your installation of Java does not already have the `endorsed` directory, then you must create it.

Note

In some cases, it may be necessary to apply the same change to the client's Java installation as well.

Understanding the Security Requirements

This section explains the security policies available to you, and how to configure the security properties, truststores, and keystores.

Security Policies

Prior to deploying a web service, you must decide how you want the service to be secured. The following are all supported security policies:

SAML Sender Vouchers with Certificates

This mechanism protects messages with mutual certificates for integrity and confidentiality, as well as with a Sender Vouchers SAML token for authorization.

When using this security policy, the SOAP client provides an SAML Subject (within an SAML Assertion) containing the name identifier, which signifies the username the server should run the request as. In this case the message payload is signed and encrypted, because the client and server have established a trust relationship using the contents of their respective keystores and truststores.

In this situation, the server only verifies that the supplied username is valid prior to handling the request.

This policy requires that the client have both a keystore and truststore corresponding to the server's. In addition, Java clients require a callback handler that provides the SAML Assertion containing the username that the server should trust.

Note

It is extremely important to understand that when using this mechanism the client is "vouching" for the username associated with the SOAP request. The SOAP client is, in essence, impersonating a given user or users without being required to supply corresponding passwords. This mechanism is really only appropriate for an SSO (Single Sign On) type scenario, in which you are certain that the client application has appropriately authenticated the users, and you are willing to extend trust to that client using certificate exchange in the keystore and truststore configuration.

Username Authentication with Symmetric Keys

The username authentication with symmetric keys mechanism protects SOAP interactions for both integrity and confidentiality. Symmetric key cryptography relies on a single, shared secret key that is used to both sign and encrypt a message. Symmetric keys are usually faster than public key cryptography.

For this mechanism the client does not possess any certificate or key of its own, but instead sends its username and password for authentication. The client and the server share a secret key. This shared, symmetric key is generated at runtime and encrypted using the service's certificate. The client must specify the alias in the truststore by identifying the server's certificate alias.

In this situation, the server authenticates the given username using the corresponding supplied password.

This policy requires that the client have a truststore corresponding to the server's configuration. In addition, Java clients require callback handlers to provide the username and password to send along with the request for the server to validate. For an example of this mechanism, see Username Authentication with Symmetric Keys Example on page 187.

X.509 Certificate Authentication Over SSL

This security policy uses SSL (`https`) for confidentiality and an X.509 certificate to authenticate the user. The user's certificate must be trusted by the server, and the username that is validated with Windchill is extracted from the certificate. By default, the username is extracted from the subject of the certificate (stripping any preceding `CN=` from the participant). Since the basic mapping of the certificate to a Windchill username may be insufficient, the logic used to extract the username is configurable. You can implement the `com.ptc.jws.security.SubjectToUidHandler` interface and configure your servlet implementation using the following:

```
wt.property com.ptc.jws.security.x509SSL.<ServletName>.SubjectToUidHandlerClass
```

which specifies your implementation.

`<ServletName>` within the property name refers to the servlet for which the `SubjectToUidHandler` implementation should be used.

Note

When writing a Java client to a service protected by SSL, you must supply the `https` URL of the WSDL for your service to the `javax.xml.ws.Service` constructor argument as an instance of `java.net.URL`. Your HTTP server's SSL certificate must either be signed by a certificate authority or manually imported into the client's `<JAVA_HOME>/jre/lib/security/jssecacerts` certificate store.

The security.properties File

The Apache Ant build tooling requires a security configuration for the server that specifies the security policy to use, as well as the truststore and keystore configurations. You must edit the `security.properties` file to specify this information before deploying a web service.

The server `security.properties` file is found at:

```
<Windchill>/bin/adminTools/WebServices/security.properties
```

It contains comments describing its contents.

For writing or testing Java clients, there is a parallel client properties file found at:

```
<Windchill>/bin/adminTools/WebServices/client/security.properties
```

It contains similar comments describing its contents.

Note

Both files, as released, contain a configuration that points to corresponding client and server truststore and keystore pairs, which can be generated for use with the web service examples discussed later in "Truststores and Keystores.". While it is possible to use these generated truststores and keystores to secure web services, it is recommended that you acquire your own certificates and use them to populate truststores and keystores for your applications.

Truststores and Keystores

Unless you are manually configuring your web service behind web server authentication, you are required to secure your web service using X509 v3 certificates. Note that the security mechanisms used require the `SubjectKeyIdentifier` extension. The keytool utility released with Java does not generate this extension.

The Apache Ant build script is located at:

```
<Windchill>/prog_examples/jws/jws-stores.xml
```

This build script can generate client or server keystore and truststore pairs for use by Info*Engine web service examples. You can also use it for testing and development purposes. The Apache Ant build script uses OpenSSL to generate client and server certificates, and then uses the Java keytool utility to import these certificates into client or server keystore and truststore files.

You can use the `jws-stores.xml` build script as an illustration on how to use your own certificate to create truststores and keystores for your web services and clients. This script only generates a single server and client certificate, and then imports those certificates into their corresponding keystores before generating the truststores for the client and server.

To generate these files, run the following script from a windchill shell:

```
% cd <Windchill>/prog_examples/jws
% ant -f jws-stores.xml
```

While the script is running, you are prompted several times for user input. You can choose to either accept the defaults (presented surrounded by brackets like `[ws-server]`) by simply pressing **Enter** when prompted, or supply your own input. If you choose the default input, then the released configured `security.properties` files should contain the proper configuration. If you decide to supply other input when running the script, then you need to update the corresponding `security.properties` configuration accordingly. When the script has finished running, the *<Windchill>*`/prog_examples/jws/stores` directory is created. This directory contains the following files:

- `server-keystore.jks`
- `server-truststore.jks`
- `client-keystore.jks`
- `client-truststore.jks`
- `server.p12`
- `client.p12`
- `server.cer`
- `client.cer`

Note

The `.p12` (PKCS #12 Personal Information Exchange) files can be used to populate a Windows certificate store for .NET client access requiring certificates. The `.cer` files are the certificates used to populate the truststores and keystores.

Writing an Info*Engine-based Web Service

The methods for writing an Info*Engine-based web service have not changed since web services were first supported with Info*Engine release 7.0. The only area in which JAX-WS-based web services are now different is the support for SOAP with attachments. For more information, see Using SOAP Attachments on page 184.

In addition to writing an Info*Engine-based web service, you can write a Java-based web service. This is illustrated with released examples found in *<Windchill>*`/prog_examples/jws`, which is discussed in Truststores and Keystores on page 164.

An Info*Engine web service consists of a set of Info*Engine tasks, each representing a web service method that is associated with a type identifier. In the case of a web service, the type identifier is used simply as a way to logically group a set of Info*Engine tasks together to form a web service, and can be thought of as analogous to a Java class that exposes only static methods (each method being the Info*Engine task supporting the corresponding web service operation).

At runtime, Info*Engine requires that the type identifier and corresponding Info*Engine tasks be registered with the Info*Engine configuration in the LDAP. You can do this manually, but since the process can be involved and is potentially error prone, Info*Engine is released with Apache Ant tooling that packages and installs the Info*Engine tasks backing your web service. These utilities are integrated with the Apache Ant framework.

This section walks you through writing a simple Info*Engine task-based web service and client using the following steps:

1. Create a project (this is a simple directory structure) to hold your web service tasks and client source.
2. Write a few very basic tasks.
3. Secure and deploy your JAX-WS web service, which consists of deploying your Info*Engine tasks and corresponding servlet.

Before You Begin

Before you can complete the steps in this section, you must meet the following prerequisites:

- All commands are assumed to be run from a windchill shell.
- It is assumed that you have run the Apache Ant script documented earlier in Truststores and Keystores on page 164.
- The example commands assume a Unix environment. If you are in a Windows environment, then you must alter the operating system-specific commands to suit your environment (for example, using \ rather than / in paths).

Create a Project

Released with the web service framework are tools to simplify authoring new web services and clients.

This project example creates a simple web service that performs basic math operations.

Note

This service is for illustration purposes only and is not the type of thing you would typically use Info*Engine to accomplish.

Create your project using the `<Windchill>/bin/adminTools/WebServices/new-project.xml` Apache Ant build script. This script can be used to create a client project, a service project, or both:

```
% cd <Windchill>
% mkdir prog_examples/jws/MyProject
% ant -Dproject.dir=<Windchill>/prog_examples/jws/MyProject  -Dservlet.name=MathService
-Dsecurity.policy=userNameAuthSymmetricKeys -Dservice.type.id=org.myorg.MathService
-Dmain.class=org.myorg.MathClient
-f bin/adminTools/WebServices/new-project.xml create
```

Note

The `project.dir` property is an absolute path so that the project is created in the appropriate location. To see what input options are supported by the `new-project.xml` build script, you should run it with no arguments. For example:

```
ant -f bin/adminTools/WebServices/new-project.xml
```

This creates the following structure within the `prog_examples/jws/MyProject` directory:

- `src/`

  The base directory for your web service (server-side).
- `src/build.xml`

  The build script used to build and deploy your web service.
- `src/tasks/org/myorg/MathService`

  A subdirectory where you can create your Info*Engine tasks. This is only a suggestion; you can create another directory hierarchy that suits your needs if desired. The tools package your service using the type identifier you specified on the `service.type.id` property when creating the project.
- `src_client/`

  The base directory for your web service client.
- `src_client/build.xml`

  The build script used to build your web service client.
- `src_client/org/myorg/MathClient.java`

  The beginnings of your web service client.

Review the contents of the two generated `build.xml` scripts to gain some understanding on how they work.

To further familiarize yourself with the framework from within the `src` and `src_client` directories, run the following `ant` commands:

```
% ant -p
% ant usage
```

The first command displays the public targets for the associated build script with their descriptions. The second command provides a detailed description of how you can tailor your build script if you find it necessary to do so. For most basic projects this should not be necessary.

Write the Info*Engine Tasks for Your Web Service

This step creates four simple tasks (`Add.xml`, `Divide.xml`, `Multiply.xml`, `Subtract.xml`) in the following locations with the given source. Note that these tasks are very simple and primarily contain documentation.

Add.xml

Source: `<`*`Windchill`*`>/prog_examples/jws/MyProject/src/tasks/org/myorg/MathService/Add.xml`

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%@taglib uri="/com/infoengine/tlds/iejstl.tld" prefix="c"%>

<!--com.infoengine.soap.rpc.def
Generates the sum of two integers.
<p>Supply two integers, a and b. You can subtract
integers using the {@link Subtract} method.

@param int a The first integer.
@param int b The second integer.
@return int ${output[0]result[0]} The sum of a and b.
@see Multiply
@see Divide
@see Subtract
-->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
<!-- just so non-web service clients could call this task
     coerce a to an int, in case it isn't already -->
<c:set var="A" value="${0+@FORM[0]a[0]}" />
<c:set var="${output[0]result}" value="${A+@FORM[0]b[0]}" />
```

Divide.xml

Source: `<`*`Windchill`*`>/prog_examples/jws/MyProject/src/tasks/org/myorg/MathService/Divide.xml`

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%@taglib uri="/com/infoengine/tlds/iejstl.tld" prefix="c"%>
```

```
<!--com.infoengine.soap.rpc.def
Generates the result of dividing two integers.
<p>Supply two integers, a and b. You can multiply
integers using the {@link Multiply} method.

@param int a The first integer.
@param int b The second integer.
@return float ${output[0]result[0]} The sum of a and b.
@see Multiply
@see Divide
@see Subtract
-->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
<!-- just so non-web service clients could call this task
     coerce a to an float, in case it isn’t already -->
<c:set var="A" value="${0.0+@FORM[0]a[0]}" />
<c:set var="${output[0]result}" value="${A/@FORM[0]b[0]}" />
```

Multiply.xml

Source: *<Windchill>*`/prog_examples/jws/MyProject/src /tasks/org/myorg/MathService/Multiply.xml`

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%@taglib uri="/com/infoengine/tlds/iejstl.tld" prefix="c"%>

<!--com.infoengine.soap.rpc.def
Generates the product of two integers.
<p>Supply two integers, a and b. You can divide
integers using the {@link Divide} method.

@param int a The first integer.
@param int b The second integer.
@return int ${output[0]result[0]} The sum of a and b.
@see Multiply
@see Divide
@see Subtract
-->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
<!-- just so non-web service clients could call this task
     coerce a to an int, in case it isn't already -->
```

```
<c:set var="A" value="${0+@FORM[0]a[0]}" />
<c:set var="${output[0]result}" value="${A*@FORM[0]b[0]}" />
```

Subtract.xml

Source: *<Windchill>*`/prog_examples/jws/MyProject/src/tasks/org/myorg/MathService/Subtract.xml`

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%@taglib uri="/com/infoengine/tlds/iejstl.tld" prefix="c"%>

<!--com.infoengine.soap.rpc.def
Generates the difference of two integers.
<p>Supply two integers, a and b. You can add
integers using the {@link Add} method.

@param int a The first integer.
@param int b The second integer.
@return int ${output[0]result[0]} The sum of a and b.
@see Multiply
@see Divide
@see Subtract
-->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="GROUP_OUT" data="outputd"/>
</ie:webject>
<!-- just so non-web service clients could call this task
     coerce a to an int, in case it isn't already -->
<c:set var="A" value="${0+@FORM[0]a[0]}" />
<c:set var="${output[0]result}" value="${A-@FORM[0]b[0]}" />
```

Deploy Your Web Service

When this project was created, a parameter named `security.policy` was supplied, for which the value of `userNameAuthSymmetricKeys` was explicitly passed. Therefore, for this example you must provide both a username and a password to your web service client.

To generate and deploy your web service, run the following commands:

```
% cd <Windchill>/prog_examples/MyProject/src
% ant
```

Note

Windchill does not need to be active to run the Apache Ant script.

At this point start (or restart) Windchill to finish deployment.

When run with no arguments, the Apache Ant script packages, installs, and deploys your web service:

- The service is packaged, compiled, and installed in Windchill. In the case of an Info*Engine task-based service, your tasks are packaged into a PTCTAR file and installed. In this example, the resulting PTCTAR file is:

  ```
  <Windchill>/prog_examples/jws/MyProject/dist_server/mathservice.ptctar
  ```

  If your web service requires Java objects, then you can simply create the package hierarchy and source for those objects within the `src` directory. The script automatically compiles and package those classes for you. Classes associated with your web service are packaged in `<Windchill>/codebase/WEB-INF/lib` within a JAR file that reflects the servlet name of your service.
- The `<Windchill>/bin/adminTools/WebServices/build.xml` script is then run to generate and compile the server-side artifacts, as well as secure and deploy the web service. This build script can also be run on its own, depending on your needs. For more information, see Using the Web Service Deployment Build Script on page 183.

  Note

  If you use the Apache Ant framework you should not need to run this build script manually. The Apache Ant framework can also be used to deploy an existing legacy web service that is already installed on your system. In this case the `src` directory contains only the `build.xml` script and no task source.

Now that your service is deployed and Windchill is running, you can review your web service WSDL by visiting a URL similar to the following:

```
http://<host>/Windchill/servlet/MathService?wsdl
```

If you are writing a Java client, as described in Writing a Java-based Web Service Client on page 173, then you do not need to know how to access the WSDL. However, you must access it if you plan to integrate your service with another web service client.

Undeploy Your Web Service

If you want to undeploy your web service from your installation, simply run the `undeploy` target for the `src/build.xml` script of your project. For example:

```
% cd <Windchill>/prog_examples/MyProject/src
% ant undeploy
```

The `undeploy` target completes the following actions:

- Removes the servlet configuration from `web.xml`
- Removes the web service configuration from `sun-jaxws.xml`
- Removes the security policy configuration file
- Removes the JAR file supporting your web service
- Uninstalls the Info*Engine tasks (if your service is based on Info*Engine tasks)

Info*Engine Web Service Parameters and Task Documentation

You should familiarize yourself with the process of properly documenting Info*Engine tasks. Visit the documentation for your new web service online at the following URL:

```
http://<host>/Windchill/infoengine/jsp/tools/doc/
index.jsp
```

Note

This utility provides documentation for several installed tasks registered under type identifiers. Most of these tasks are not written to be accessed as web services, and should not be used as such.

From the **Repository Types** pane in your web browser, select **com.ptc.windchill**. From the **Classes** pane, select your **org.myorg.MathService** service type identifier. The documentation that appears is based on the contents of the tasks you deployed. For more information on writing task documentation, see:

```
http://<host>/Windchill/infoengine/jsp/tools/doc/
howto.html
```

For more information on writing SOAP tasks, see SOAP Services on page 193.

Writing a Java-based Web Service Client

The project directory (`src_client`) and the example Java source file `src_client/org/myorg/MathClient.java` were created in the step Create a Project on page 166.

The `MathClient.java` source file is dependent upon source artifacts, which are generated from your deployed web service. Before actually being able to complete your client source, you must generate and compile these artifacts. As generated, `MathClient.java` compiles, but does nothing until you use it to invoke a web service method.

In order to get these artifacts, you must compile your empty client using the following commands:

```
% cd <Windchill>/prog_examples/jws/MyProject/src_client
% ant
```

The Apache Ant script also generates an executable JAR file of your application:

```
<Windchill>/prog_examples/jws/MyProject/dist_client/MathServiceClient.jar
```

However, since your main class does nothing, then executing the JAR also does nothing. Running the previous `ant` command processes the WSDL for your web service and generates Java source artifacts to the following:

`<Windchill>/prog_examples/jws/MyProject/gensrc_client`

This creates Java source code within the com.ptc.jws.service.org.myorg.mathservice package. To avoid Java code being generated into a package with a name derived from the service name space, you can use the `jaxws.package` property within the `build.xml` file to explicitly define which package you want the source code generated into.

Edit the client source code to complete your client:

1. In a text editor, open the following file:

   ```
   <Windchill>/prog_examples/jws/MyProject/src_client/org/myorg/MathClient.java
   ```

   The source looks something like:

   ```
   package org.myorg;

   // import the classes generated when compiling your client
   //import com.ptc.jws.service.?.*;
   import com.ptc.jws.client.handler.*;

   public class MathClient
   {
       public static void main ( String [] args ) throws java.lang.Exception
       {
   ```

```
// depending on your security requirements you may need to specify credentials up
// front, or on the command-line where most policies will prompt for them
//Credentials.setUsername ( "demo" );
//Credentials.setPassword ( "demo" );
// TODO implement your client         /*
        MathServiceImplService service = new MathServiceImplService();
        MathServiceImpl port = service.getMathServiceImplPort ();
        //invoke an action
        //port.methodName ( <arguments> );
        */
    }
}
```

2. Update the source to import the server-side web service artifacts, and invoke a method as follows:

```
package org.myorg;

// import the classes generated when compiling your client
import com.ptc.jws.service.org.myorg.mathservice.*;

public class MathClient
{
    public static void main ( String [] args ) throws java.lang.Exception
    {
        int a = args.length > 0 ? Integer.parseInt ( args[0] ) : 0;
        int b = args.length > 1 ? Integer.parseInt ( args[1] ) : 0;

        MathServiceImplService service = new MathServiceImplService();
        MathServiceImpl port = service.getMathServiceImplPort ();
        System.out.printf ( "a+b=%d\n", port.add ( a, b ) );
        System.out.printf ( "a-b=%d\n", port.subtract ( a, b ) );
        System.out.printf ( "a*b=%d\n", port.multiply ( a, b ) );
        System.out.printf ( "a/b=%f\n", port.divide ( a, b ) );
    }
}
```

3. Recompile your client and update your executable JAR to include the updated classes:

```
% ant compile
% ant dist
```

4. Run your new web service client from the command line:

```
% java -jar ../dist_client/MathServiceClient.jar 10 20
Username:<password>
Password:<username>
a+b=30
```

```
a-b=-10
a*b=200
a/b=0.500000
```

Where *<username>* is the username and *<password>* is the password you have supplied. When running your client, you are prompted on the command line for credentials.

Client Callback Handlers

When creating the web service project, the `-Dsecurity.policy=usernameAuthSymmetricKeys` property was supplied. This instructed the framework to explicitly secure the web service with the security policy, which is described in Username Authentication with Symmetric Keys on page 161. If this property had not been explicitly provided, then the security policy would have defaulted to the value of the `security.policy` property as configured in `<Windchill>/bin/adminTools/WebServices/security.properties`.

When generating the client portion of the project, this policy causes a default client-side callback handler configuration to be used. This is specified in the `handler.config` property of the `src_client/build.xml` build script. To view the format of the `handler.config` property, run the `ant usage` command from within the `src_client` directory.

The com.ptc.jws.client.handler package also contains some basic callback handlers for your web service client to use for the supported security policies (these handler classes are automatically packaged for you when you build your client). You can choose to use these classes to provide security information for your web service clients, or you can choose to author your own by providing a class that implements `javax.security.auth.callback.CallbackHandler` and then providing that class as part of the `handler.config` property.

The default callback handler supplied for `usernameAuthSymmetricKeys` prompts for credentials on the command line. It can be configured to prompt the user for credentials using swing dialogs or to fetch the credentials from system properties. It only prompts for credentials once per thread, and reuses the credentials until you programatically clear them.

If you want to programmagically provide credentials, you should do so using the `com.ptc.jws.client.handler.Credentials` class using the **setUsername** and **setPassword** methods.

The web services framework makes use of two separate callback mechanisms:

- The WSIT (Web Services Interoperability Technologies) mechanism uses the `javax.security.auth.callback.CallbackHandler` implementations.

These handlers are used for security policies other than `webServerAuthenticated` to gather the credentials or security tokens that are passed to the web service. If these implementations do not meet the needs of your application, you can implement your own and supply them using the `handler.config` property in the `build.xml` file.

- The JAX-WS handler chain is essentially a list of Java classes typically implementing the following:

  ```
  javax.xml.ws.handler.soap.SOAPHandler<javax.xml.ws.handler.soap.SOAPMessageContext>
  ```

  Instead of implementing the client code, these handler classes can provide generic functionality and conditioning for SOAP interactions. The web services framework provides some basic handlers to handle details, such as supplying the URL to your web service and adding credentials to requests when web server authentication is used.

  These handlers can also be written to perform additional functions, such as add headers to SOAP requests or handle faults in a generic fashion. You can tailor the handler chain of you client by changing the values of the `handler.chain` properties in the `build.xml` file. These properties are described in greater detail within the `build.xml` file.

Portable Web Service Clients

There are certain issues that must be addressed to allow a web service client to run on a client machine properly or to interact with multiple identical services hosts on separate Windchill instances. Security policy configuration is essentially hard-coded at compile time, requiring a new client to be reconfigured for each client host. Similarly, the address of the web service is fixed at compile time, and therefore writing a client that interacts with separate service instances requires the service endpoint to be provided programatically. To improve the portability of web service clients, the Web Services Framework uses the JAX-WS handler chain, and allows client JAR files to be reconfigured as needed.

Using Security Policies

When securing a web service with one of the supported security policies (other than web server authentication), a Java client needs access to its own keystore and truststore. Paths to these files (keystore and truststore) are configured for use by the build framework in the following:

```
<Windchill>/bin/adminTools/WebServices/client/security.properties
```

The paths to the keystore and truststore are absolute, and must be compiled into deployment descriptors packaged with your client JAR file in its META-INF directory.

If you compile a web service client from the same host where Windchill is running with the standard configuration, then these files are available to your client, but only on that host. Before redistributing your web service client to another host, you must do one of the following:

- Update the `client/security.properties` file (or make a copy of it in your `src_client` directory and then update it) to contain paths to the location where these files exist on the new client host, and then rebuild your client.
- Reconfigure the client JAR file for every host from which it is run. For more information, see Reconfiguring the Client JAR File on page 177.
- Compile a copy of the client for every host on which it is intended to run, with each containing the appropriate paths for the specific host. For more information, see Compiling a Copy of Your Client for Every Host on page 179.

Reconfiguring the Client JAR File

After compiling and packaging your client, the `dist_client` directory of your project includes two JAR files: one containing your compiled client, and the `webservices-support.jar` file. This JAR file is executable and can be used to reconfigure your client as follows:

```
% java -jar webservices-support.jar
-clientJar <jar file> [-targetJar <jar file>] [-securityProperties
<properties file>] [-wsdl <service wsdl>]
```

If `-targetJar` is not specified, then `-clientJar` is overwritten. You must also specify `-wsdl` or `-securityProperties`, or both.

To reconfigure the paths to the truststore and keystore locations on your client, copy the `security.properties` file to the client host and run the utility with the `-securityProperties` argument. This can also be used to reconfigure the default URL to your web service. Alternatively, you can supply the URL to your web service using a system property, or write your own JAX-WS handler to provide the URL.

Note

The file paths to the truststore and keystore locations within the property file must be absolute and cannot contain other property references.

When reconfiguring a client JAR in this way, you also need to supply additional information about what callback handlers your client should to use. If you do not supply this information, then your reconfigured client cannot function unless it is programmatically supplying the information normally supplied by callback handlers. Specifically, this includes the classes required to gather user credentials or perform other general processing through the JAX-WS handler chain.

Additional properties you might want to set are:

`handler.config`

`handler.chain.authentication`

`handler.chain.extras`

`handler.chain.url`

`handler.chain.authentication`

Note

For descriptions of these properties, run the `ant usage` command from within your `src_client` project directory.

For an existing client JAR, the configuration is supplied in the `security.properties` file. At build time this configuration is typically found within your `src_client/build.xml` Ant script.

If you are reconfiguring only the paths to truststores and keystores, then you should check the value of the `handler.config` property in your `src_client/build.xml` script. Supply this property in the `security.properties` file when reconfiguring your client JAR.

If you are reconfiguring your client JAR because your web service security policy has changed, then you might need to change the handler configuration of your client. For example:

```
#web service security.policy=userNameAuthSymmetricKeys
handler.config=usernameHandler:com.ptc.jws.client.handler.
UsernamePasswordCallbackHandler,passwordHandler:com.ptc.jws.client.handler.
              UsernamePasswordCallbackHandler
#web service security.policy=samlsv
handler.config=samlHandler:com.ptc.jws.client.handler.SamlCallbackHandler
```

Note

These properties would be in addition to the com.ptc.jws.client. handler package.

If you are reconfiguring a client to a web service that is secured by web server authentication (`security.policy=webServerAuthenticated`) then you must supply the system property `requires.authentication=true` when running the command. A `security.properties` file is only necessary if you want to specify your own JAX-WS handler chain configuration. For example:

```
% java -D requires.authentication=true -clientJar <client jar> -wsdl <service wsdl>
```

In this case the default value for `handler.chain.authentication` is automatically used.

Compiling a Copy of Your Client for Every Host

Another option is to compile a copy of your client for every host it is intended to run on, with each containing the appropriate paths for the specific host.

For example:

```
% cd <Windchill>/prog_examples/jws/MathService/src_client
% cp <Windchill>/bin/adminTools/WebServices/client/security.properties .
```

Update the local copy of `security.properties` as necessary for the individual client. This ensures that these security properties take precedence over the common properties found in `<Windchill>/bin/adminTools/WebServices/ client/security.properties`.

Dynamically Specifying a Web Service URL

When creating a Java-based web service client, the HTTP connection information is determined from your Windchill configuration. When building the JAR for your web service client, the framework stores a local copy of the Web Service Definition Language (WSDL) to your web service and references that WSDL from a catalog file. This allows your client to avoid having to access the WSDL over the network at runtime.

By default, the web services framework configures your JAX-WS handler chain to include a class that imposes the URL to your server based on the Windchill configuration. The default implementation is `com.ptc.jws.client.handler.WebServiceURLHandler`, which can be overridden in `build.xml` using the `handler.chain.url` property. You should therefore not need to specify this information to run your client.

Your client can use the `wt.webservice.url` system property to override the web server URL, or you can supply the URL programmatically using the `setWebServiceURL(java.net.URL url)` method on the `com.ptc.jws.client.handler.WebServiceURLHandler` class.

Alternatively, you can override the `handler.chain.url` property with your own implementation to supply this URL. Doing this requires that you set the `javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY` within the instance of `javax.xml.ws.handler.soap.SOAPMessageContext` given to your handler implementation.

Web Server Authenticated Services and Clients

Using the standard security policies (such as SAML Sender Vouches or Username Authentication with Symmetric Keys) means that your web service servlet is deployed in a manner that is unprotected by web server authentication. In these cases it is the responsibility of the web service, not the web server, to enforce authentication.

You can also deploy your servlet behind web server authentication. In order to do this, use the **webServerAuthenticated** security policy when deploying, as documented in:

```
<Windchill>/bin/adminTools/WebServices/
security.properties
```

The tooling automatically updates the protected resource configuration within `<Windchill>/apacheConf` (provided you are using a project to build and deploy the service). If running, these changes are propagated to your local Apache configuration, which should simply be restarted. If you are using a web server other than Apache, or your Apache is not located on the same host as Windchill, you must propagate these changes manually to your web server configuration.

By default, the JAX-WS handler chain is configured to contain a handler implementation to impose user credentials in outgoing requests for your web service client. These credentials can be supplied programmatically using the `setUsername(String user)` and `setPassword(String password)` methods on the `com.ptc.jws.client.handler.Credentials` class. Alternatively, credentials can be supplied on the `wt.webservice.user` and `wt.webservice.password` system properties.

You can also override the handler implementation using the `handler.chain.authentication` property within your `build.xml` file. If you implement your own handler, you must set the `javax.xml.ws.BindingProvider.USERNAME_PROPERTY and` `javax.xml.ws.BindingProvider.PASSWORD_PROPERTY` properties within the `javax.xml.ws.handler.soap.SOAPMessageContext` object supplied to your handler implementation.

Note

If you undeploy a web service that is protected by web server authentication, the tooling does not automatically update your web server configuration to remove the previously protected resource. You must reconfigure your web server manually.

Writing a Java-based Web Service

You also have the option of manually implementing your service directly in Java. In this case, you can generate a web service project using the following Apache Ant script:

`<Windchill>/bin/adminTools/WebServices/new-project.xml`

with a `service.type` of `java`. Then, in place of specifying the `service.type.id` property, instead specify the class to implement Java using the `service.class` property.

For example:

```
% cd <Windchill>
% mkdir prog_examples/jws/MyJavaProject
% ant -Dproject.dir=<Windchill>/prog_examples/jws/MyJavaProject
-Dservlet.name=MyJavaService
-Dsecurity.policy=userNameAuthSymmetricKeys -Dmain.class=org.myorg.MyJavaServiceClient
-Dservice.type=java -Dservice.class=org.myorg.MyJavaService
 -f bin/adminTools/WebServices/new-project.xml create
```

Creating the new project generates a base class for your new web service, which can be found at:

`<Windchill>/prog_examples/jws/MyJavaProject/src/org/myorg/MyJavaService.java`

This contains source code that looks something like:

```
package org.myorg;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import com.ptc.jws.servlet.JaxWsWebService;

@WebService()
```

```
public class MyJavaService extends JaxWsWebService
{
    @WebMethod(operationName="add")
    public int add ( int a, int b )
    {
      return a+b;
    }
}
```

You can also simply replace the example **add** method with your own web service method. You can also add other web service methods and any required supporting classes within the `src` directory of your project. Then compile, package, and deploy your new service as follows:

```
% cd <Windchill>/prog_examples/jws/MyJavaService/src/
% ant
```

This compiles your web service and packages it in a JAR file based on your servlet name within `<Windchill>/codebase/WEB-INF/lib`. This also deploys the service, which is secured using the security policy you specified when creating your project. Then you can implement your web service client in the same manner as described in Writing a Java-based Web Service Client on page 173.

If you choose to generate the client portion of your web service project, the client source code example might not compile properly if uncommented (it is commented out in the main method of your web service client). The client-generated source code assumes the web service in question is Info*Engine-based, and therefore generates source code based upon that assumption. You must then compile your client, examine the client side source code artifacts in the `gensrc_client` directory, and then adjust your client source accordingly. Finally, you must then rerun the `ant compile` and `ant dist` commands to recompile and package your client.

Consult the standard Java `javadoc` for the javax.jws, javax.xml.ws, and related packages to learn more about writing a Java web service. You can download these at http://java.sun.com.

Note that in the generated source code example your new web service class extends `com.ptc.jws.servlet.JaxWsService`. This is not an absolute requirement, but it is strongly suggested and provides you with some basic conveniences, such as:

- Access to an implicitly instantiated log4j logger in the **$log** instance variable, the logger name of which is based on your web service class name.

  You can log messages by simply invoking methods on the logger such as:

  ```
  $log.debug (“my message”);
  ```

Consult the log4j documentation for more information on logging with log4j, found at http://logging.apache.org/log4j/.

- Access to the `javax.xml.ws.WebServiceContext` associated with a web service request in the **$wsc** instance variable.
- Access to the `javax.xml.ws.handler.MessageContext` associated with a web service request using the **getMessageContext()** method.
- Access to the associated `javax.servlet.http.HttpServletRequest` object using the **getServletRequest()** method. This is important because this object is the only **HttpServletRequest** object that has been properly configured with the authenticated username used to access your service.

Note

Standard Java web services provide access to the **HttpServletRequest** object through the **MessageContext** object stored under the `MessageContext.SERVLET_REQUEST` key. You should NOT use this servlet request object, but rather use the one returned using the **getServletRequest()** method of your web service. Using the **HttpServletRequest** object would create a conflict, as the security layer that preprocesses web service requests extracts and validates the security credentials from the incoming SOAP requests based on your security policy, and then imposes those credentials using a `javax.servlet.http.HttpServletRequestWrapper` object. The **MessageContext** key SERVLET_REQUEST is an immutable property, and so the security layer must store its version of the **HttpServletRequest** object under another key within the **MessageContext**. This object has the validated username imposed in the **getRemoteUser()** and **getUserPrincipal()** methods of this **HttpServletRequest** object.

Using the Web Service Deployment Build Script

Released with Windchill is an Apache Ant built script that can be used to both generate and deploy a web service. If you are manually writing your web service and are making use of the project support provided, then this script is automatically called for you. This means that you do not need to understand its functionality directly (aside from the `security.properties` configuration). If, however, you have a legacy web service already installed that you do not want to repackage and deploy (for example the com.ptc.windchill.ws Generic Web Services released with Windchill), then you can choose to run this script manually (the same can also be achieved using the supplied Apache Ant framework). Running this script without any input properties only displays its requirements, and at a minimum you must supply the `servlet.name` and `type.id` or `webservice.class` properties as input. On the command line you can also choose to explicitly override properties included from `security.properties` by adding `-D<propertyName>= <propertyValue>` arguments. An example is specifying a `security.policy` property value that differs from the default values you have configured.

You can also choose to redeploy the com.ptc.windchill.ws web service after it has been secured using SAML Sender Vouches. To redeploy it under a servlet named "GenericWebService" run the following commands:

```
% cd <Windchill>/bin/adminTools/WebServices
% ant -Dservlet.name=GenericWebService -Dtype.id=com.ptc.windchill.ws
-Dsecurity.policy=samlsv generate
```

For example, you could choose to explicitly specify the `security.policy` property on the command line. If you simply want to use the security policy configuration in the `security.properties` file, then this property is not necessary.

Running the above command line performs the following actions:

1. Generates a JAX-WS based web service that exposes the com.ptc.windchill.ws Info*Engine-based web service.
2. Uses the wsgen utility to generate the required Java source artifacts in support of the web service deployment.
3. Compiles the previously generated source.
4. Based on the configured security policy, it applies the appropriate security policy information for use by the service. For most policies this results in a file in `<Windchill>/codebase/WEB-INF` named `<wsit-class>.xml`, which contains the security policy instructions for the web service layer.
5. JAR the compiled web service classes for use by Windchill.
6. Adds servlet and servlet-mapping deployment information for your web service servlet to:

   `<Windchill>/codebase/WEB-INF/web.xml`
7. Creates or updates `<Windchill>/codebase/WEB-INF/sun-jaxws.xml` to deploy the web service, and associates the server-side callback handler responsible for extracting security information from the SOAP requests corresponding to the configured security policy.

It is not necessary for Windchill to be running to deploy a web service; however, you should restart Windchill after running the script to deploy or redeploy a web service.

Using SOAP Attachments

JAX-WS web services support the ability to include binary data in a scalable fashion in your SOAP requests.

Just as with legacy web services, to upload binary data (available on the input stream of your Info*Engine task), simply add a parameter to your task with a type of `javax.activation.DataSource`.

Unlike legacy Info*Engine web services, which used standard MIME SOAP attachments, JAX-WS web services use attachment references to make binary data appear as inline web service parameters. However, this approach no longer implicitly allows a SOAP client to include associated information like the filename or content type with the attachment using the `Content-Disposition` MIME header. As a result, Info*Engine web services now allow the filename to be specified as a separate parameter associated with the attachment. If an additional parameter is not used to specify an attachment filename, then the attachment has a default filename of "unknown."

You can also explicitly specify the content type of the attachment. This, however, only alters the content type that is exposed to Info*Engine components for the incoming data. If not specified, then the default value for the attachment content type is `application/octet-stream`.

For example:

```
@param string fileName The filename
@param javax.activation.DataSource file The file to stage {fileName:fileName}
 {contentType:image/jpeg}
```

In the previous example, your task receives a BLOB (attachment) and the filename of the attachment is specified by the accompanying `fileName` parameter value. The **Content-Type** of that attachment is exposed to the Info*Engine task as `image/jpeg`. It is important to note that this does nothing more than provide a value for the **Content-Type** of the incoming attachment. It does not force a web service client to comply with that **Content-Type** when supplying the data.

As with legacy web services, return an attachment from a task by simply specifying its return type as `java.io.InputStream`. In both the upload and download scenario, a Java web client is presented with a parameter or return value with a type of `javax.activation.DataHandler`.

In both the upload and download scenarios, an Info*Engine web service tunnels data between your task and the web service client to efficiently transfer it to and from Windchill. An efficient upload, however, is somewhat dependent upon how the client supplies the data. Small amounts of data can be internalized to memory, but clients should chunk the MIME data when sending large amounts. When using a Java-based client you must explicitly instruct the client to chunk data destined for the server or your client will likely die with an `OutOfMemoryError`. You should add the HTTP_CLIENT_STREAMING_CHUNK_SIZE property in your request context prior to uploading an attachment to Windchill. It is also very important to note that if you do not do this, it is likely that your application may put undue stress on the memory resources of the server on which your web service is running.

For example:

```
import java.io.File;
import javax.activation.FileDataSource;
import javax.activation.DataHandler;
import javax.xml.ws.BindingProvider;
import com.sun.xml.ws.developer.JAXWSProperties;
...
((BindingProvider)port).getRequestContext ().put ( JAXWSProperties.
   HTTP_CLIENT_STREAMING_CHUNK_SIZE, 8192 );
File f = new File ( path );
DataHandler file = new DataHandler ( new FileDataSource ( f ) );
port.upload ( f.getName(), file );
```

In addition, when performing downloads it is important to test the resulting **DataHandler** to see if it is an instance of **StreamingDataHandler**. When downloading large amounts of data, the Sun classes may store that data in a temporary file. If you do not explicitly close an instance of **StreamingDataHandler**, then this temporary file remains on your client machine occupying disk space.

For example:

```
import javax.activation.DataHandler;
import com.sun.xml.ws.developer.StreamingDataHandler;
...
DataHandler file = null;
try
{
  file = port.download ( ... );
  ..
  // transfer the data to another stream/file, etc.
  file.writeTo ( ... );
 }
finally
{
  if ( file instanceof StreamingDataHandler )
    ((StreamingDataHandler)file).close(); // remove any temporary file
}
```

Consuming External Services from Windchill

It is possible to write web service clients that can be called from within Windchill. If done, you must use the `jaxws.package` property of your client's `build.xml` file. To facilitate its interactions, each client must have a unique package name for the generated resources. Each client source should also remain packaged in its own JAR file, because there are several static resources stored within the `META-INF` directory of a client's JAR that overlap with those of other web service clients.

If the client is a third-party service, you can use the `wsdl.url` property to specify the location of the remote service's WSDL. If the service allows for anonymous interactions, then you must set the `anonymous.service` property.

Web Service Examples

There are several web service examples released with Windchill to illustrate the basics on how to write a web service and clients. You can find the examples in `<Windchill>/prog_examples/jws`.

For each, review the service source contained in the `src` directory and the client source contained in the `src_client` directory.

Each example was developed using the `new-project.xml` script, and are therefore organized in a manner consistent with this documentation. To exercise each example, enter the following commands:

```
% cd <Windchill>/prog_examples/jws/<ExampleDir>/src
% ant
```

This compiles, packages, and deploys the web service. At this point you should start Windchill, or restart it if it is already running. Then run the following commands:

```
% cd <Windchill>/prog_examples/jws/<ExampleDir>/src_client
% ant
```

This generates, compiles, and builds the distributable JAR for the client. Some example clients take optional parameters. Read the client source code within the `src_client` directory to see what parameters, if any, they may use.

To run the client from the `src_client` directory, enter the following:

```
% java -jar ../dist_client/<ClientJarFile>
```

Running `ant clobber` from both the `src` and `src_client` directories removes all generated and compiled files, which restores the example directory back to its original state.

After you are done reviewing a particular example web service, you can undeploy the service using the following:

```
% cd <Windchill>/prog_examples/jws/<ExampleDir>/src
% ant undeploy
```

Username Authentication with Symmetric Keys Example

This example illustrates how to author a C# Web Services client that digitally signs its requests and embeds credentials used for authentication.

This example client was authored on a Windows 2008 Server with .NET 3.0 and Visual Studio 2008.

1. Generate the Client/Server and Truststore/Keystore Pairs

In order for a server and a client to communicate using digitally signed SOAP requests, the client and server are required to trust one another by trading certificate information. An example utility is provided with Windchill that can be used to generate certificates, Java keystores, and truststores using OpenSSL. Use a windchill shell to invoke the following:

```
% cd <windchill>/prog_examples/jws
% ant -f jws-stores.xml
```

During invocation of this script you are prompted to enter values for certificate aliases, subjects, and passwords. The default values are typically sufficient. If you alter any of these values while running this script you may need to account for those differences later.

Of particular importance is the keystore alias for the server. The default value presented by the script above is the value of `wt.rmi.server.hostname` from your `wt.properties` file. If this is not the hostname where your web server is running, then you can change this value. From a Java perspective this alias can be any basic string identifier, but .NET prefers that these aliases reflect real host names that can be resolved through DNS. If you decide to use an alias that does not reflect your server's hostname then you must use additional configuration on the client side.

If you do not accept the default values when running this script then you should review the contents of the `security.properties` file to make sure it matches the information provided when running the script. This file is located in:

```
<Windchill>/bin/adminTools/WebServices/
security.properties
```

When this script has completed running, a directory named `<Windchill>/prog_examples/jws/stores` is created, which contains a file named `server.p12` (a personal information exchange file) with the server's public key, which is used later in this process.

The `jws-stores.xml` ant script generates both server and client keypairs. This example, however, only requires the server's public certificate.

2. Deploy the Web Service

Once you have run the `jws-stores.xml` script you can create and deploy your web service. Here the simple Java web service example that comes with Windchill is used, and which performs basic addition. In this case the service must be secured using user authentication with symmetric keys. For more information, see Understanding the Security Requirements on page 161.

Execute the following to compile and deploy the example web service:

```
% cd <Windchill>/prog_examples/jws/JavaService/src
% ant
```

When the script completes, restart Windchill to finish the deployment of the web service. After it has been restarted, the web service's WSDL is accessible at a URL such as:

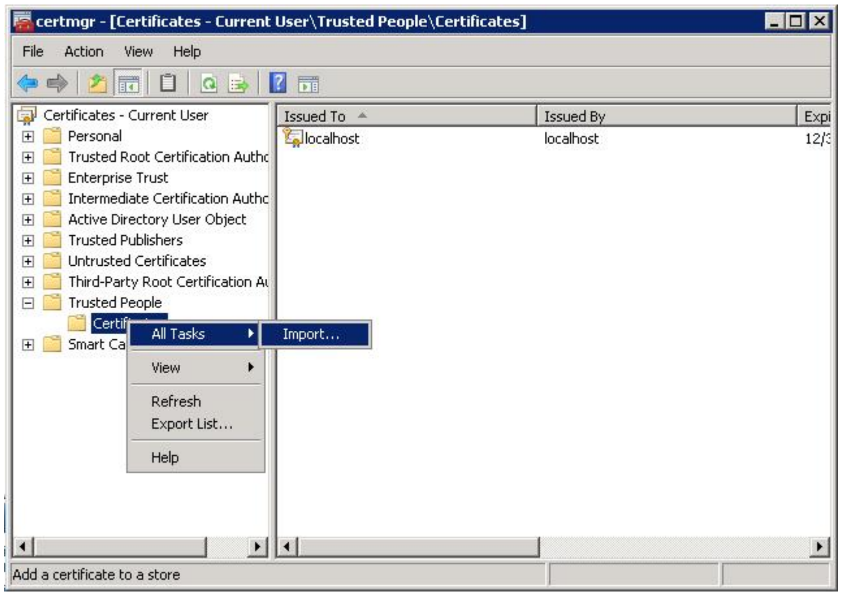
```
http://<host>/Windchill/servlet/
ExampleJavaService?wsdl
```

3. Import the Server's Public Key into the Client's Certificate Store

1. Transfer the `server.p12` file created in the first step to the client's machine.
2. Start the Certificate Manager by navigating to **Start** ▸ **Run** and entering the following command:

   ```
   C:\Windows\System32\certmgr.msc
   ```

3. Next, import `server.p12` into the “Trusted People” store:

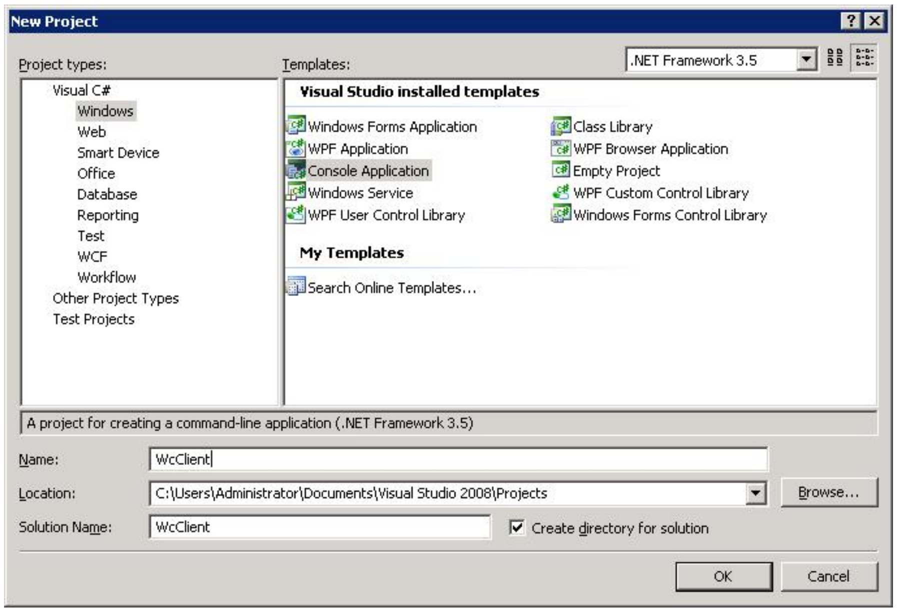

certmgr - [Certificates - Current User\Trusted People\Certificates]
File Action View Help
Certificates - Current User
Personal
Trusted Root Certification Autho
Enterprise Trust
Intermediate Certification Autho
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Au
Trusted People
Certif
Smart Ca
All Tasks
Import...
View
Refresh
Export List...
Help
Issued To
Issued By
Expi
localhost
localhost
12/3
Add a certificate to a store

4. When the Import Wizard runs, browse to the location of the `server.p12` file, select it, and then supply the password you chose when generating the certificate. The default password is “changeit.”

4. Write the Web Services Client

1. Start Visual Studio and select **File** ▸ **New** ▸ **Project** to create a new project.
2. In the **New Project** window, select **Console Application** from the **Templates** pane.

Enter a name for your application (“WcClient” is used in this example):



A project for creating a command-line application (.NET Framework 3.5)

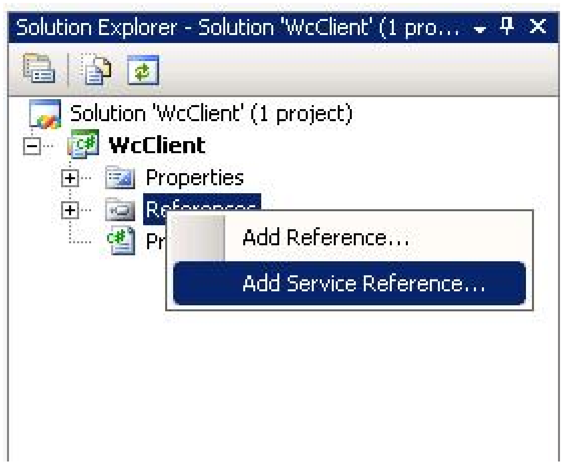
Name: WcClient

Location: C:\Users\Administrator\Documents\Visual Studio 2008\Projects

Solution Name: WcClient

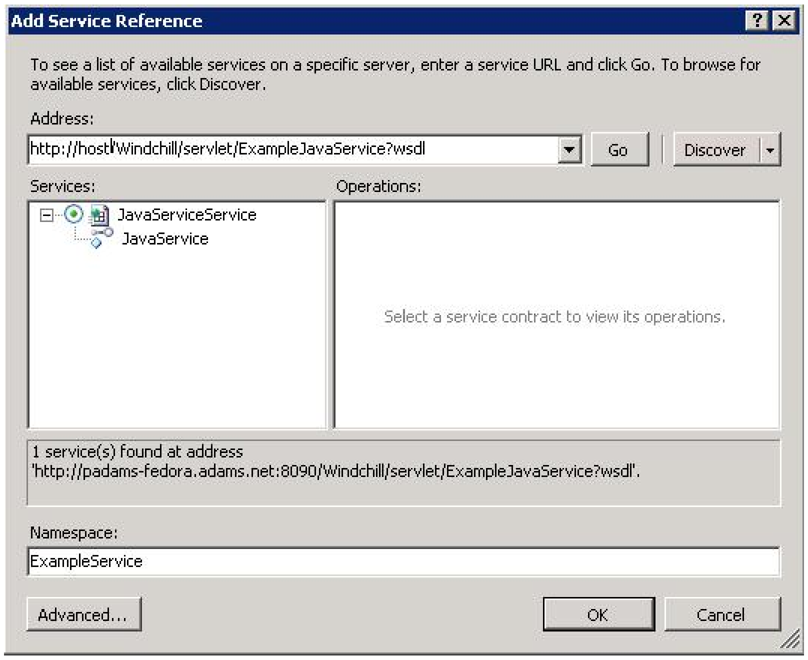
☑ Create directory for solution

3. Once your project is created, the base `.cs` file opens in the main editor window. In the **Solution Explorer** pane, right-click the **References** folder and select **Add Service Reference**:


Solution Explorer - Solution 'WcClient' (1 pro...
Solution 'WcClient' (1 project)
WcClient
Properties
References
Pr
Add Reference...
Add Service Reference...

4. In the **Add Service Reference** window, enter the URL of the service’s WSDL in the **Address** field, and click **Go** to fetch the WSDL document.

5. Enter a **Namespace** (“ExampleService” is used in this example) and click **OK**:



To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address:
http://host/Windchill/servlet/ExampleJavaService?wsdl

Services:
JavaServiceService
JavaService

Operations:
Select a service contract to view its operations.

1 service(s) found at address
'http://padams-fedora.adams.net:8090/Windchill/servlet/ExampleJavaService?wsdl'.

Namespace:
ExampleService

6. In the **Solution Explorer** pane, right-click again on the **References** folder, and this time select **Add Reference**.
7. In the window that opens, scroll down and add the **System.Net** reference. Repeat this process to add the **System.Security** reference.
8. Now you must write the client source code to access the web service. For example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography.X509Certificates;

namespace WcClient
{
    class Program
    {
        static void Main(string[] args)
        {
            ExampleService.JavaServiceClient client = new ExampleService.
 JavaServiceClient();
            client.ClientCredentials.UserName.UserName = "demo";
            client.ClientCredentials.UserName.Password = "demo";
            client.ClientCredentials.ServiceCertificate.SetDefaultCertificate(
                   StoreLocation.CurrentUser,
                   StoreName.TrustedPeople,
                   X509FindType.FindBySubjectName,
                   "host.company.com");
            Console.WriteLine("1+2=" + client.add(1, 2));
        }
    }
}
```

9. Update the subject name when fetching the X509 certificate to match your hostname (in the example above it’s written as “`host.company.com`”).
10. Select **Build ▸ Build Solution** or press F6 to build your client.
11. Test your client by pressing Ctrl+F5 or by selecting **Build ▸ Debug ▸ Start Without Debugging**.

5. Use a Certificate Alias that Does Not Reflect the Hostname of Your Server

As mentioned previously, .NET prefers that the certificate alias match the true hostname where your service resides. If for some reason your alias does not match your server’s hostname and you attempt to run your client, you receive a DNS-related error. You can work around this problem by editing your client’s `app.config` file by adding a DNS alias to your service reference. For example, if your certificate alias is “ws-server” you might update `app.config` to make “ws-server” an alias for your web service endpoint:

```
<endpoint address="http://host.company.com/Windchill/servlet/ExampleJavaService"
                binding="customBinding" bindingConfiguration="JavaServicePortBinding"
                contract="ExampleService.JavaService" name="JavaServicePort">
                <identity>
                    <dns value="ws-server" />
                </identity>
</endpoint>
```

10

SOAP Services

SOAP (Simple Object Access Protocol) is a lightweight, XML-based protocol for exchanging information and making remote procedure calls in a distributed, decentralized environment.

For general information about SOAP, use the following link:

http://www.w3.org/TR/SOAP

The following topics discuss legacy SOAP services, provide information for writing tasks with SOAP, and explain how SOAP requests work within an Info*Engine Java Platform, Enterprise Edition (Java EE) connector. A step-by-step example on how to a write a Java SOAP client is also included.

Legacy Services

When running Windchill, it is recommended that you use the SimpleTaskDispatcher servlet rather than the RPC servlet. The two servlets behave identically externally, but in the case of a Windchill installation, the SimpleTaskDispatcher delegates directly to the SimpleTaskDispatcher service running in your MethodServer. As a result, this servlet performs better than the standalone Info*Engine RPC servlet.

In the following topics, wherever there is a reference to the RPC servlet address, you can simply replace "RPC" with "SimpleTaskDispatcher" in that URL.

For example:

```
http://<host>/Windchill/servlet/RPC
```

can be replaced with

```
http://<host>/Windchill/servlet/SimpleTaskDispatcher
```

Note

If your site is using form-based authentication, programmatic clients attempting to access the RPC or SimpleTaskDispatcher servlet must use the `/protocolAuth` URL prefix. For example:

```
http://<host>/Windchill/protocolAuth/servlet/
SimpleTaskDispatcher
```

or

```
http://<host>/Windchill/protocolAuth/servlet/RPC
```

For more information about form-based authentication, see the PTC Windchill Advanced Deployment Guide.

SOAP RPC Servlet

Info*Engine provides a SOAP Remote Procedure Call (RPC) servlet that receives SOAP requests from a SOAP client and sends responses back to the client. The servlet gathers the information sent from the client (consisting of the methods to execute and other request information) and processes the methods by executing tasks that people at your site have created explicitly for use with the SOAP client. These tasks are used to execute task webjects, and the servlet allows SOAP clients to invoke these specially-written Info*Engine business tasks. For more information about the required SOAP task format, see Info*Engine Task Invocation with SOAP on page 198.

The Info*Engine SOAP RPC servlet can be used to pass data to and from a SOAP client that has been developed using the Info*Engine Java EE connector or some other third-party SOAP client. For more information, see Info*Engine Java EE Connector on page 211.

The SOAP RPC servlet automatically generates Web Services Definition Language (WSDL) for a given SOAP service when it receives an HTTP GET request (for example, when using a standard web browser to access the servlet directly). When issuing a GET request to the RPC servlet, you must supply the CLASS query argument, and can optionally supply the VERSION and STYLE query arguments. The CLASS query argument specifies the CLASS of the given SOAP service for which you would like WSDL generated. The VERSION and STYLE parameters are used to decide what type of WSDL you would like to have generated. For more information about the STYLE and VERSION parameters, see the webject description in Generate-WSDL on page 422.

For example, if you generate the default WSDL for an Info*Engine SOAP service that you authored under the following type identifier:

```
com.myCompany.myService
```

You might request the WSDL as follows:

```
http://<host>/Windchill/servlet/RPC?CLASS=com.myCompany.myService
```

If you are attempting to integrate your SOAP service with a third-party Service Oriented Architecture (SOA) platform for the task of attempting to call your SOAP methods from within a Business Process Execution Language (BPEL) orchestration, you would want Info*Engine to behave in a WS-I compliant manner to ease integration. In such a situation you would likely request your WSDL as follows:

```
http://<host>/Windchill/servlet/RPC?CLASS=com.myCompany.myService
         &VERSION=1.1&STYLE=document
```

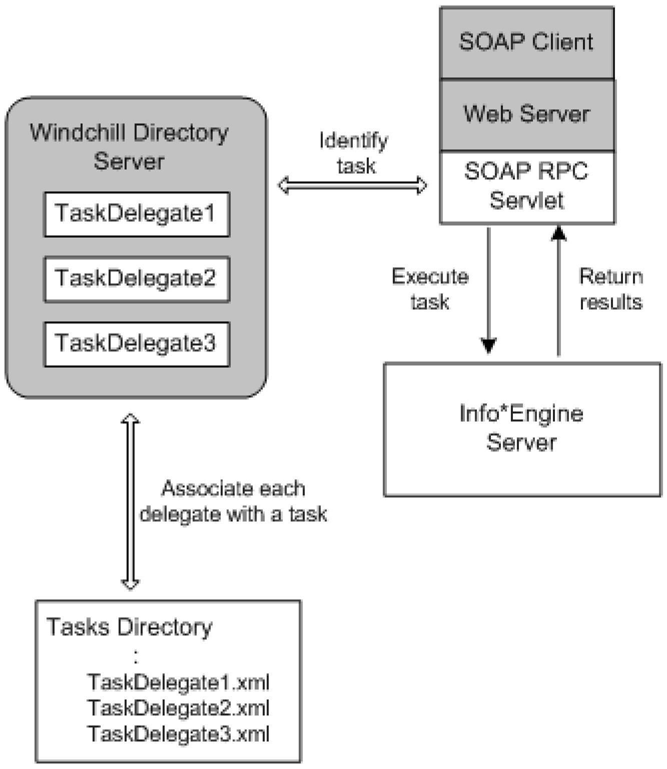
The Info*Engine Java Platform, Enterprise Edition (Java EE) connector uses WSDL to generate client-side source code to simplify interactions with Info*Engine.

The Microsoft SOAP Toolkit uses WSDL to define, validate, and constrain the remote classes and methods that clients can call, the parameters that are supported by each method, and the kinds of results that are returned. For more information, and to download the Microsoft SOAP Toolkit, use the following URL:

http://www.microsoft.com/downloads

Other third-party clients can also use WSDL to generate source code or to drive interactions with Info*Engine.

The following diagram illustrates how a SOAP client interacts with Info*Engine:



When a SOAP request is made, the web server processes the HTTP request and directs it to the SOAP RPC servlet. The SOAP RPC servlet accesses the LDAP repository in your Windchill Directory Server to identify which task should be executed. In the LDAP directory, you have created task delegates that identify tasks that are to be executed for the methods the SOAP client can execute. The SOAP RPC servlet passes the name of the task and the other request information on to an Info*Engine server, which executes the task.

After the Info*Engine server executes the task, an appropriate response is sent to the SOAP client. The response sent to the SOAP client is based on the data returned by the Info*Engine task. A task authored for invocation by a SOAP client can return the following:

- A primitive value, for example **int**, **float**, **date**, and so on.
- A Java bean.
- An array of primitive values.
- An array of Java beans.
- An Info*Engine collection, group, element, attribute, or group XML as a string.

If an Info*Engine group is returned and the SOAP task does not explicitly state its return type, the group is returned to the SOAP client as a single string that contains the Info*Engine XML representation of that group, named using the input `$(@FORM[]group_out[0])`.

If an error is detected, a SOAP fault is returned to the client. This fault can then be processed by the SOAP client using its own implementation-specific mechanisms. For example, Info*Engine Java EE connector clients receive SOAP faults as Java exceptions; Visual Basic clients can make use of instance variables such as **faultcode**, **faultstring**, **faultactor**, and **detail**.

SOAP Requests

SOAP requests are defined by the SOAP client, which sends the request to the web server, which then passes the request on to the SOAP RPC servlet. The SOAP RPC servlet can be found in the following location:

```
http://<host>/Windchill/servlet/RPC
```

where *Windchill* is the web application defined for the SOAP RPC servlet.

When sending a SOAP request, an Info*Engine SOAP client must specify a method to be executed and the class that supports the method. This information can be conveyed to the SOAP service in multiple ways:

- In the `SOAPAction` HTTP header in a URI (Uniform Resource Identifier) of the form:

  ```
  uri:ie-soap-rpc:class!method
  ```

  where *method* is the method to be executed and *class* is the class that supports the method.
- On the CLASS and METHOD query arguments.

If CLASS is specified as a query argument and METHOD is not, then the SOAP service attempts to resolve the method name by looking to the `SOAPAction` HTTP header and then the SOAP message body. If METHOD is specified as a query argument and CLASS is not, then the SOAP service attempts to resolve the class name by looking to the `SOAPAction` HTTP header.

Note

GET requests to the SOAP service must contain a CLASS parameter, and return the WSDL document for the specified class. POST requests to the SOAP service are handled as SOAP requests.

The association between the method name sent by the SOAP client and the Info*Engine task that gets executed is maintained in an LDAP repository as a task delegate entry. The class sent by the SOAP client corresponds to a type identifier entry maintained in an LDAP repository. Type identifiers represent object types (or classes) and contain task delegates (or methods). The Info*Engine server that processes the task invocation is represented by a repository, which declares its repository type.

Repository types represent types of information systems and maintain what object types they can act upon. Repository types contain type identifiers. The repository is stored in the Windchill Directory Server that is used by Info*Engine. For more information on repositories, repository types, type identifiers and task delegates, see the online help available through the Task Delegate Administration utility.

Info*Engine Task Invocation with SOAP

For an Info*Engine task to be invoked through a SOAP request, the following requirements must be met:

- The task should be located in a directory path under the task root directory that corresponds with the logical CLASS argument. This keeps the tasks (or methods) supported by the class logically and hierarchically organized. The task can be located anywhere within the task root, as long as the associated task delegate entry properly references the task. Tasks can be located anywhere within the task hierarchy.
- The task must be registered as a task delegate within a type identifier whose name reflects the CLASS used within the SOAP request. Task delegates can be manually created, either using the Delegate Administration utility, or using supplied tools.
- The task must include special SOAP comments lines within the SOAP markup comment if the task requires input parameters or returns a response other than an Info*Engine group. For more information, see Inserting SOAP Comments on page 200.

Note

Despite the fact that leaving the @return comment out and specifying a comment produce the same results in the client, PTC recommends that you include a comment such as the following:

```
@return INFOENGINE_GROUP $(output)
```

Not specifying an @return comment means the data is serialized as XML and tunneled in the response as a string, requiring the client to deserialize the SOAP and process the internal XML. When returning an Info*Engine Group, specifying the @return line is suggested since doing this produces a more useful interface and performs better.

Supported Data Types

The Info*Engine SOAP RPC servlet supports the following data types, which include primitive types, Java classes, and special Info*Engine classes:

- **boolean**, `java.lang.Boolean`
- **double**, `java.lang.Double`
- **long**, `java.lang.Long`
- **float**, `java.lang.Float`
- **int**, `java.lang.Integer`
- **short**, `java.lang.Short`
- **byte**, `java.lang.Byte`
- **string**, `java.lang.String`
- **dateTime**, `java.util.Date`
- `java.math.BigDecimal`

The following Info*Engine-specific data types are also supported for purposes of transferring Info*Engine data structures as embedded XML within SOAP requests:

- INFOENGINE_COLLECTION
- INFOENGINE_GROUP
- INFOENGINE_ELEMENT
- INFOENGINE_ATTRIBUTE

Parameters of each of these types are provided to the supporting task as the equivalent Java class. Single values of each of these types may be used as input parameters or as an output value.

Note

In order for the SOAP service to supply strongly-typed input parameters, the SOAP client must supply a strongly-typed SOAP request. Some third-party clients, such as Microsoft's SOAP Toolkit, do not do this and as a result all input parameters are supplied to the task as strings.

In addition to basic data types and arrays of basic data types, Info*Engine SOAP also supports complex types in the form of Java beans, both individually and in arrays. Indexed properties and nested beans are also supported. Returning nested beans requires generation of complex Info*Engine groups. These complex groups must be created manually as Info*Engine webjects do not currently generate complex nested groups.

The class of a Java bean must be found in the Info*Engine server and SOAP RPC servlet CLASSPATH so that the bean can be instantiated, examined, and manipulated at runtime. Similarly, if the client side of the SOAP communication uses Info*Engine classes, such as the Info*Engine Java EE connector, the Java bean class must also be available in the client CLASSPATH. For more information on the Info*Engine Java EE connector, see Info*Engine Java EE Connector on page 211.

SOAP Comments

An Info*Engine SOAP task must contain a comment section that describes the input parameters and response type. This SOAP comment section is used by Info*Engine to generate a WSDL that describes the interface each Info*Engine task exposes to external SOAP clients. The generated WSDL can then be used by third-party software to verify input parameters and cast response types, or to generate client-side source code used to interact with the Info*Engine SOAP service. Certain tools supplied with Info*Engine make use of the WSDL to generate client side data access objects (DAOs) that can be used to invoke Info*Engine tasks and abstract all of the details involved in using SOAP as the communication protocol. For more information, see Generating DAOs on page 220.

The SOAP comment section can also be used to generate Javadoc-like documentation for tasks. For instructions on generating this documentation, and a link to the generating tool, see the document available in Windchill at the following URL:

`http://<host>/<servlet>/infoengine/jsp/tools/doc/howto.html`

where *<servlet>* is the application URL defined for the SOAP servlet.

Note

This comment section should be included in all SOAP tasks, even if the task includes no input parameters and returns the default Info*Engine group as a response.

The comment section is as follows:

```
<!--com.infoengine.soap.rpc.def
description
special_comments
-->
```

where `description` is an optional task description, and `special_comments` are the special SOAP comment lines.

Special SOAP comment lines can be included in the comment sections to specify input parameters and output values. These special comment lines are discussed in the next two sections.

@param

The @param comment line is used to specify input parameters. The comment line is specified in the following format:

```
   @param type[] name comment
```

where:

- *`type`* is the data type of the parameter.
- `[]` indicates an array of values. There are no spaces between the parameter type and `[]`. Arrays are available in the @FORM context group as multi-valued attributes. This allows for simple manipulation of multi-valued input with standard Info*Engine substitution syntax.
- *`name`* is the name of the parameter.
- *`comment`* is additional text describing the parameter. This comment is optional.

Note

Some of the Info*Engine-specific data types are processed differently from the other data types in that they do not need scriptlet code to extract the input.

When a collection is given as a parameter, the collection itself is not given as input to the task. Instead, all groups within the input collection are placed in the VDB so that they can be used by the task. The task author is expected to know the names of the groups. For example:

```
@param INFOENGINE_COLLECTION input_collection A collection of Groups.
```

When a group is given as a parameter, that group is placed directly in the task's VDB so that it can be used directly. Additionally, the group is renamed using the parameter name if the group supplied by the caller is named differently. The contents of the group parameter below are available in a task using form substitution such as `$(input_group[]attribute[])`:

```
@param INFOENGINE_GROUP input_group A group.
```

When an element or array of elements is supplied as input, they are placed in a group named using the parameter name and added to the VDB. The input element specified with the parameter below would be available in a task using form substitution such as `$(input_element[0]attribute[])`. In the following example the group named `input_element` would contain a single element:

```
@param INFOENGINE_ELEMENT input_element An element.
```

@return

The @return comment line is used to specify output or response type. If no @return comment line is present, then the Info*Engine SOAP service responds with a serialized Info*Engine XML representation of the output group. The output group must be named using the `$(@FORM[]group_out[])` parameter value. The comment line is specified in the following format:

```
@return type[] substitution comment
```

where:

- *`type`* is the data type of the response.
- `[]` indicates an array of values. There is no white space between the response type and `[]`.
- *`substitution`* is a form similar to the standard Info*Engine substitution syntax:

  ```
  $(groupName[elementIndex]attributeName[attributeIndex])
  ```

  where:

  - *`groupName`* is the name of the output group.
  - *`elementIndex`* is the position of the element. If not specified, the *`elementIndex`* value defaults to 0, the first element.
  - *`attributeName`* is the name of the attribute.
  - *`attributeIndex`* is the position of the attribute on the element. If not specified, the *`attributeIndex`* value defaults to 0, the first attribute.

  To populate a return array, the asterisk character "`*`" must be present in either the `elementIndex` or `attributeIndex` portion of the substitution syntax, but not both, as two dimensional arrays are not supported. If "`*`" is specified as the `attributeIndex`, then the array is generated from a multi-valued attribute from within a single element of the output group. If "`*`" is specified as the `elementIndex`, then the array is generated using the first value of each `attributeName` from each element of the output group.

When returning a Java bean, the @return comment line is specified as above, with the following two exceptions:

- The `attributeName[attributeIndex]` portion of the substitution syntax is not present.
- To return an array of Java beans, the "`*`" character must be in the `elementIndex` location of the substitution syntax.

The Java bean (or array of Java beans) are populated based on attributes found in referenced elements of the output group. Attribute names in the response group correspond directly to the properties of the Java bean. Attribute names in the response group might need to be manipulated (for example, using the Change-Group webject) to force the group attribute names to match the corresponding properties of the Java bean.

When returning Info*Engine data using the Info*Engine-specific data types, the format for the @return parameter is as follows:

```
@return INFOENGINE_COLLECTION
@return INFOENGINE_GROUP $(group_Name)
@return INFOENGINE_ELEMENT $(groupName[elementIndex])
@return INFOENGINE_ATTRIBUTE $(groupName[]attributeName)
```

where:

- `groupName` is the name of the output group.
- `elementIndex` is the position of the element. If not specified, the `elementIndex` value defaults to 0, the first element.
- `attributeName` is the name of the attribute.

Simple Data Type Examples

The following examples do not represent the types of activities that Info*Engine tasks should be written to perform. Rather, they are intended to offer an example of how input and output parameters function.

Note

Because these tasks contain scriptlet code that assumes strongly-typed input parameters, they therefore cannot be called directly from a web browser using the Info*Engine servlet because the @FORM data would not be properly typed. These special SOAP tasks can be successfully called only by SOAP clients that generate strongly-typed SOAP requests. If the request is not strongly typed, a `ClassCastException` occurs.

The following example specifies two integers as input parameters, and returns their sum:

```
<%@page language="java"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
prefix="ie"%>

<!--com.infoengine.soap.rpc.def

this task takes two integers and adds them together

@param int x
```

```
@param int y

@return int $(output[]sum[])
-->
<%

Integer x = (Integer)getParam ( "x" );
Integer y = (Integer)getParam ( "y" );
String element = "sum=" + (x.intValue()+y.intValue());

%>

<ie:webject name="Create-Group" type="GRP">
   <ie:param name="ELEMENT" data="<%=element%>"/>
   <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
```

The following example returns the sum of an array of floats:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
prefix="ie"%>

<!--com.infoengine.soap.rpc.def
this task sums an array of floats

@param float[] toSum

@return float $(output[0]sum[0])
-->
<%

java.util.Vector floats = getParams ( "toSum" );
Float [] toSum = new Float[floats.size()];
floats.copyInto ( toSum );

float sum = 0.0;
for ( int i = 0; toSum != null && i < toSum.length; i++ )
sum +=toSum[i].floatValue();

String element = "sum=" + sum;
%>

<ie:webject name="Create-Group" type="GRP">
   <ie:param name="ELEMENT" data="<%=element%>"/>
   <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
```

The following example queries an employee database and returns an array of all employee numbers:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
prefix="ie"%>

<!--com.infoengine.soap.rpc.def
this task queries an employee database and returns an array of all
employee numbers
@return string[] $(output[*]empno[])
-->

<ie:webject name="Query-Objects" type="OBJ">
   <ie:param name="INSTANCE" data="jdbcAdapter"/>
   <ie:param name="CLASS" data="EMP"/>
   <ie:param name="WHERE" data="()"/>
   <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
```

Java Bean Examples

Note

These Java bean examples are using a JNDI adapter to create user objects in an LDAP directory. These user objects should not be confused with user objects that a Windchill can use.

The examples in this section use the following Java bean:

```
package my.org;

// exposes properties:
// cn, sn, dn, uid and indexed property mail
// properties could be any primitive types or java beans
// but for the sake of an LDAP example things are just strings

import java.util.Vector;

public class Person {
   private String cn;
   private String sn;
   private String dn;
   private String uid;
   private Vector mails = new Vector ( 1 );
   public void setCn ( String n ) { cn = n; }
   public String getCn () { return cn; }

   public void setSn ( String n ) { sn = n; }
   public String getSn () { return sn; }

   public void setDn ( String n ) { dn = n; }
   public String getDn () { return dn; }

   public void setUid ( String n ) { uid = n; }
   public String getUid () { return uid; }

   public void setMail ( int index, String m ) {
      if ( index == mails.size() )
         mails.addElement ( m );
   else
      mails.setElementAt ( m, index );
  }

   public String getMail ( int index ) {
      return (String)mails.elementAt ( index );
  }

   public void setMail ( String vals[] ) {
      mails.clear();
      for ( int i = 0; vals != null && i < vals.length; i++ )
         mails.addElement ( vals[i] );
  }

   public String [] getMail () {
      if ( mails.isEmpty() ) return null;
   String arr [] = new String[mails.size()];
   mails.copyInto ( arr );
   return arr;
  }
}
```

The following example looks up a user entry by user id (`uid`):

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!--com.infoengine.soap.rpc.def
looks up a user entry by uid

@param string uid
```

```
@return my.org.Person $(output[0])
-->

<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE" data="com.myCompany.myHost.ldap"/>
  <ie:param name="BASE" data="ou=People,o=Company"/>
  <ie:param name="SCOPE" data="subtree"/>
  <ie:param name="FILTER" data="uid=$(@FORM[]uid[])"/>
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>

<ie:webject name="Change-Group" type="GRP">
  <ie:param name="GROUP_IN" data="output"/>
  <ie:param name="GROUP_OUT" data="output"/>
  <ie:param name="RENAME" data="'object'='dn'"/>
</ie:webject>
```

The following example searches for a list of users using an LDAP search filter:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!--com.infoengine.soap.rpc.def
searches for a list of users by an LDAP search filter
@param string filter

@return my.org.Person[] $(output[*])
-->

<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE" data="com.myCompany.myHost.ldap"/>
  <ie:param name="BASE" data="ou=People,o=Company"/>
  <ie:param name="SCOPE" data="subtree"/>
  <ie:param name="FILTER" data=" $(@FORM[]filter[])" default="uid=*"/>
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>

<ie:webject name="Change-Group" type="GRP">
  <ie:param name="GROUP_IN" data="output"/>
  <ie:param name="GROUP_OUT" data="output"/>
  <ie:param name="RENAME" data="'object'='dn'"/>
</ie:webject>
```

The following example creates a user in an LDAP directory. The results are returned in Info*Engine XML format, as there is no @return comment specified:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
```

```
<!--com.infoengine.soap.rpc.def
creates a user in LDAP. returns I*E XML.

@param string dbuser
@param string passwd
@param my.org.Person toCreate
-->
<%
my.org.Person toCreate = (my.org.Person)getParam ( "toCreate" );
String dn = toCreate.getDn();
String uid = "uid=" + toCreate.getUid();
String cn = "cn=" + toCreate.getCn();
String sn = "sn=" + toCreate.getSn();

String [] mails = toCreate.getMail();
StringBuffer sb = new StringBuffer();
for ( int i = 0; mails != null && i < mails.length; i++ )
  sb.append ( "mail=" )
    .append ( mails[i] )
    .append ( ( i < (mails.length-1) ) ? ";" : "" );
%>

<ie:webject name="Create-Object" type="ACT">
  <ie:param name="INSTANCE" data="com.myCompany.myHost.ldap"/>
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="DN" data="<%=dn%>"/>
  <ie:param name="FIELD" data="objectClass=inetOrgPerson"/>
  <ie:param name="FIELD" data="objectClass=person"/>
  <ie:param name="FIELD" data="<%=uid%>"/>
  <ie:param name="FIELD" data="<%=cn%>"/>
  <ie:param name="FIELD" data="<%=sn%>"/>
  <ie:param name="FIELD" data="<%=sb.toString()%>" delim=";"/>
  <ie:param name="GROUP_OUT" data="output"/>
</ie:webject>
```

BLOB Attachments on SOAP Requests

Attachments can be used with SOAP requests to upload and download small amounts of binary data.

Note

Only relatively small amounts of binary data should be transferred using SOAP. For information on uploading and downloading larger BLOBs, see Uploading and Downloading BLOBs on page 54 .

Two special data types are used when attaching data to a SOAP request: `javax.activation.DataSource` and `java.io.InputStream`. These special data types support a specially formatted SOAP comment optionally listing the binary content type. Specifying the content type allows for more complete generated WSDL, including what type of content data is expected or is returned. Any valid content type can be specified, for example:

- `image/.gif`
- `image/.jpeg`
- `application/octet-stream`

If no content type is specified, then the default `application/octet-stream` is used.

`javax.activation.DataSource`

Indicates that the client is attaching binary data. This data type is used only in an @param SOAP comment.

Use the following format:

```
@param javax.activation.DataSource file {contentType:content_type}
```

where *`content_type`* is the content type of the binary data.

`java.io.InputStream`

Indicates that the task responds with binary data. Used only in an @return SOAP comment.

Use the following format:

```
@return java.io.InputStream {contentType:content_type}
```

where *`content_type`* is the content type of the binary data. When the response is an attachment, no substitution syntax is specified.

Info*Engine does not support returning both an attachment and data. When a task produces a BLOB for SOAP, the SOAP response contains an empty response element.

Example BLOB Upload

The following task adds a BLOB to a database row.

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                     prefix="ie"%>

<!--com.infoengine.soap.rpc.def
upload a blob

@param string filename - the filename to store
@param javax.activation.DataSource file - the file to store
(should be gif or jpeg)
-->

<ie:unit>
  <ie:webject name="Do-Sql" type="ACT">
    <ie:param name="INSTANCE" data="soapJDBCAdapter"/>
    <ie:param name="SQL" data="DELETE FROM BLOBTABLE WHERE
                                   name='$(@FORM[]filename[0])'"/>
    <ie:param name="CLASS" data="BLOBTEST"/>
      <ie:param name="GROUP_OUT" data="deleteResult"/>
    <ie:param name="BLOB_COUNT" data="0"/>
  </ie:webject>
  <ie:failure/>
</ie:unit>

<ie:webject name="Do-Sql" type="ACT">
  <ie:param name="INSTANCE" data="soapJDBCAdapter"/>
  <ie:param name="SQL" data="INSERT INTO BLOBTABLE VALUES
('$(@FORM[]filename[0])', NULL)"/>
  <ie:param name="CLASS" data="BLOBTEST"/>
  <ie:param name="GROUP_OUT" data="insertResult"/>
  <ie:param name="BLOB_COUNT" data="0"/>
  </ie:webject>

<ie:webject name="Put-Blob-Stream" type="OBJ">
  <ie:param name="INSTANCE" data="soapJDBCAdapter"/>
  <ie:param name="CLASS" data="BLOBTABLE"/>
  <ie:param name="ATTRIBUTE" data="FILECONTENT"/>
  <ie:param name="WHERE" data="(NAME='$(@FORM[]filename[0])')"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"
                                                default="output"/>
</ie:webject>
```

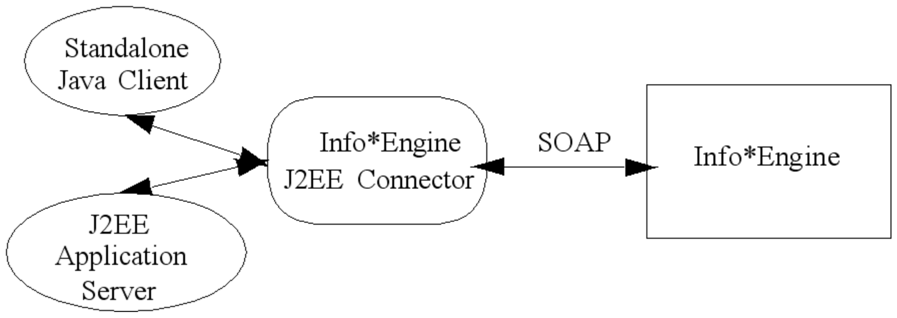
Example BLOB Download

The following task retrieves a BLOB from a database row.

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                     prefix="ie"%>

<!--com.infoengine.soap.rpc.def
download a blob

@param string filename - the file name of the blob/image to
download (value from ListBlobs call)
@return java.io.InputStream a stream that contains the blob
-->

<ie:webject name="Send-Blob-Stream" type="OBJ">
  <ie:param name="INSTANCE" data="soapJDBCAdapter"/>
  <ie:param name="CLASS" data="BLOBTABLE"/>
  <ie:param name="ATTRIBUTE" data="FILECONTENT"/>
  <ie:param name="MIMETYPE" data="'application/octet-stream'"/>
  <ie:param name="WHERE" data="(NAME='$(@FORM[]filename[0])')"/>
  <ie:param name="FILENAME" data="test.doc"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"
                                                default="output"/>
</ie:webject>
```

Info*Engine Java EE Connector

The Info*Engine Java Platform, Enterprise Edition (Java EE) connector is Info*Engine's implementation of Java EE Connector Architecture (JCA) version 1.0. JCA was designed to supply standard implementation for interaction between Java EE application servers and enterprise information systems (EIS), in this case Info*Engine.



The connector is a low level software driver that allows access to Info*Engine from a Java EE application server environment (such as JBOSS or Sun ONE), or a standalone Java client (command-line or swing application). The Info*Engine Java EE connector uses SOAP as its communication protocol. The connector is similar to a JDBC driver, in that the connector understands the low-level aspects of connecting to and interacting with Info*Engine similar to how a JDBC driver connects to and interacts with a relational database.

For more information on configuring your application server environment for use with the Info*Engine Java EE connector, see the appropriate white paper for your environment (Sun ONE or JBoss). These white papers can be found on the Windchill Info*Engine page of the Reference Documents section of the PTC website at the following URL:

www.ptc.com/appserver/cs/doc/refdoc.jsp

The Info*Engine Java EE connector communicates with Info*Engine using SOAP. SOAP is a standard request/response protocol for interacting with a web service using XML. Info*Engine includes the SOAP RPC servlet which allows Info*Engine tasks to be exposed to a client using HTTP or JMS.

Info*Engine can describe a SOAP service to a client at runtime using Web Service Definition Language (WSDL). Delivered along with the Java EE connector are tools that use WSDL metadata information to generate client-side Java classes that can be used to simplify interactions with Info*Engine. The supplied tools can generate Data Access Objects (DAOs) and Enterprise Java Beans (EJBs) that expose standard Java methods that drive interactions with Info*Engine.

JCA Contracts and the Common Client Interface

JCA defines three contracts that a connector must fulfill: connection management, transaction management, and security management. Fulfillment of these three contracts by the Java EE connector allows an application server to plug in value-added services such as connection pooling, automatic transaction management, and security control. JCA also defines the common client interface (CCI). The CCI provides a suggested client API that a Java EE connector can optionally implement. The Info*Engine Java EE connector implements the CCI.

Note

This information is provided as a general overview and is not required to write applications using the Info*Engine Java EE connector.

Connection Contract

A connector must implement the following service provider interfaces:

- `javax.resource.spi.ManagedConnectionFactory`
- `javax.resource.spi.ManagedConnection`
- `javax.resource.spi.ManagedConnectionMetaData`
- `javax.resource.spi.ConnectionManager`

The Info*Engine connector implementation of these classes is as follows:

- `com.infoengine.connector.IeManagedConnectionFactory`
- `com.infoengine.connector.IeManagedConnection`
- `com.infoengine.connector.IeManagedConnectionMetaData`
- `com.infoengine.connector.IeConnectionManager`

These classes are described in the following sections.

Note

Object references in code should be made using the appropriate interface definition and not the implementation class. For example:

```
ManagedConnectionFactory mcf = new IeManagedConnectionFactory ();
```

IeManagedConnectionFactory

The **IeManagedConnectionFactory** class is used to create instances of EIS specific connection factories that can create actual connections to Info*Engine. The class also contains logic required to create and compare physical connections. These additional methods are used by a connection manager, such as an application server or other implementation, to facilitate connection pooling. As an application developer, the only time you might reference this class directly is when you are developing a standalone Java SOAP client. For more information, see Example Standalone Java SOAP Client on page 217.

IeManagedConnection

The **IeManagedConnection** class represents a physical connection to an EIS. As an application developer you do not need to instantiate or interact directly with this class.

IeManagedConnectionMetaData

The **IeManagedConnectionMetaData** class can be used to get general information about a physical connection to an EIS. The Info*Engine connector has merged this class implementation with that of `javax.resource.cci.ConnectionMetaData` as part of the CCI implementation. For more information, see Common Client Interface (CCI) on page 215.

For example, a client could ask for metadata about a connection in the following manner:

```
Connection cx = cxFactory.getConnection();
ConnectionMetaData meta = cx.getMetaData ();
```

IeConnectionManager

The **IeConnectionManager** class is required for managing connections when a connector is used outside of a Java EE application server, such as in a standalone Java SOAP client. An application server's implementation of this class incorporates functions such as connection pooling and automated transaction management. The Info*Engine Java EE connector implementation of this class supplies support for simple connection pooling. It does not support automated transaction management. Transaction support is only available when the Info*Engine Java EE connector is deployed against a Windchill system. When the connector is used by a standalone application, the application must manage transaction demarcation manually. As an application developer, you never need to instantiate or interact directly with this class. For more information, see Example Standalone Java SOAP Client on page 217.

Transaction Contract

The Info*Engine Java EE connector contains **LocalTransaction** support only. Transactions are only supported if the connector is deployed against Windchill. When using the Info*Engine Java EE connector from a Java EE SOAP client, it is possible to have the application server manage transactions for you. When deployed, each EJB method can declare its transaction requirements. When using Windchill and a transacted delegate is invoked from an EJB, the @FORM context group contains information that the task must use to be enclosed within the transaction. The @FORM attributes are:

- **session_instance** — The Windchill instance the transaction is started against. This must be used as the value of the INSTANCE parameter to Windchill adapter webjects that are part of a transaction.
- **session_id** — The session identifier. This must be used as the value of the SESSION_ID parameter to Windchill adapter webjects that are part of a transaction.

Transactions can be demarcated manually if you are executing either from a standalone Java SOAP client or from an EJB that wants to control a transaction itself. For example:

```
Connection cx = cxFactory.getConnection ();
LocalTransaction tx = cx.getLocalTransaction ();
...
```

```
tx.begin ();
...
tx.commit ();
```

Security Contract

The Info*Engine connector supports **BasicPassword** authentication using the `javax.resource.spi.security.PasswordCredential` credential interface. Actual validation occurs with the first interaction. If Info*Engine is running in an insecure mode (for instance, with no HTTP server authentication) then the username and password are ignored as if the given credentials are valid and are passed along to be used by supporting tasks. The `/com/infoengine/connector/validateUser.xml` task is supplied for EJBs to use at creation time to verify that the user can be properly authenticated against the supporting adapter. Automatic EJB generation makes use of this task to validate users at EJB creation time to avoid authentication failures with later interactions.

Common Client Interface (CCI)

Note

Interactions with Info*Engine should ideally be performed through generated Data Access Objects (DAOs) rather than at the CCI level. For more information on DAO generation, see Example Standalone Java SOAP Client on page 217.

The Info*Engine Java EE connector avoids a proprietary API for interaction by implementing the Common Client Interface (CCI).

The CCI simplifies the problem of writing code to connect a client to an underlying EIS's data store. The EIS vendor can use the CCI to write a generic interface to its product. With this one interface, the product works with clients on any Java EE compliant platform. Likewise, application developers only need to learn and use one set of API calls to connect their client to any underlying EIS, rather than having to learn a different API for each EIS.

The following classes are used by the CCI:

- `com.infoengine.connector.IeConnectionSpec`
- `com.infoengine.connector.IeInteractionSpec`
- `com.infoengine.connector.IeInteractionResult`

com.infoengine.connector.IeConnectionSpec

The `com.infoengine.connector.IeConnectionSpec` class allows credentials to be passed during connection creation. This class supports the standard **UserName** and **Password** properties. In addition, this class supports a locale property that allows Info*Engine to generate localized data in response, and an **authUser** property that can be used in Single Sign On (SSO) scenarios. The **authUser** property is only accepted by Info*Engine if the connector is configured to digitally sign outgoing requests or if the client resides on a host that is trusted.

com.infoengine.connector.IeInteractionSpec

The `com.infoengine.connector.IeInteractionSpec` class drives interactions with Info*Engine. The standard **FunctionName** property specifies the method of function to invoke. This value corresponds to the name of the task delegate to invoke. The additional **ClassName** property specifies the class the task delegate belongs to. This value corresponds to a type identifier that contains the task delegate to invoke.

com.infoengine.connector.IeInteractionResult

The `javax.resource.cci.Interaction` implementation supports executing with input record only. It does not support executing with input and output record. The interaction implementation always returns an instance of `com.infoengine.connector.IeInteractionResult` (implements `javax.resource.cci.Record`). The **IeInteractionResult** class is a simple wrapper for the SOAP response. The response object can be retrieved using the **getResult()** method. An interaction accepts as input either **MappedRecord** or **IndexedRecord**.

In the case where **MappedRecord** is passed, the keys in the map are parameter names and the values are parameter values. In the case where **IndexedRecord** is passed, each value must be a *`name`*`=`*`value`* pair of parameters to be passed. Ideally, **MappedRecord** should be used, as the use of *`name`*`=`*`value`* strings does not allow for any data type other than `java.lang.String` to be passed.

The following is an example interaction with Info*Engine:

```
ConnectionFactory cxf = getLDAPFactory (); // look up in JNDI
    Connection cx = null;
    try {
      // get connection with credentials
      IeConnectionSpec cxSpec = new IeConnectionSpec ();
      cxSpec.setUserName ( "wcadmin" );
      cxSpec.setPassword ( "wcadmin" );
      cx = cxf.getConnection ( cxSpec );
      // or anonymous
      //cx = cxf.getConnection ();
      Interaction ix = cx.createInteraction ();
      IeInteractionSpec ixSpec = new IeInteractionSpec ();
      ixSpec.setClassName ( "org.myOrg.Math" );
      ixSpec.setFunctionName ( "sum" );
      RecordFactory rFact = cxf.getRecordFactory ();
      MappedRecord iRec = rFact.createMappedRecord ( "input" );
      iRec.put ( "x", new Integer ( 4 ) );
      iRec.put ( "y", new Integer ( 5 ) );

      IeInteractionResult result =
        (IeInteractionResult)ix.execute ( ixSpec, iRec );
      Integer sum = (Integer)result.getResult ();
      System.out.println ( "sum is " + sum.intValue () );
    } catch ( Exception ex ) {
      ex.printStackTrace ();
    } finally {
      if ( cx != null ) cx.close ();
    }
```

Example Standalone Java SOAP Client

Standalone Java SOAP clients can use the Java EE connector to interact with Info*Engine. They do not, however, provide the added value that an application server can bring to your connections.

Writing a standalone Java SOAP client requires the following steps:

1. Decide on the classes and methods required by your application.
2. Implement the required tasks inside of a directory hierarchy that reflects the required classes.

   In our example this includes the `average.xml` and `sum.xml` tasks in the `org/myOrg/Math` subdirectory of the task root. These are described in Implementing Tasks and Java Classes on page 218.
3. Create the required type identifiers and delegates in the LDAP directory. This is discussed in Registering Delegates on page 219.
4. Generate Data Access Objects (DAOs) for each class from Step 1. DAOs are discussed in Generating DAOs on page 220.
5. Decide how connections are to be created, and if necessary bind connection factory objects in the LDAP directory. This is described in Creating a Connection Handle on page 222 and Managing Connection Factories in the LDAP Directory on page 223.
6. Write client source code that uses the DAOs from Step 4 to access Info*Engine. This is described in Putting It All Together on page 226.

The following sections use a simple example application to demonstrate how a standalone Java SOAP client should communicate with Info*Engine.

Before You Begin Writing the Standalone Java SOAP Client

Before you begin writing your standalone Java SOAP client, you must do one of the following:

- Ensure that the following JAR files are in your CLASSPATH:
  - `$(<`*`Windchill`*`>)/lib/servlet.jar`
  - `$(<`*`Windchill`*`>)/codebase/WEB-INF/lib/ieWeb.jar`

    - `$(<Windchill>)/codebase/WEB-INF/lib/ie3rdpartylibs.jar`

      where <*Windchill*> is the Info*Engine installation directory.

or

- Extract the contents of `$(<Windchill>)/ieconnector/ie.rar` into a directory and add all JAR files found there to your CLASSPATH.

Assumed Knowledge

Before implementing an Info*Engine SOAP client, it is assumed that you have the following knowledge:

- You should be familiar with the Info*Engine SOAP RPC servlet and writing tasks for use with SOAP.
- You should have a solid working knowledge of writing Java applications. These sections do not cover fundamentals such as compiling example source code or setting the CLASSPATH.

Implementing Tasks and Java Classes

For our example, the `sum.xml` and `average.xml` tasks are exposed to a standalone SOAP client using the "org.myOrg.Math" class.

Note

For ease of instruction, the tasks used in this sample application are supplied only as examples, and do not represent the types of activities for which Info*Engine tasks are typically written to accomplish.

sum.xml

The `sum.xml` task takes two integers and returns their sum. The following code is the contents of `/org/myOrg.Math/sum.xml`:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/tag
lib/core"prefix="ie"%>

<!--com.infoengine.soap.rpc.def
this task takes two integers and adds them together
@param int x
@param int y

@return int $(output[]sum[])
-->
```

```
<%
Integer x = (Integer)getParam ( "x" );
Integer y = (Integer)getParam ( "y" );
String element = "sum=" + (x.intValue()+y.intValue());
%>

<ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT" data="<%=element%>"/>
    <ie:param name="GROUP_OUT" data="output"/>
  <ie:webject>
```

average.xml

The `average.xml` task takes an array of numbers and returns their average. The following code is the contents of `/org/myOrg/Math/average.xml`:

```
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/tag
lib/core"prefix="ie"%>

<!--com.infoengine.soap.rpc.def
this task takes an array of numbers and averages
them

@param double[]nums

@return double $(output[]avg[])
-->
<%
java.util.Vector nums = getParams ( "nums" );
double sum = 0;
for ( int i = 0; i < nums.size(); i++ )
  sum += ((Double)nums.elementAt(i)).doubleValue();
String element = "avg=" + (sum/(double)nums.size());

%><ie:webject name="Create-Group" type="GRP">
    <ie:param name="ELEMENT" data="<%=element%>"/>
    <ie:param name="GROUP_OUT" data="output"/>
  <ie:webject>
```

Registering Delegates

For any SOAP client to invoke an Info*Engine task, the task must be registered as a delegate in the LDAP directory.

The tasks from our example would be registered using the type identifier of "org. myOrg.Math." This type identifier would contain two delegates: `sum` and `average`.

For more information on registering tasks as delegates, see the online help available from the Task Delegate Administration utility.

Generating DAOs

Data Access Objects (DAOs) are generated from Info*Engine tasks. Each DAO exposes one method signature per exposed Info*Engine task.

A DAO generated from the "org.myOrg.Math" class would expose the following public method signatures:

```
public int sum ( int x, int y ) throws Exception;
public double average ( double [] nums ) throws Exception;
```

And the following constructor (assuming a class name of "MathDAO"):

```
public MathDAO ( javax.resource.cci.Connection c, javax.
resource.cci.RecordFactory r );
```

Creation of the required instances of `javax.resource.cci.Connection` and `javax.resource.cci.RecordFactory` are discussed in Creating a Connection Handle on page 222.

Generated DAO methods may throw a `java.lang.Exception`. The reason such a generic exception is used is because the underlying classes used to issue a SOAP request can change. For example, the connection can be configured to use HTTP or JMS as the underlying protocol. An HTTP SOAP connection issues a `java.net.ConnectionRefused` exception if the HTTP service is unavailable, where a JMS SOAP connection issues a `javax.jms.JMSException` if a JMS-related error occurs.

DAO generation requires the following information:

- `endPoint`

  This is the location of the Info*Engine SOAP service. This value is optional and defaults to `http://`*`<host>`*`/`*`<Windchill>`*`/servlet/RPC`. Depending on your configuration, the default value may not be suitable, in which case you must explicitly specify this information. If credentials are required to access the service then the URL form of "http://<user>:<password>@host/..." should be used.

- `soapClass`

  This is the base class (type identifier) from which the DAO is generated. In our example this value is "org.myOrg.Math."

- `fileSystem`

This value points to a local directory where the root of your Java source tree is located.

- `package`

  This is the name of the Java package to which the generated source belongs.
- `class`

  This is the name of the class being generated.

There are two ways to generate a DAO: manually invoke a Java command to run the DAO generation tool, or use an Ant extension from an Ant build script.

For the purposes of this example assume the following is true:

- You are developing the standalone Java client on the same host where the SOAP service is running.
- The SOAP service requires the credentials username (`wcadmin`) and password (`wcadmin`).
- The root of your Java source tree is `/home/user/src`.

To generate a DAO for the “org.myOrg.Math” class using a Java command line, invoke the following command (all on one line):

```
java com.infoengine.connector.dao.DAOGenerator endPoint=http://wcadmin:wcadmin@localhost/
Windchill/servlet/RPC soapClass=org.myOrg.Math fileSystem=/home/user/src package=org.
myOrg.Math class=MathDAO
```

An Ant build script that generates the DAO should look similar to the following code:

```
<?xml version="1.0"?>
<project name="generateDAO" default="all" basedir=".">



  <property name="wt.home" value="/opt/ptc/Windchill"/>



  <path id="cp">
    <pathelement location="$(wt.home)/codebase/WEB-INF/
lib/ieWeb.jar"/>

    <pathelement location="$(wt.home)/codebase/WEB-INF/
lib/ie3rdpartylibs.jar"/>
    <pathelement location="$(wt.home)/lib/servlet.jar"/>
  </path>



  <target name="declare">
    <taskdef name="generator" classname="com.infoengine.
connector.dao.AntDAOGenerator">
```

```
      <classpath refid="cp"/>
    </taskdef>
  </target>
  <target name="all" depends="declare">
    <generator
      endPoint="http://wcadmin:wcadmin@localhost/Windchill/
servlet/RPC"
      soapClass="org.myOrg.Math"
      fileSystem="/home/user/src"

      package="org.myOrg.Math"

      class="MathDAO"/>
  </target>


</project>
```

Using either method of DAO generation results in a Java source file being generated with a fully qualified class name of “org.myOrg.Math.MathDAO” (`/home/user/src/org/myOrg/Math/MathDAO.java`).

Creating a Connection Handle

Connection handles (instances of `javax.resource.cci.Connection`) are used to create interactions with Info*Engine. Connection handles do not represent physical connections to Info*Engine. As a result, invoking **Connection.close()** does not necessarily close the physical connection. Instead, it simply frees the connection handle and allows the physical connection to be returned to a connection pool for later reuse.

Connection handles should always be closed when you are finished using them. Ideally, closing of a connection handle should be performed in a FINALLY block to ensure that, regardless of error conditions, the handle is freed properly and the physical connection can be reused. If this is not done, physical connections can remain marked as busy and never be properly cleaned up. The underlying connection manager is responsible for connection pooling and closing bad or expired physical connections. These are details your source code does not need to deal with.

Connection handles can be retrieved using a connection factory (instance of `javax.resource.cci.ConnectionFactory`). There are potentially two ways of getting an instance of a connection factory:

- The connection factory can be manually configured and created in Java source code.
- The connection factory can be retrieved from an LDAP directory.

Looking up a connection factory from an LDAP directory is the preferable method for the following reasons:

- Manually configuring and creating the connection factory in source may require code changes if the SOAP service were to move.
- Storing the connection factory in an LDAP directory allows many clients to share the connection factory's configuration.

  In this situation updating the location of the SOAP service would only need to be reconfigured in a single place, as opposed to wherever a SOAP client resides. As long as the location of the LDAP directory does not change, clients do not need to be updated.

Managing Connection Factories in the LDAP Directory

Info*Engine supplies the `com.infoengine.connector.AdminTool` Java class for managing connection factories in the LDAP directory. This tool allows you to bind a new connection factory or unbind an existing connection factory.

The variables and actions used by the tool are supplied as parameters on the Java command. The properties file containing the connections configuration properties must also be supplied.

Variables

`principal`
The username.

`password`
The password for the user.

Note

Any parameters being specified from the variables section must precede a single parameter from the actions section.

Actions

The following actions are available from the tool:

`-bindConnectionFactory` *`provider object ConnectionImplementation PropertiesFile`*
Where:

- *`provider`* is the LDAP system providing the connection.
- *`object`* is the connection factory being bound.

- *`ConnectionImplementation`* is the connection implementation specified in the properties file.
- *`PropertiesFile`* is the name of the Java properties file that contains the connection configuration properties. This properties file is discussed below.

`-unbindConnectionFactory` *`provider object`*
Where:

- *`provider`* is the LDAP system providing the connection.
- *`object`* is the connection factory being unbound.

Configuration Properties

The following configuration properties can be specified in the Java properties file:

HTTP Connection Implementation
This property applies to an HTTP connection implementation:

`ConnectionURL`
Specifies the endpoint of the Info*Engine SOAP service. For example:

`http://`*`<host>`*`/`*`<Windchill>`*`/servlet/RPC`

JMS Connection Implementation
These properties apply to a JMS connection implementation:

`in.queue`
Specifies the queue to which the SOAP requests are submitted. This property is required.

`out.queue`
Specifies the queue on which to wait for the SOAP responses. This property is required.

`out.queue.wait`
Specifies how long, in milliseconds, to wait for the SOAP response. The default value for this property is `-1`, which means to wait indefinitely for a response.

`provider.url`
Specifies the LDAP URL for the location of the subtree containing administered objects. For example:

```
ldap://localhost/cn=MQSeries,o=MyCompany
```

This property is required.

`provider.principal`
If required by your LDAP access controls, specifies the participant needed to bind to `provider.url`. For example: `cn=Manager`.

`provider.credentials`
: The password for `provider.principal`, if required by your LDAP access controls.

`queueConnectionFactory`
: Specifies the relative distinguished name (**dn**) of the queue connection factory administered object. This property is required.

`queueConnectionFactory.user`
: If required, specifies the username needed to connect to `queueConnectionFactory`.

`queueConnectionFactory.password`
: If required by your LDAP access controls, specifies the password associated with `queueConnectionFactory.user`.

Non-Connection Implementation Specific

The following properties are not connection implementation specific:

`signRequests`
: Enables or disables digital signing of SOAP requests. Possible values are TRUE and FALSE, with FALSE being the default value.

`keyStoreType`
: Specifies the type of keystore. The default value for this property is `JKS`.

`keyStorePackageProvider`
: Specifies the keystore package provider. This property is optional.

`keyStoreFilename`
: Specifies the path to the keystore. The default value for this property is `.keystore` in the user's home directory.

`keyStorePassword`
: Specifies the password for the keystore. This property is required.

`certificateAlias`
: Specifies the alias of certificate to use. The default value for this property is `iesoap`.

`privateKeyAlias`
: Specifies the alias of the private key. The default value for this property is value of the `certificateAlias` property.

`privateKeyPassword`
: Specifies the private key password. The default value for this property is value of the `keyStorePassword` property.

Java Properties File

When creating a connection factory you must supply a Java properties file that contains the connection's configuration properties.

An example of an HTTP connection factory configuration is:

```
#ConnectionImplementation=com.infoengine.connector.HTTPConnection
ConnectionURL=http://host/Windchill/servlet/RPC
```

An example of a JMS connection factory configuration is:

```
#ConnectionImplementation=com.infoengine.connector.JMSConnection
in.queue=cn=SOAP.in
out.queue=cn=SOAP.out
out.queue.wait=60000
queueConnectionFactory=cn=SOAP.qcf
provider.url=ldap://localhost/cn=MQSeries,o=MyCompany
provider.principal=cn=Manager
provider.credentials=admin
```

Assuming the properties listed previously for an HTTP connection factory were stored in a properties file named `http.properties`, the following command could be used to bind a new connection factory to an object at the distinguished name `cn=cxFactory.HTTP,o=`*`MyCompany`*:

```
java com.infoengine.connector.AdminTool -principal=cn=Manager
-password=admin -bindConnectionFactory "ldap://localhost/o=MyCompany"
cxFactory.HTTP com.infoengine.connector.HTTPConnection
./http.properties
```

The following command could then be used to unbind the connection factory just created:

```
java com.infoengine.connector.AdminTool -principal=cn=Manager -password=admin
-unbindConnectionFactory "ldap://localhost/o=MyCompany" cxFactory.HTTP
```

Putting It All Together

The simple Java SOAP client

`/home/user/src/org/myOrg/client/Test.java`

illustrates how to interact with Info*Engine using SOAP from a standalone application:

```
package org.myOrg.client;

import org.myOrg.Math.MathDAO;

import com.infoengine.connector.IeConnectionSpec;

import java.util.Hashtable;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.naming.InitialContext;
import javax.naming.NamingException;

// only used in undesirable local configured and created
// ConnectionFactory situation
import com.infoengine.connector.IeManagedConnectionFactory;
import java.beans.PropertyVetoException;
import java.io.FileInputStream;
import java.io.IOException;

public class Test {

  private static ConnectionFactory getLDAPFactory ()
    throws NamingException {

    Hashtable env = new Hashtable ( 5 );
    env.put ( "java.naming.factory.initial",
              "com.sun.jndi.ldap.LdapCtxFactory" );
    env.put ( "java.naming.provider.url",
              "ldap://ldap.mycompany.com/o=MyCompany" );
    env.put ( "java.naming.security.authentication", "simple" );
    env.put ( "java.naming.security.principal", "cn=Manager" );
    env.put ( "java.naming.security.credentials", "admin" );
    InitialContext ctx = new InitialContext ( env );
    return (ConnectionFactory)ctx.lookup (
        "cn=cxFactory.HTTP " );
  }

  private static ConnectionFactory getManualFactory ()
    throws PropertyVetoException,IOException,ResourceException {

    IeManagedConnectionFactory mcf =
      new IeManagedConnectionFactory ();
    // following optional since HTTPConnection is the default
    mcf.setConnectionImplementation (
      "com.infoengine.connector.HTTPConnection" );
    // configure from properties file
    mcf.loadConnectionProperties (
      new FileInputStream ( "./http.properties" ) );
    // or via method call
    // mcf.setConnectionProperties (
    //   "ConnectionURL=http://localhost/Windchill/servlet/RPC" );
    return (ConnectionFactory)mcf.createConnectionFactory ();
  }

  private static String formatArray ( double [] arr ) {
    StringBuffer sb = new StringBuffer ();
    for ( int i = 0; i < arr.length; i++ )
      sb.append ( arr[i] )
        .append ( ( (i+1) < arr.length ) ? " + " : "" );
    return sb.toString ();

  }
  public static void main ( String [] args ) throws Exception {
    ConnectionFactory cxf = getLDAPFactory ();
    //ConnectionFactory cxf = getManualFactory ();

    Connection cx = null;
    try {
      // get connection with credentials
      IeConnectionSpec cxSpec = new IeConnectionSpec ();
      cxSpec.setUserName ( "wcadmin" );
      cxSpec.setPassword ( "wcadmin" );
      cx = cxf.getConnection ( cxSpec );

      // or anonymous
      //cx = cxf.getConnection ();

      MathDAO dao = new MathDAO ( cx, cxf.getRecordFactory () );

      int fourAndFive = dao.sum ( 4, 5 );
      System.out.println (
        "4 + 5 = " + fourAndFive );

      double [] nums = new double [] { 4, 5, 6, 7 };
      double avg = dao.average ( nums );
      System.out.println (
        "average of " + formatArray ( nums ) + " = " + avg );

    } finally {
      if ( cx != null ) cx.close ();
      System.exit ( 0 );
    }
  }
}
```

11
Server Access Kit

The following topics discuss the Info*Engine Server Access Kit (SAK) and provide an example of an SAK application.

About the Server Access Kit

The Info*Engine Server Access Kit (SAK) allows Java applications to execute Info*Engine tasks, pass parameters to them, and inspect their results. Tasks are instantiated as Java classes. After constructing a task instance, methods on the task object can be called to add parameters and establish execution options before the task is invoked. After adding parameters and establishing options, the **invoke** method for the task can be called to execute the task. When the task completes, methods on the task object can be called to obtain the group objects it produced.

Documentation for the Info*Engine APIs that are available through the SAK can be found in the Javadoc files that reside in the `codebase/infoengine/docs/apidocs` directory where Info*Engine is installed.

Connecting to Info*Engine

Before creating or attempting to execute any task instances, an application must initialize its connection with the Info*Engine Naming Service. The Naming Service enables the tasks executed by the application to locate adapters and other services deployed across the enterprise network. The Naming Service connection is established by calling a static factory method as follows:

```
import com.infoengine.au.NamingService;
  :
  :
NamingService namingService =
   NamingService.newInstance
      ("com.myCompany.namingService",
         "ldap://ldap.mycompany.com/dc=myCompany,dc=com");
```

This static factory method locates a Naming Service definition named “com. myCompany.namingService” (a Naming Service that serves a particular enterprise) in the LDAP directory. It searches the LDAP directory tree for the Naming Service definition. The root where the search starts is defined by the directory entry with the distinguished name `dc=myCompany,dc=com` and the host on which the LDAP server resides is named “ldap.mycompany.com”. This value could be a path to the `ie.properties` file or an LDAP URL that points to the place in the directory to find the properties. To avoid confusion it is best to point to the `ie.properties` file.

You must replace this Naming Service name, the distinguished name, and the LDAP host name with ones that apply to your network and LDAP directory structure. The LDAP directory structure and the Naming Service definition are typically created when the Info*Engine software is first installed.

Upon locating the Naming Service definition, the SAK reads its connection parameters, automatically loads all Info*Engine configuration properties from the LDAP directory service, and then returns control to the calling application. At that point, the application can create and execute task instances.

Executing Tasks

To create an instance of a task object, use the constructor that takes a URL as a parameter. For example:

```
import com.infoengine.SAK.Task;
  :
  :
Task task = new Task ("/com/acme/infoapp/QueryBOM.xml");
```

The parameter of this **Task** constructor is the URL of an Info*Engine task. A variety of URL formats can be specified. For example URLs, see Info*Engine JSP Pages on page 63.

After constructing a task instance, the application can add parameters to the task by calling the **addParam** method. This method accepts the name of the parameter to be added and its value. For example:

```
import com.infoengine.SAK.Task;
  :
  :
Task task = new Task ("/com/acme/infoapp/QueryBOM.xml");
task.addParam ("class", "wt.part.WTPart");
task.addParam ("where", "name='Engine'");
task.addParam ("group_out", groupOutName);
```

The same parameter name can be specified in more than one call to **addParam** in order to add multivalued parameters. Parameter values can be of any object type. Parameters are referenced within a task by using normal Info*Engine substitution against the @FORM context group. Thus, each call to the method **addParam** adds an attribute with the specified name and value to the @FORM context group of the task to be executed.

In addition to adding parameters to a task, an application can also establish execution options against the task before invoking it. For example, an application can set the username under which the task should execute. This username can then be used by Info*Engine as a credentials mapping key to select the credentials (for example, username and password) that are provided to each adapter with which the task communicates. The task's username can be set as follows:

```
task.setUsername ("guest");
```

This sets the **auth-user** attribute in the @SERVER context group of the task. Other attributes can be added to the @SERVER context group also. This context group usually contains information about the runtime environment in which a task is executing. For example, when a task is executed using the Info*Engine servlet, the @SERVER context group is populated with attributes derived from information provided by the web server including all HTTP protocol headers.

Applications that create and execute task instances can use the **setServerAttribute** method to populate the @SERVER group with runtime environment information. For example:

```
task.setServerAttribute ("accept-language", "en-US");
```

By default, Info*Engine executes tasks within the Java Virtual Machine (JVM) of the calling application. Sometimes it is more appropriate to execute tasks in a non-local JVM that is hosting an Info*Engine task processor. This is particularly true when the source code of the task can be accessed only by that non-local task processor. For example, if the task source contains sensitive information, such as privileged passwords, access to it may be restricted. When a task must be executed by a non-local task processor, an application can call the **addProcessor** method to specify the names of the Info*Engine task processors in which the task can be executed. For example:

```
task.addProcessor ("com.mycompany.engineering.windchill");
```

After adding all necessary parameters and service options, the task is executed by calling its **invoke** method as follows:

```
task.invoke ();
```

Catching Errors

If any error conditions are detected while executing the task, Java exceptions are thrown. The calling application can use normal Java try/catch logic to catch and inspect exceptions when that is desirable. Of course, if the task executes successfully without detecting any error conditions, no exceptions are thrown.

When the **addProcessor** method is used to specify that a task must be executed in a non-local task processor, Info*Engine automatically establishes communications with the task processor, conveys the request to execute the non-local task (including transmission of all parameters and service options), and retrieves all results. If a remotely executed task throws an exception, the exception is conveyed back to the local JVM and rethrown there. If more than one task processor is specified by calling **addProcessor** more than once, Info*Engine attempts to establish communications with the specified processors in the order in which their names were added. The task is executed in the first processor with which communications are established successfully.

In general, when Info*Engine detects error conditions during the course of executing tasks, it throws exceptions. When underlying services and APIs used by Info*Engine throw exceptions, Info*Engine wraps these in its own exception classes. Consequently, all Info*Engine exception classes can contain nested **Throwable** objects. The Info*Engine exception classes provide methods for obtaining these nested **Throwable** objects so that applications can determine the root causes of error indications.

For details on the `com.infoengine.util.IEException` class and its subclasses, see the Javadoc.

Inspecting Task Results

The application resumes control when the task completes its execution. The application can then inspect the results. Results are usually returned as Info*Engine group objects stored in the task Virtual Database (VDB). The names of all groups currently stored in the VDB at any point in time can be obtained by calling the **getGroupNames** method as follows:

```
java.util.Enumeration groupNames = task.getGroupNames ();
```

Any group can be obtained by name from the VDB using the **getGroup** method provided by the **Task** class. For example:

```
import com.infoengine.SAK.Task;
import com.infoengine.object.factory.Group;
  :
  :
Task task = new Task ("/com/acme/infoapp/QueryBOM.xml");
task.addParam ("class", "wt.part.WTPart");
task.addParam ("where", "name='Engine'");
task.addParam ("group_out", "engine_parts");
task.invoke ();
Group group = task.getGroup ("engine_parts");
```

Groups can be added to the VDB by an application, too. This is particularly useful when a task is designed to operate upon a group that is generated by an application. In this case, the application adds the group to the task's VDB before calling the **invoke** method. For example:

```
import com.infoengine.object.factory.Group;
import com.infoengine.SAK.Task;
  :
  :
Group bomGroup = new Group ("product-structure");
  :
<bomGroup generated>
  :
Task task = new Task ("/com/acme/infoapp/UpdateBOM.xml");
task.addParam ("group_in", bomGroup.getName ());
task.addGroup (bomGroup);
task.invoke ();
```

Resulting Data

A **Group** object (class `com.infoengine.object.factory.Group`) is a container for **Element** objects (class `com.infoengine.object.factory.Element`), and an **Element** object is a container for **Att** objects (class `com.infoengine.object.factory.Att`). Usually, an **Element** object represents a business object or a row from a database table (in the case of simple database adapters). An **Att** object represents a single attribute of a business object. Each **Att** object has a name and one or more values. A value of an **Att** object can be any class of object. Therefore, a **Group** is a container for one or more business objects. Each business object is represented as an **Element** object and each **Element** object contains one or more name/value pairs represented as **Att** objects.

Resulting Metadata

In addition to regular data, **Group**, **Element**, and **Att** objects can contain metadata. Applications can, and Info*Engine itself does, use metadata to register information about an object on the object itself. Each metadata item is a name/value pair. Most metadata items created and used by Info*Engine have names beginning with `com.infoengine`. Applications can add their own metadata items, but they should avoid using names that begin with `com.infoengine` because this namespace is reserved by Info*Engine. All of the information classes provide methods that support inspection, creation, and management of metadata. For details on these methods, see the Javadoc.

Common Methods

The **Group**, **Element**, and **Att** classes all support a variety of methods for inspecting and managing their content. For details on these methods, see the Javadoc. These classes commonly include the following methods:

Common Group Methods

getAttributeValue
Retrieves the value of a specific attribute (**Att**) of a specific **Element** object contained in the **Group**.

addElement
Adds a new **Element** object to the **Group**.

getElementCount
Returns the number of **Element** objects contained in the **Group**.

getElements
Returns all of the **Element** objects of the **Group** as an enumeration.

getElementAt
Retrieves a specific **Element** object from the **Group** by index.

removeElementAt
Removes a specific **Element** object from the **Group** by index.

toXML
Renders the entire **Group** to XML and returns the result as a string or writes it to an output stream.

Common Element Methods

addAtt
Adds a new attribute (**Att**) to the **Element**, or adds a new value to an existing attribute of the **Element**.

getAtt
Retrieves a specific attribute (**Att**) object from the **Element** by name.

getAtts
Returns all attribute objects of the **Element** as an enumeration.

getValue
Retrieves the value of a specific attribute (**Att**) of the **Element**.

getValues
Retrieves all values of a specific attribute (**Att**) of the **Element** as a vector.

removeAtt
Removes a specific attribute (**Att**) from the **Element** by name.

toXML
Renders the **Element** to XML and writes it to an output stream.

Common Att Methods

addValue
Adds a value to the attribute.

getValue
Retrieves the value of the attribute.

getValues
Retrieves all values of the attribute as an enumeration.

toXML
Renders the attribute to XML and writes it to an output stream.

SAK Example Method

The following example method traverses a **Group** object to display all of its **Element** objects and their attributes:

```
private void displayObjects (Group objects) {
 //
 // xmlDisplay is a boolean instance variable that defines
// whether the Group should be rendered as XML or as plain
// text.
//

   if ( xmlDisplay ) {
       PrintWriter pw = new PrintWriter (System.out);
       objects.toXML (pw, true, true);
   }
   else {
       int n = objects.getElementCount ();
       for ( int i = 0; i < n; ++i ) {

    //
    // Print one empty line between Elements
    //
        System.out.println ("");
        Element object = objects.getElementAt (i);
        Enumeration atts = object.getAtts ();
        while ( atts.hasMoreElements () ) {

    //
    // Render each attribute as "name: value"
    // If an attribute is multivalued, indent each
    // value such that they are vertically aligned.
    //

            Att att = (Att)atts.nextElement ();
            String name = att.getName ();
            System.out.print (name + ": ");
            Enumeration values = att.getValues ();

            int v = 0;

            while ( values.hasMoreElements () ) {

              Object value = values.nextElement ();

                 if ( v++ > 0 ) {

                     System.out.print ("  ");

          for ( int s = 0; s < name.length (); ++s )
                        System.out.print (" ");
                   }

            System.out.println (value);
            }
          }
      System.out.println ("");
    }
  }
```

Example SAK Application

The following example application provides a simple example that shows the use of the Server Access Kit (SAK). The example is named `ExecuteTask.java` and is located in the `prog_examples/SAK_apps` directory where Info*Engine is installed.

The example application executes a task supplied when you run the application.

Compile the Application

To compile the application:

1. Set your CLASSPATH environment variable.

   Your CLASSPATH environment variable must include the following files/ folders:

   %JAVA_HOME%/lib/tools.jar

   %*<Windchill>*%/lib/servlet.jar

   %*<Windchill>*%/codebase

   %*<Windchill>*%/codebase/WEB-INF/classes

   %*<Windchill>*%/codebase/WEB-INF/lib/ieWeb.jar

   Example:

   ```
   set
   CLASSPATH=C:/jdk1.3.1/lib/tools.jar;C:/ptc/Windchill/lib/servlet
   .jar;C:/ptc/Windchill/codebase;C:/ptc/Windchill/codebase/WEB-INF/
   classes;C:/ptc/Windchill/codebase/WEB-INF/lib/ieWeb.jar
   ```
2. Create the following directory:

   %*<Windchill>*%/codebase/WEB-INF/classes
3. Change to the directory:

   %*<Windchill>>*%/prog_examples/SAK_apps

and execute the following:

```
javac -d %<Windchill>%/codebase/WEB-INF/classes ExecuteTask.java
```

Run the Application

To run the application, use the following syntax:

```
   java com.infoengine.examples.applications.ExecuteTask
   -t <task_uri>
   -u <username>
   -props <property resource>
   -n <naming service name>
```

where:

- `<task_uri>` is the URI of the task to be executed
- `<username>` associates the username specified with the task
- `<property resource>` is the directory path to the property resource file `ie.properties`
- `<naming service name>` uses the specified Naming Service

For example:

```
   java com.infoengine.examples.applications.ExecuteTask -t
C:/ptc/Windchill/tasks/infoengine/examples/CreateGroup.xml -u
wcadimn    -props C:/ptc/Windchill/codebase/WEB-INF/ie.properties
-n com.ptc.host.namingService
```

Application Source

The following code implements the example:

```
package com.infoengine.examples.applications;

import com.infoengine.au.NamingService;
import com.infoengine.object.factory.Group;
import com.infoengine.SAK.Task;
import com.infoengine.util.IEException;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Enumeration;
import java.util.Vector;

public class ExecuteTask {

    /**
     * Demonstrate the com.infoengine.SAK interface of Infoengine.
     *
     * command line: java com.infoengine.examples.applications.ExecuteTask
     * -t <task uri>
     * -p <processor>
     * -u <username>
     * -n <naming service name>
     * -props <property resource>
     *
     * @param args
     */
    public static void main ( String [] args ) {

      String taskUri = null;
        String processor = null;
        String username = null;
        String namingService = null;
        String props = null;

        for ( int i = 0; i < args.length; i++ ) {
            if ( args[i].equals ( "-t" ) ) {
                if ( i < args.length-1 )
                    taskUri = args[++i];
            }
            else if ( args[i].equals ( "-p" ) ) {
                if ( i < args.length-1 )
                    processor = args[++i];
            }
            else if ( args[i].equals ( "-u" ) ) {
                if ( i < args.length-1 )
                    username = args[++i];
            }
            else if ( args[i].equals ( "-n" ) ) {
                if ( i < args.length-1 )
                    namingService = args[++i];
            }
            else if ( args[i].equals ( "-props" ) ) {
                if ( i < args.length-1 )
                    props = args[++i];
            }
            else {
                System.err.println
                    ("unrecognized parameter \"" + args[i] + "\"");
                usage ();
            }
        }
        if ( taskUri == null ) {
            System.err.println ( "missing required parameter -t <task uri>" );
            usage ();
        }
        if ( namingService == null )
            namingService = "namingService";
        if ( props == null )
            System.err.println
               ("missing required parameter -props <property resource>");
            usage ();
        // Initialize the naming service and read the infoengine properties
        try {
            NamingService ns =
                NamingService.newInstance (namingService, props);
        }
        catch ( FileNotFoundException f ) {
            System.err.println (props + ": property file not found");
            System.exit (1);
        }
        catch ( Exception e ) {
            System.err.println (props + ": unable to read properties");
            e.printStackTrace (System.err);
            System.exit (1);
        }
        // Create a Task object to execute
        Task task = null;
        if ( processor == null ) {
            task = new Task (taskUri);
        }
        else {
            Vector procVector = new Vector ();
            procVector.addElement (processor);
            task = new Task (taskUri, procVector);
        }
        // execute the task
        try {
            task.invoke ();
        }
        catch ( Exception e ) {
            e.printStackTrace (System.err);
            System.exit (1);
        }
        // retrieve and print the results
        Enumeration groups = task.getGroupNames ();
        System.out.println ("groups produced by " + taskUri + ": " + groups);
        PrintWriter pw = new PrintWriter (System.out);
        while ( groups.hasMoreElements () ) {
            Group group = task.getGroup ((String)groups.nextElement ());
            group.toXML (pw, true, true);
        }
        System.exit ( 0 );
```

```
    }
    /**
     * Print the command line parameters
     */
    public static void usage () {
        System.err.println
            ("java com.infoengine.examples.applications.ExecuteTask");
        System.err.println ("\t-t <task uri>");
        System.err.println ("\t-p <processor>");
        System.err.println ("\t-u <username>");
        System.err.println ("\t-n <naming service name>");
        System.err.println ("\t-props <property resource>");
        System.exit (1);
    }
}
```

12

Info*Engine Custom Tag Reference

This following topics provide reference information for the tags contained in the Info*Engine core and directory tag libraries. They also contain an overview of common JSP element types, with tips for using the elements on Info*Engine JSP pages. The elements described in these topics can be used when creating Info*Engine JSP pages and Info*Engine standalone tasks.

Scriptlets

A scriptlet is a standard JavaServer Page (JSP) element that is commonly used in Info*Engine standalone tasks and on Info*Engine JSP pages. It is a valid Java code fragment.

Standalone Task Use

The task compiler copies scriptlets directly to the Java source code that is passed to the Java compiler. This allows Java programming logic to be embedded within Info*Engine tasks in order to implement conditional execution (if/then/else), iteration (for/while), and nontrivial computation.

Tasks can include embedded Java scriptlets (`<%` … `%>`).

Note

The following applies to the use of scriptlets when you have configured a task to use guaranteed task execution (if you have set `ENABLERECOVERY` to TRUE on the `page` directive).

Be careful when positioning scriptlets in tasks used in an Info*Engine environment where the guaranteed task execution feature has been implemented. Intermixing scriptlets with action webjects or **task** tags can interfere with the code generation used for supporting guaranteed task execution. To avoid problems, do one or more of the following:

- Place your scriptlet either before the first webject or **task** tag, or after the last.
- Avoid surrounding action webjects or **task** tags with scriptlets. For example, do not include the webjects or **task** tags in the `for` or `while` loops or in if/then/else constructs.
- Specify the `resumable=true` attribute in all of your action webjects and **task** tags to disable the guaranteed task execution feature for the action webjects and tasks.

The last option is recommended in the cases where tasks are processed immediately when Info*Engine encounters the code rather than those submitted to a queue for execution.

JSP Use

Scriptlets are executed when the JSP interpreter processes the page containing them. This means that nested scriptlets are executed at the point when they are read. However, Info*Engine custom tags are defined so that the execution of nested custom tags is deferred until the end tag of the outermost block is reached. Therefore, any nested scriptlet on a JSP page is executed outside of context of the custom tag block in which it is nested.

You should not nest scriptlets in any Info*Engine tag block on a JSP page. To provide sophisticated solutions using the Java programming logic available through nested scriptlets, do not embed the solution in a JSP page. Instead, create standalone tasks that then can be executed on the page through the use of the Info*Engine **task** tag.

Tag Syntax

`<% scriptlet %>`

Example

The following scriptlet builds a `where` clause for querying a relational database table:

```
<%
   String column = request.getParameter ( “column” );
   String query = request.getParameter ( “query” );
   if ( column == null )
       column = “AUTHOR”;
   String query = request.getParameter ( “query” );
   String whereClause = "SELECT * FROM BOOK WHERE upper(" + column +
                      ") LIKE upper('%" + query + "%')";

%>
<ie:webject name="Do-SQL" type="OBJ">
   <ie:param name="INSTANCE" data="jdbcAdapter"/>

   <ie:param name="SQL" data="<%=whereClause%>"/>
   <ie:param name="GROUP_OUT" data="books"/>
</ie:webject>
```

Expressions

An expression specifies an arbitrary Java expression.

Standalone Task Use

The task compiler generates Java source code that evaluates the Java expression and then embeds the results within the task at the point where the expression resides in the task.

Note

In tasks, expressions can only be placed within the attributes of the **param** tag. Where previously only the **data** attribute could contain an expression, now all attributes can be used, including the following: **name**, **data**, **delim**, **default**, **valueSeparator**, and **elementSeparator**.

JSP Use

The expression is evaluated and its result is displayed at the point in the JSP page where the tag is defined.

Note

The use of expressions within Info*Engine custom **param** tag attribute values may not give you the results you expect when you are using the Tomcat servlet engine. In the Tomcat servlet engine, when an expression is embedded within a **data** attribute value, it is not evaluated correctly. However, if the entire value for the **data** attribute is an expression, the result is correct.

To work around this issue, you can create the entire value in a scriptlet, and then use the results in an expression in the **data** attribute. For example, the following **data** attribute on a **param** tag is not evaluated correctly:

```
<ie:param name="ELEMENT" data="ENAME=<%=ENAME%>"/>
```

Instead, you can code this as follows:

```
<% String elValue = "ENAME=" + ENAME; %>
<ie:param name="ELEMENT" data="<%=elValue%>"/>
```

Tag Syntax

```
<%= expression %>
```

Example

In the following paragraph, two expressions are used to insert the Java variables **table** and **where**:

```
<p><b>
Query table <%= table%> with where clause <%= where%>.
</b></p>
```

Declarations

A declaration specifies text that declares variables and methods that are used on the JSP page or Info*Engine standalone task.

Standalone Task Use

The task compiler emits the text in each declaration at the outermost scope of the Java class it generates for the task. Therefore, the declarations become static or instance variables and methods on the class. You can also use declarations to define static initialization blocks for the generated class.

JSP Use

Declarations are initialized when the JSP page is initialized and are made available to other declarations, scriptlets, and expressions.

Tag Syntax

`<%! declaration %>`

Example

The first declaration declares an integer and initializes it to zero, the second declaration declares a method:

```
<%! int i = 0; %>

<%! public int add ( int a, int b ) { return a+b; } %>
```

Directives

Directives provide global information that is conceptually valid independent of any specific request received by the JSP page or by the standalone task. For example, a directive is used to define a prefix that is required for Info*Engine custom tags.

You can use the following directives on JSP pages and in standalone tasks:

- The `page` directive defines the general characteristics of a JSP page.
- The `taglib` directive performs the following actions:
  - Declares that the page or task uses a tag library.
  - Uniquely identifies the tag library using a URI.
  - Associates a tag prefix that distinguishes usage of the actions in the library.

page Directive

The `page` directive defines the general characteristics of a JSP page. The `page` directive is a standard JSP directive and is also used in standalone tasks. You include the `page` directive as the first tag on an Info*Engine JSP page and the first line in a task file.

JSP Use

Use this directive to specify the characteristics of the page, such as:

- Identify the Java language used in scripts on the page
- Specify the MIME type and character encoding
- Specify the page scope
- Import any class files or packages needed for the page
- Determine what page is displayed for errors
- Identify the page as the error page

Standalone Task Use

The task compiler recognizes a standard `page` directive, but only uses the **import** attribute for importing the necessary class files and the **access** attribute for determining how a task can be invoked.

Note

The **access** attribute is accepted only by tasks, not JSPs.

Tag Syntax for Common Page Characteristics

```
<%@page language="java"
         session="boolean"
         import="list_of_classes"
         contentType="mimeType [;charset=characterSet]"
         errorPage="error_page_url"
         isErrorPage="boolean"
         access="http|soap|internal"%>
```

Attribute Descriptions

Required attributes: **language**

access

A list of access methods acceptable to an Info*Engine task. The possible values are:

- `internal` — The task is invoked by callers internal to the Windchill VM. No external clients are allowed to invoke a task. This value is implicit.
- `soap` — The task is called by SOAP clients.
- `http` — The task is called by raw HTTP clients through the IE servlet.

Note

Windchill uses `internal` task access by default. Therefore, you must explicitly specify task access for tasks that are intended to be called by external clients.

The **access** attribute is accepted only by tasks, not JSPs.

Values can be specified as a pipe-separated list. For example, use `access=`“`http|soap`” for tasks that can be invoked using a SOAP client or the IE servlet.

Best Practice

In situations where several tasks compose a service, you can specify a value for the subdirectory to avoid editing each individual task. To do this, update the `.delegateInfo` file in the subdirectory where your tasks are installed. For example:

```
<Windchill>/tasks/com/ptc/windchill/ws/
.delegateInfo
```

The `.delegateInfo` file is a simple Java properties file, so in this case you would not use quotes when specifying access. For example, `access=soap`.

contentType

The MIME type and character encoding the JSP file uses for the response it sends to the client. You can use any MIME type or character set that is valid for the JSP. The default MIME type is `text/html` and the default character set is `ISO-8859–1`.

errorPage

Defines the URL to the error page for the current page. Any errors not caught on the page cause the page named in this attribute to execute. The error page named must include the `isErrorPage="true"` attribute in its `page` directive.

import

Identifies the Java class files or packages that are made available to the scripting language that is on the page or in the task. You can specify a comma-separated list containing either fully qualified type names or package names followed by "*". Specifying a package name makes all public types in the package available.

isErrorPage

Indicates whether the current page is used as the target of another JSP page **errorPage** attribute. To use the current page as an error page for another page, set this attribute to `true`. By default, this attribute is set to `false`.

language

Defines language that is used in the page. The value of this attribute must be `java`.

This attribute is required.

session

Sets the scope for the page. To set the page scope to the session, set the session attribute to `true`. To set the page scope to the current page, set the session attribute to `false`.

This is an optional attribute. If you omit this attribute, the page scope is the session.

Example

The following `page` directive sets the language to Java, sets the scope of the page to the session, imports the Info*Engine factory and object packages, and sets the error page to `IEError.jsp`:

```
<%@page language="java" session="true"
      import="com.infoengine.object.factory.*,
              com.infoengine.object.*"
      errorPage="IEError.jsp"%>
```

For additional information about error pages, see Catching Exceptions on page 81.

taglib Directive

The `taglib` directive declares that a JSP page or Info*Engine standalone task uses custom tags defined in a tag library. You must put this directive before any lines that use the custom tags in the library.

Tag Syntax

`<%@taglib uri="ie_uri" prefix=“ie_prefix" %>`

Attribute Descriptions

Required attributes: **prefix** and **uri**

prefix

Defines the prefix that distinguishes tags provided by a given tag library from those provided by other tag libraries. If you specify multiple `taglib` directives, each prefix must be unique.

This attribute is required.

uri

Identifies an Info*Engine tag library, where the URI is:

`http://www.ptc.com/infoengine/taglib/`

followed by the tag library name.

This attribute is required.

Example

The following `taglib` directive declares that the page contains custom tags from the Info*Engine core tag library and that the tags use the `ie` prefix:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
              prefix="ie" %>
```

Info*Engine Tags

Info*Engine tags provide access to a set of custom actions that encapsulate recurring functionality so that the same functionality can be reused in multiple Info*Engine JSP pages or standalone tasks.

Using Info*Engine tags reduces the necessity to embed large amounts of Java code in JSP pages and in standalone tasks, and allows you to quickly create the pages and tasks that are required for your application.

The tags for Info*Engine JSP pages are provided in the following tag libraries:

- Core
- Directory
- Supplied

Note

The tags in the directory tag library are not supported in standalone tasks.

To use the Info*Engine tags, you must include the `taglib` directive, which identifies the tag library and provides a prefix for uniquely identifying the tags on the page. For example, to use the directory Info*Engine tags and identify them by the `iedir` prefix, include the following directive:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>
```

The syntax for each tag described here assumes that the `taglib` prefix attribute is set to one of the following. However, if you specify a different prefix, use the prefix you specified in place of the supplied tag syntax examples shown here:

Tag Library	taglib Prefix
Core	`ie`
Directory	`iedir`
Supplied – JSTL	`c`
Supplied – XSL	`xsl`
Supplied – Logging	`log`

The following Info*Engine tags are available:

Tag	Tag Library and Description
choose on page 290	Supplied Tag Library (JSTL). Allows for conditional evaluation with an alternative for evaluation failure (if/else).
createObjects on page 283	Directory Tag Library. Creates new entries in the specified LDAP directory service.
deleteObject on page 285	Directory Tag Library. Deletes an LDAP entry from the LDAP directory service.
displayResource on page 256	Core Tag Library. Extracts and displays a string directly from a resource bundle or from a group created with the Get-Resource webject.
failure on page 258	Core Tag Library. Allows you to supply code for failure processing within a unit.
forEach on page 259	Core Tag Library. Allows you to iterate through an existing Info*Engine group, one element at a time.
forEach on page 291	Supplied Tag Library (JSTL). Supports iteration over an iterable, group, or array of objects. Provides JSTL support for Info*Engine tasks.
getService on	Core Tag Library. Establishes a variable reference to the

Tag	Tag Library and Description
page 261	Info*Engine object (`com.infoengine.jsp.InfoEngine`) that is being used by the Info*Engine custom tags on a JSP page.
getValue on page 263	Core Tag Library. Retrieves the string value of the specified attribute from the first element in the input group.
if on page 291	Supplied Tag Library (JSTL). Allows for conditional evaluation of its body. Provides JSTL support for Info*Engine tasks.
info, debug, warn, error, trace on page 296	Supplied Tag Library (Logging). One tag for each corresponding log4j log level. Each issues a log message to the corresponding log method.
init on page 265	Core Tag Library. Supplies the initialization for a unit.
listObjects on page 285	Directory Tag Library. Creates an Info*Engine group containing elements that are the relative distinguished names of the LDAP directory entries located directly under the specified base directory level.
otherwise on page 292	Supplied Tag Library. (JSTL). If the **when** tag evaluates to `false`, then the content of the **otherwise** tag is invoked and executed (representing the “else” of if/else logic). Provides JSTL support for Info*Engine tasks.
parallel on page 267	Core Tag Library. Allows you to define a set of webjects or tasks that are executed concurrently.
param on page 268	Core Tag Library. Defines a parameter for use within webject tags, task tags, and directory service tags.
param on page 296	Supplied Tag Library (XSL). Specifies a value for an XSL parameter to be supplied for the transformation. This tag is embedded within a **transform** tag and is optional. Provides XSL transformation tag support.
queryObjects on page 286	Directory Tag Library. Creates an Info*Engine group containing elements that are the LDAP directory entries matching the search criteria specified.
resetService on page 272	Core Tag Library. Resets the Info*Engine object (`com.infoengine.jsp.InfoEngine`) that is being used by the Info*Engine custom tags on a JSP page or in a session.
return on page 293	Supplied Tag Library (JSTL). Terminates task invocation. Provides JSTL support for Info*Engine tasks.
set on page 293	Supplied Tag Library (JSTL). Sets a variable in either the tasklet context or request context. Provides JSTL support for Info*Engine tasks.
setLogger on	Supplied Tag Library (Logging). Sets the log4j logger to be

Tag	Tag Library and Description
page 295	used.
success on page 274	Core Tag Library. Allows you to supply code for success processing within a unit.
task on page 275	Core Tag Library. Identifies an XML task that you want to execute.
transform on page 297	Supplied Tag Library (XSL). Performs XSL transformation on an input group and streams the result to the task's output stream. Supports embedded **param** tags.
unit on page 278	Core Tag Library. Allows you to group a sequence of webjects, tasks, or Java code so that the group is executed as a unit.
updateObjects on page 288	Directory Tag Library. Updates existing entries in an LDAP directory service.
webject on page 281	Core Tag Library. Identifies the webject you want to execute.
when on page 294	Supplied Tag Library (JSTL). Similar to the **if** tag, but within the **choose** parent tag (represents the "if" of if/else logic). Provides JSTL support for Info*Engine tasks.

The following section describes some rules you need to be familiar with, and is followed by topics describing each tag in detail.

Attribute and Value Rules for Info*Engine JSP Pages and Tasks

In addition to the general JSP rules, a few additional rules concerning attributes and their data values must be followed for an Info*Engine task or JSP page to be well-formed and valid:

- All values for attributes must be enclosed in quotation marks. For example, the following are correct:

  ```
  <ie:param name="GROUP_OUT" data="employees"/>
  <ie:param name="GROUP_OUT" data='employees'/>
  ```

  However, the following is incorrect:

  ```
  <ie:param name="GROUP_OUT" data=employees/>
  ```

- The quotes must match on either side of the attribute value. Using the same example from above, the following is correct:

  ```
  <ie:param name="GROUP_OUT" data='employees'/>
  ```

  However, the following is incorrect:

  ```
  <ie:param name="GROUP_OUT" data="employees'/>
  ```

- If a particular quotation mark must appear within an attribute value, the quotation marks surrounding the entire value cannot be of the same type. For example, the following is correct:

  ```
  <ie:param name="HEADER" data="Dave's"/>
  ```

  However, the following is incorrect:

  ```
  <ie:param name="HEADER" data='dave's'/>
  ```

  Instead of using different quotation marks, you can use an escape character for the internal quotation mark. For example, the following is correct:

  ```
  <ie:param name="HEADER" data='dave\'s'/>
  ```

- Some special characters cannot appear directly in an attribute value for display as part of that value. These values can cause problems and should be used with caution. To include these characters as part of an attribute value, they must be encoded as follows, similar to special character encoding in HTML:

Symbol Name	Appearance	Character Encoding
less than	<	`&lt;`
greater than	>	`&gt;`
ampersand	&	`&amp;`
apostrophe	'	`&apos;`
double quote	"	`&quot;`

- Info*Engine maintains the @FORM, @SERVER, and @COOKIE context groups as part of the VDB. Therefore, you cannot use @FORM, @SERVER, and @COOKIE as names of any groups created in Info*Engine tasks. In addition, the **Auth-Map** context group is created as a result of executing an authentication task. Do not use this group name for anything other than the user authentication group. For additional information about the **Auth-Map** context group, see Credentials Mapping on page 125.

Note

The URIs shown in this guide use the forward slash as the separator (/) in file paths even though the back slash (\) is the directory separator used on NT systems. Info*Engine correctly identifies all system URIs when you specify the forward slash. If you prefer to use the back slash for NT URIs, you must escape the back slash in the URI. This means that you enter two \\ for each \ in the URI. For example URI locations, see Specifying URIs and URLs on page 68.

Core Library Tags

To use the Info*Engine tags, include the `taglib` directive, which identifies the tag library and provides a prefix for uniquely identifying the tags on the page. For example, to use the core Info*Engine tags and identify them by the `ie` prefix, include the following directive:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
```

The syntax for the tag assumes that you have specified the `ie` prefix in the `taglib` directive. If you specify a different prefix, use the prefix you specified in place of `ie` in the tag syntax.

displayResource

The **displayResource** tag extracts a string from a group created with the Get-Resource webject or directly from a resource bundle. The tag then replaces one variable place holder in the extracted string (if you supply replacement text) and displays the resulting string.

This tag is provided as an alternative to using the Display-Resource webject.

Note

Tags do not support multivalued attributes. Therefore, if an extracted string contains more than one variable place holder, you cannot use the **displayResource** tag to properly display the resulting string. Instead, you must use the Display-Resource webject.

Syntax

```
<ie:displayResource bundle="bundle_name"
                    groupIn="group_name"
                    key="bundle_key"
                    param="text"/>
```

Attribute Descriptions

Require Attributes: **bundle** or **groupIn**, and **key**.

bundle

Indicates the Java class resource bundle from which the localized string is extracted. This attribute is required if no **groupIn** attribute is supplied.

groupIn

Indicates the bundle group from which the message is to be extracted. This attribute is required if no **bundle** attribute is supplied.

key

Indicates the key into the resource bundle. This can be either the number or the actual Java variable reference name. This is a required attribute.

param

Specifies text to be inserted into a localized message that contains a variable place holder. For example, if the extracted string contains a variable place holder for text, such as:

```
The validation of user "{0}" has failed.
```

You can include the text that replaces `{0}` in the **param** attribute. For example, assume that you include the following attribute:

```
param="abc123"
```

Then, the resulting string becomes:

```
The validation of user "abc123" has failed.
```

This is an optional attribute. If the attribute is omitted, then the substitution does not occur.

If an extracted string contains more than one variable place holder, you must use the Display-Resource webject rather than this tag.

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The **displayResource** tag in the example specifies that the messages associated with line 19 of the resource bundle retrieved using the Get-Resource webject be displayed:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:webject name="Get-Resource" type="MGT">
  <ie:param name="BUNDLE"
                 data="com.infoengine.util.IEResource"/>
  <ie:param name="GROUP_OUT" data="IEResource"/>
</ie:webject>

<b>
<ie:displayResource groupIn="IEResource" key="19"/>
</b>
```

failure

The **failure** tag allows you to supply code for failure processing within a unit. The code between the start and end **failure** tag executes only when the body of the unit does not complete successfully. You can include multiple **failure** tag blocks in a unit:

- To include general failure processing, specify the **failure** tag with no attributes.
- To include code for processing a specific exception, specify the **failure** tag with the name of the exception in the **exception** attribute.

If you do not include a **failure** tag block for processing an error that occurs in the body of a unit, no failure processing occurs and the page continues to be processed following the **unit** end tag.

You nest this tag in **unit** tag blocks. To provide failure code sets for specific errors, you can specify multiple **failure** tag blocks. On each **failure** tag, you can name a specific error in the **exception** attribute. You can nest multiple **webject**, **task**, **unit**, and **parallel** tags in this tag block.

To manage exceptions within a **failure** tag block, you can include the Throw-Exception webject on page 436. Including this webject is a way to propagate (rethrow) exceptions that are caught by **failure** tags. You can simply add the following in the block:

```
<ie:webject name="Throw-Exception" type="MGT"/>
```

Including this webject causes the caught exception to be rethrown. Rethrowing exceptions is useful when **unit** blocks are nested or when you want exceptions caught in a **unit** block to be passed on to the page.

If an exception occurs in a **failure** tag block, the exception is propagated outside the **unit** block.

Scriptlets that are nested in **unit**, **init**, **success**, and **failure** tag blocks are not processed on JSP pages the same way they are processed in the Info*Engine task processor. Use the following guidelines to determine when you can nest scriptlets in these tags:

- You can nest scriptlets within tag blocks in a standalone task.
- You should not nest scriptlets in any tag block on a JSP page. Instead, create standalone tasks that contain the scriptlets. You can execute these tasks from the JSP page by using the Info*Engine **task** tag.

For additional information about scriptlets, see Scriptlets on page 246.

Note

Embedded HTML is not supported within **failure** tags in JSPs.

Syntax

```
<ie:failure exception="exception_name">
            .
            . (webject, task, unit, or parallel tag blocks)
            .
</ie:failure>
```

Attribute Descriptions

exception

Specifies a Java exception name for limiting the access to the failure processing code. When this attribute is included, the **failure** block is executed only when the exception named has occurred.

If you omit this attribute, the exception name is assumed to be "java.lang. Exception."

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The following **failure** tags provide general error processing for the unit and specific error processing for the `IENotFoundException` exception:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:unit>

     <ie:failure
 exception="com.infoengine.exception.fatal.IENotFoundException">
      <!-- specific failure processing for IENotFoundException-->
     </ie:failure>

     <ie:failure>
        <!--Add general failure processing here. -->
     </ie:failure>

</ie:unit>
```

forEach

The **forEach** tag block allows you to iterate through an existing Info*Engine group one element at a time. The result of each iteration produces one element (including all attributes of the element) in the output group specified. This resulting element is only available within the **forEach** tag block for which it was produced. You can then use this output group as the input group in Info*Engine tags that are nested in the block.

Each time the **forEach** end tag is reached, the processing loops back to the **forEach** start tag until there are no more elements in the input group. After all elements have been processed, the last element in the **forEach** input group is now in the output group (named in the **groupOut** attribute). Processing continues on to the next line after the **forEach** end tag.

Note

This tag cannot be nested in other Info*Engine tags, but other Info*Engine tags can be nested under this tag.

Syntax

```
<ie:forEach groupIn="group_name" groupOut="group_name">
            .
            . (other Info*Engine tag blocks)
            .
</ie:forEach>
```

Attribute Descriptions

Required Attributes: **groupIn** and **groupOut**.

groupIn

Specifies the name of the Info*Engine group to use as input. For each iteration of the loop, the next element (including all attribute values in the element) from the input group is moved to the output group.

This attribute is required.

groupOut

Specifies the name of the Info*Engine group to generate for each iteration.

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example assumes that the "employees" group exists as a result of a "CreateEmployeeGroup" task. The **forEach** tag block iterates through the "employees" group, one element at time. Nested within the block is the Display-Object webject, which displays the attributes with each element using the caption "One Employee":

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<!-- create input group -->
```

```
<ie:task uri="CreateEmployeesGroup"/>

<!-- iterate group, displaying the attributes for one employee -->
<!-- in each iteration -->
<ie:forEach groupIn="employees" groupOut="employee">
   <ie:webject name="Display-Object" type="DSP">
      <ie:param name="GROUP_IN" data="employee"/>
      <ie:param name="CAPTION" data="One Employee"/>
   </ie:webject>
</ie:forEach>
```

getService

The **getService** tag establishes a variable reference to the Info*Engine object (`com.infoengine.jsp.InfoEngine`) that is being used by the Info*Engine custom tags on a JSP page.

After you define the variable, you can use it in code that uses methods from the Info*Engine Server Access Kit (SAK). For example, you can access VDB information, retrieve groups, format rows and columns, and so on.

This tag also establishes implicit variable references for the context groups in use by tags within the page (`com.infoengine.object.factory.Group`). The variables instantiated are:

formGroup
Contains attributes that are obtained from the CGI query specification data that is received with the URL used to access the template. It also contains any HTML form data that was received as the result of a web browser POST request. This is the same information stored in the FORM context group.

serverGroup
Contains attributes that are derived from the protocol used to communicate from the web browser to the web server. It can contain values such as **accept-language** or **auth-user**. Refer to the current web-browser-to-web-server protocol specification to find more information on the individual attributes found in this group. This is the same information stored in the SERVER context group.

cookieGroup
Contains one element that has an attribute for each cookie that is processed during the connection to the JSP page. This is the same information stored in the COOKIE context group.

authGroup
Contains attributes that provide a credentials map for the task in which the webject executes. The map contains authentication information used by adapters in establishing connections to back-end information systems. Each

element of a credentials map provides a username and associated credentials that are used in connecting to a specific back-end system. This is the same information stored in the **Auth-Map** context group.

Note

Only use this tag on JSP pages; do not use it in standalone Info*Engine tasks.

Each page can only have one **getService** tag on it.

This tag cannot be embedded within other Info*Engine tags.

Syntax

`<ie:getService varName="variable_name"/>`

Attribute Descriptions

Required Attributes: **varName**

varName

Specifies a Java variable name for the Info*Engine object. This tag defines the variable; you do not need to define it before specifying the name here.

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example creates the “EMPLOYEEdata” group through a query and then displays the number of rows in the group. To get the number of rows, the example uses the **getService** tag to identify the service as “ieObj” and then uses the **getElementCount** method from `com.infoengine.object.factory.Group` class to retrieve the number of rows:

```
<%@page language="java" session="true" errorPage="../IEError.jsp"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
<html>
<body>

<!-- perform a query -->
<ie:webject name="Query-Objects" type="OBJ">
   <ie:param name="INSTANCE" data="com.myCompany.jdbcAdapter"/>
   <ie:param name="CLASS" data="EMP"/>
   <ie:param name="WHERE" data="()"/>
   <ie:param name="GROUP_OUT" data="EMPLOYEEdata"/>
</ie:webject>

<!-- display how many elements were returned -->
<ie:getService varName="ieObj"/>
   <P>Search returned
     <b><%=ieObj.getElementCount()%> employees. employees.>
</body>
</html>
```

getValue

The **getValue** tag retrieves the string value of the specified attribute from the first element in the input group.

This tag can be nested in other Info*Engine tags.

Do not imbed this tag in scriptlets; doing so causes a compiler error because the tag is read as plain text. For example, the following scriptlet does not compile:

```
<%
float total = 0;
%>

<ie:forEach ...>
<% total += <ie:getValue name="SAL"/>;%>
</ie:forEach>
```

Instead, within a scriptlet you can use the **getAttributeValue** method. For an example that uses this method, see the “Examples” section below.

Syntax

`<ie:getValue name="attr_name" groupIn="group_name"/>`

Attribute Descriptions

Required Attributes: **name**

groupIn

Specifies the name of the Info*Engine group to use as the input group.

This attribute is optional. If you omit this attribute, the last group added to the VDB is used.

name

Specifies the name of the attribute whose value you want to retrieve from the first element in the Info*Engine input group.

This attribute is required.

Examples

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example assumes that the "employees" group exists as a result of a "CreateEmployeesGroup" task. The **getValue** tags are nested within **table** HTML tags, producing values for elements in the table rows that are displayed.

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<!-- create input group -->
<ie:task uri="CreateEmployeesGroup"/>

<!-- iterate group, displaying the salary for one employee -->
<!-- in each iteration -->
<table>
<tr><td>Employee Name</td><td>Salary</td></tr>
<ie:forEach groupIn="employees" groupOut="employee">
   <tr>
      <td><ie:getValue name="ENAME"/></td>
      <td>$<ie:getValue name="SAL"/></td>
   </tr>
</ie:forEach>
</table>
```

The following example page uses the **getValue** tags to display selected attributes in each element and uses a scriptlet to calculate a running salary total. Computing the salary total uses the Info*Engine **getAttributeValue** method:

```
<%@page language="java" session="false" errorPage="../IEError.jsp"

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<!-- create a group that contains employee name, number, and salary -->
<ie:task uri="CreateEmployeesGroup"/>

<html>
<body>
<ie:getService varName="pie"/>
<% float tot_sal = 0; %>

<!-- iterate group, displaying the salary for one employee -->
<!-- in each iteration and calculating the total salary -->
<ie:forEach groupIn="employees" groupOut="one-element">
   <b>Employee Number</b>:<ie:getValue name="EMPNO"/><br>
   <b>employee name:</b><ie:getValue name="ENAME"/><br>
   <b>salary:</b>$<ie:getValue name="SAL"/><br>
   <hr><br>
   <%
      String ssal = pie.getAttributeValue ( "one-element", 0, "SAL" );
      tot_sal += Float.parseFloat ( ( (ssal != null && !ssal.equals(""))
? ssal :
"0" ) );
   %>
</ie:forEach>
<!-- Display the salary total for all employee in the group -->
<b>Salary Total:</b>$<%=tot_sal%><br>

</html>
</body>
```

init

The **init** tag supplies the initialization for a unit. The code between the start and end **init** tags executes first in a unit each time the unit is executed. Using an **init** code block allows you to identify specific code that always executes first when the unit is executed, even if the code block is not at the beginning of the unit. For example, in the initialization code, you could save the initial state of any objects that are manipulated in the unit. Then, if an error occurs in the body of the unit, the **failure** block could restore the objects to this initial state.

If a failure occurs in an **init** tag block, the error is returned to the code block from which the unit was executed. For example, assume that a JSP page has one unit nested in another unit as follows:

```
<!-- Top of page -->
:
:
<!-- First Unit -->
<ie:unit>
   <ie:init>
     <!-- Initialization of First Unit -->
     <!-- Errors occurring here are passed back to the page -->
   </ie:init>

<!-- Body of First Unit -->
   <ie:webject > ... </ie:webject>
   <ie:webject > ... </ie:webject>

   <!-- Nested Unit -->
   <ie:unit>
      <ie:init>
   <!-- Initialization of Nested Unit -->
```

```
      <!-- Errors occurring here are passed back to first unit -->
         </ie:init>

      <!-- Body of Nested Unit -->
         <ie:webject > … </ie:webject>
         <ie:webject > … </ie:webject>

         <ie:failure>
         <!-- Nested Unit failure processing accessed when -->
         <!-- there is an error in the body of the nested unit -->
         </ie:failure>

      <!-- End of Nested Unit -->
      </ie:unit>

      <ie:failure>
         <!-- First Unit failure processing accessed when -->
         <!-- there is an error in the body of the first unit-->
      </ie:failure>
   <!-- End of First Unit -->
   </ie:unit>

   <!-- Page Error Processing-->
   :
   :
   <!-- End of Page-->
```

If the initialization of the nested unit fails, the failure is recorded as an error in the body of the first unit. If the initialization of the first unit fails, this error is processed by the page, which may automatically send it to its error page.

Scriptlets that are nested in **unit**, **init**, **success**, and **failure** tag blocks are not processed on JSP pages the same way they are processed in the Info*Engine task processor. Use the following guidelines to determine when you can nest scriptlets in these tags:

- You can nest scriptlets within tag blocks in a standalone task.
- You should not nest scriptlets in any tag block on a JSP page. Instead, create standalone tasks that contain the scriptlets. You can execute these tasks from the JSP page by using the Info*Engine **task** tag.

For more information, see Scriptlets on page 246.

Note

Embedded HTML is not supported within **init** tags in JSPs.

You nest this tag in **unit** tag blocks.

You can nest multiple **webject**, **task**, **unit**, and **parallel** tags in this tag block.

Syntax

```
<ie:init>
            .
            . (webject, task, unit, or parallel tag blocks)
            .
</ie:init>
```

parallel

The **parallel** tag allows you to define a set of webjects or tasks that are executed concurrently.

Each webject and each task that is nested in a **parallel** tag block is executed in its own environment at the same time as the other webjects and tasks in the block. After all of the webjects and tasks successfully complete, then the VDBs of the individual processes are merged with the VDB in use by the page or task. Processing continues starting with the line after the **parallel** end tag.

If an exception occurs within the **parallel** block, by default the exception is propagated outside the **parallel** block after the parent VDB has been updated. If a webject or task within a **parallel** tag block fails, you should assume that the content of the resulting VDB is unpredictable.

You can nest this tag in **unit**, **init**, **success**, or **failure** tag bocks. You can also nest multiple **webject** and **task** tags in this tag block.

Note

Display webjects cannot be nested in a **parallel** tag bock.

Syntax

```
<ie:parallel>
            .
            . (webject or task tag blocks)
            .
```

```
</ie:parallel>
```

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The following **parallel** tag block executes two Query-Object webjects and a task at the same time:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:parallel>

   <ie:webject  name="Query-Objects" type="OBJ">
      <ie:param name="INSTANCE"      data="jndiAdapter"/>
      <ie:param name="FILTER"        data="(objectClass=*)"/>
      <ie:param name="GROUP_OUT"     data="dirOut"/>
   </ie:webject>

   <ie:webject  name="Query-Objects" type="OBJ">
      <ie:param name="INSTANCE"      data="jdbcAdapter"/>
      <ie:param name="CLASS"         data="EMP"/>
      <ie:param name="WHERE"         data="()"/>
      <ie:param name="GROUP_OUT"   data="dbOut"/>
   </ie:webject>

   <ie:task URI="stask.xml">
      <ie:param name="P1" data="v1"/>
      <ie:param name="P2" data="v2"/>
      <ie:param name="GROUP_OUT" data="people"/>
      <ie:param name="GROUP_OUT" data="employees"/>
   </ie:task>

</ie:parallel>
```

If this **parallel** tag block is part of a JSP page, all four of the groups created in the block are available to any display webjects that follow the block. If the block is in a task, you would use the Return-Groups webject after the block to make the groups available to display webjects.

param

The **param** tag defines a parameter for use within **webject** tags, **task** tags, and directory service tags:

- Webject parameters provide the input criteria for the webject in which they are specified.
- Task parameters provide a way to set the following items:

    - @FORM context group variables so that they are available to the task.
    - The VDB groups that are available to the task.
    - The VDB groups that are available when the task finishes.
- Directory service parameters provide the names of groups that are used as input to and output from creating, updating, querying, and listing directory service entries.

In each case, the parameter requirements are dictated by the individual webjects, tasks, and directory service actions where they are specified. For additional information, see the description of the specific webject, task, or directory service action you want to accomplish.

You can include expressions as attribute values on a **param** tag. For more information, see Expressions on page 247.

You can nest this tag in the **webject**, **task**, **createObjects**, **listObjects**, **queryObjects**, and **updateObjects** tags.

Syntax

```
<ie:param name="PARAMETER_NAME"
       data="value_list"
       delim="delimiter"
       default="default_value"
       elementSeparator="character"
       valueSeparator="character"/>
```

Attribute Descriptions

Required attributes: **name** and **data**.

data

Specifies one or more data values to assign to the parameter named in the **name** attribute. Include the data values within quotation marks. The number and type of data values that you can assign are determined by the specific parameter.

For parameters that can have multiple data values assigned to them, the most common way to enter multiple data values is by including multiple **param** tags, all with the same **name** attribute and different **data** attributes.

You can also include multiple values in the **data** attribute by separating the values using a delimiter and including the separator in **delim** attribute. To use the comma as a value separator, use the following syntax:

```
data="value1,value2,...,valueN" delim=","
```

For example, to return multiple groups in the Return-Groups webject, you can include either the following two **param** tags:

```
<ie:param name="GROUP_IN" data="employees"/>
```

```
<ie:param name="GROUP_IN" data="consultants"/>
```

or one **param** tag that includes the **delim** attribute:

```
<ie:param name="GROUP_IN" data="employees,consultants" delim=","/>
```

The **data** attribute is required.

default

Specifies a literal string, which is the default value that is used if a substitution expression returns no values. When you specify this attribute, you must include the default value within quotation marks.

Whenever you include substitution expressions in the **data** attribute, you should include a default for the expression. For example, the following **param** tag sets the default for the ATTRIBUTE parameter to the string “*”:

```
<ie:param name="ATTRIBUTE" data="$(@FORM[]attr[])" default="*"/>
```

This attribute is optional.

delim

Defines the delimiting symbol that Info*Engine uses to separate multiple values in the **data** attribute. When you specify this attribute, you must include the symbol within quotation marks. For example, if you want to use the comma as the delimiter, the syntax for the **data** and **delim** attributes is:

`data="`*`value1,value2,...,valueN`*`" delim=","`

The following **param** tag example has three values defined in the **data** attribute and uses the comma as the delimiter:

```
<ie:param name="ATTRIBUTE" data="ename,phone,title" delim=",">
```

By using multiple **param** tags that have only one value in each **data** attribute, this same parameter could also have been specified as follows:

```
<ie:param name="ATTRIBUTE" data="ename">
<ie:param name="ATTRIBUTE" data="phone">
<ie:param name="ATTRIBUTE" data="title">
```

In an actual webject, the resulting parameter in both cases tells the processor that the webject expects to receive employee name (ename), employee telephone number (phone), and employee title (title) information within the group of data returned from the data repository.

This attribute is optional.

elementSeparator

Specifies the element separator that Info*Engine uses when processing substitution expressions that are in the **data** attribute. The specified character is used when concatenating together attribute values from multiple elements (rows). The selection of multiple elements can occur when the expression contains the asterisk (*) as the element selector.

By default, Info*Engine uses the semicolon as the element separator. For more information about defining substitution expressions, see Dynamic Parameter Value Substitution on page 44.

The following **param** tag includes a substitution expression in the **data** attribute that has the asterisk as the element selector and sets the element separator to “#”:

```
<ie:param name="ATTRIBUTE" data="$(grp1[*]attr[0]}" elementSeparator="#"/>
```

This attribute is optional.

name

Specifies a parameter name to which a data value is assigned. Include the parameter name within quotation marks. The names of the parameters are not case-sensitive, but are documented using upper case characters.

This attribute is required.

valueSeparator

Specifies the value separator that Info*Engine uses when processing substitution expressions that are in the **data** attribute. The specified character is used when concatenating together multiple values that are contained in one attribute location. The location is defined by an attribute (column) and an element (row) pair that results from processing substitution expressions. Multiple values can occur when the expression contains the asterisk (*) as the value selector.

By default, Info*Engine uses the comma as the value separator. For more information about defining substitution expressions, see Dynamic Parameter Value Substitution on page 44.

The following **param** tag includes a substitution expression in the **data** attribute that has the asterisk as the value selector and sets the value separator to “|”:

```
<ie:param name="ATTRIBUTE" data="$(grp1[0]attr[*])" valueSeparator="|"/>
```

This attribute is optional.

Example

The following example declares that the page or task uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The **param** tags define two parameters for the Copy-Group webject:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:webject name="Copy-Group" type="GRP">
   <ie:param name="GROUP_IN" data="grp1"/>
   <ie:param name="GROUP_OUT" data="grp2"/>
</ie:webject>
```

resetService

The **resetService** tag resets the Info*Engine object (`com.infoengine.jsp.InfoEngine`) that is being used by the Info*Engine custom tags on a JSP page or in a session. If you include a `page` scope, the reset object exists only while the page executes. If you specify a `session` scope, the reset object is available to all pages in the session.

When you reset an Info*Engine object using a `page` scope, the following things happen:

- All VDB groups that were available to the page are no longer available in the new object.
- All VDB groups created after the object is reset are only available to the page (even if the `page` directive for the page includes `session=TRUE`).

You can copy a group from a session VDB to a page VDB or from a page VDB to a session VDB using the **addGroup** method from the `com.infoengine.object.factory.Group` class.

When you reset an Info*Engine object using a session scope, all VDB groups that were available to the session are no longer available in the new object.

Unless you explicitly save an existing Info*Engine object, the existing object is no longer available when you reset the object using the **resetService** tag.

If you supply a new variable name for the reset object, the new service object can be referenced by the name in code that uses methods from the Info*Engine Server Access Kit (SAK). For example, you can access VDB information, retrieve groups, format rows and columns, and so on.

The **resetService** tag can be very useful in the following situations:

- Avoiding conflicts when accessing groups in the VDB. You may need to do this when an application that consists of multiple JSP pages has session scope and there can be multiple requests for VDB groups occurring at the same time. Ensuring that each request gets back the intended groups may not be possible unless you reset the Info*Engine object for each page. This allows you to restrict the groups available in the VDB to those created on the page while setting other information to a `session` scope.
- Cleaning up the VDB. When your application has created many VDB groups that are no longer needed by the application, you can reset the VDB to free up resources.

Note

Only use this tag on JSP pages; do not use it in standalone Info*Engine tasks.

You can include multiple **resetService** tags on the same page as long as you either omit the variable name or specify unique object names. You cannot specify the same variable name on multiple **resetService** tags, or conversely on a **getService** tag and a **resetService** tag.

This tag cannot be embedded within other Info*Engine tags.

Syntax

```
<ie:resetService varName="variable_name"
     scope="[PAGE | SESSION]"/>
```

Attribute Descriptions

varName

Specifies a Java variable name for the Info*Engine object. This tag defines the variable; do not define it before specifying the name here. The variable name cannot be the same name specified in either the **getService** tag or other associated **resetService** tags.

This attribute is optional. If omitted, the object cannot be referenced through SAK methods.

scope

Specifies where the Info*Engine object is stored. The valid values for scope are:

`PAGE`—Sets the object scope to the page in which the tag is used.

`SESSION`—Sets the object scope to the session in which the tag is used.

This attribute is optional. If omitted, the value defaults to the location of the Info*Engine object in use. If `session=true` on the `page` directive, then the session Info*Engine object is reset. Otherwise, the page Info*Engine object is reset.

Example

The following example declares that the page has a session scope and uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example expands on the **getService** tag example which gets the number of rows in the “EMPLOYEEdata”. The example resets the service object for the page using the **resetService** tag and continues on with other queries:

```
<%@page language="java" session="true" errorPage="../IEError.jsp"%>
```

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>
<html>
<body>
<!-- perform a query -->
<ie:webject name="Query-Objects" type="OBJ">
   <ie:param name="INSTANCE" data="jdbcAdapter"/>
   <ie:param name="CLASS" data="EMP"/>
   <ie:param name="WHERE" data="()"/>
   <ie:param name="GROUP_OUT" data="EMPLOYEEdata"/>
</ie:webject>

<!-- display how many elements were returned -->
<ie:getService varName="ieObj"/>
   <P>Search returned
     <b><%=ieObj.getElementCount()%></b> employees.</P>

<!-- reset the VDB for the page -->
<ie:resetService varName="ieObjII" scope="PAGE"/>
<!-- perform additional queries -->
   .
   .
   .
</body>
</html>
```

Note

Because the **resetService** tag only resets the service for remainder of the page, at the end of the page processing, the session VDB still includes the "EMPLOYEEdata" group.

success

The **success** tag allows you to supply code for success processing within a unit. The code between the start and end **success** tag executes only when the body of the unit completes successfully.

If a failure occurs in this block, the exception is propagated outside the enclosing **unit** block.

Nest this tag in **unit** tag blocks. You can also nest multiple **webject**, **task**, **unit**, and **parallel** tags in this tag block.

Scriptlets that are nested in **unit**, **init**, **success**, and **failure** tag blocks are not processed on JSP pages the same way they are processed in the Info*Engine task processor. Use the following guidelines to determine when you can nest scriptlets in these tags:

- You can nest scriptlets within tag blocks in a standalone task.
- You should not nest scriptlets in any tag block on a JSP page. Instead, create standalone tasks that contain the scriptlets. You can execute these tasks from the JSP page by using the Info*Engine **task** tag.

For more information, see Scriptlets on page 246.

Note

Embedded HTML is not supported within **success** tags in JSPs.

Syntax

```
<ie:success>
            .
            . (webject, task, unit, or parallel tag blocks)
            .
</ie:success>
```

Example

See the **unit** tag example on page 280.

task

The **task** tag identifies an XML task that you want to execute.

You can nest this tag in **unit**, **init**, **parallel**, **success**, and **failure** tag blocks.

You can specify the parameters for a task by nesting the **param** tag within this tag block. Task parameters provide a way to set the following items:

- The VDB groups that are available to the task. Specifying GROUP_IN task parameters allows you to define which groups are initially available to the task.
- The VDB groups that are available when the task finishes. Specifying GROUP_OUT task parameters allows you to filter the groups that are returned to the VDB of the task in which the nested task is executed.

  By default, the GROUP_OUT parameter on the last webject executed identifies the groups that are returned. If there is no GROUP_OUT webject parameter, then the last group added to the VDB is returned.

When multiple groups are returned through the last webject (which can be the case with the Return-Groups webject), specifying a subset of these groups in GROUP_OUT task parameter limits the groups that are returned to the VDB of the calling task.

- @FORM context group variables so that they are available to the task. An @FORM context group variable is set for each parameter name and data pair that you specify in a **param** tag nested in a **task** tag block. This includes any GROUP_IN and GROUP_OUT parameters specified for the task.

  Because the @FORM group GROUP_IN variables contain the names of the VDB groups that are initially available in the task, you can get the VDB group names by reading the values from the @FORM group GROUP_IN variables.

Note

Using this tag with the Info*Engine task processor `.secret.text` or `.secret.text2` and `.secret.algorithm` properties allows for validation to occur before a remote processor executes the task. For information about configuring these properties, see the Info*Engine Implementation Guide.

Syntax

When there are parameters, you can use the following syntax:

```
<ie:task uri="uri_task_source"
     processor="processor1"
     processor="processor2"
            .
            .
            .
     processor="processorn"
     resumable="[true|false]">
```

  .
  . (Nest task parameters using **param** tags)
  .

```
</ie:task>
```

When there are no parameters, you can use the following syntax:

```
<ie:task uri="uri_task_source"
     processor="processor1"
     processor="processor2"
     .
     .
     .
```

```
      processor="processorn"
      resumable="[true|false]"/>
```

Attribute Descriptions

Required attribute: **uri**

processor

Specifies one or more names of remote Info*Engine task processors to which the task can be sent for execution. Each name you specify must map to a task processor that is available from your current environment. The names you can specify in this attribute are those service names defined for task processors through the Info*Engine Property Administration utility.

This attribute is optional. When it is not specified, the task named in the **task** tag is executed by the task processor that is currently executing the JSP page or task that contains the **task** tag. For JSP pages, the task processor used by default is running in the JSP engine. To direct the nested task to execute in the Info*Engine Server task processor (rather than in the JSP engine), you must include the **processor** attribute that identifies the server task processor. For example, if the task processor has the default name of "com.myCompany. server.taskProcessor" you can include the following **processor** attribute:

```
processor="com.myCompany.server.taskProcessor"
```

If your current environment has other task processors set up for your use, you can direct the task to choose one of those task processors by specifying the processor names on **processor** attributes in the order you want them selected. For example, assume that your site has set up "xxx.taskProcessor" and "yyy. taskProcesor" for your use, and that you would prefer running the task on "yyy.taskProcessor". To accomplish this, include the following **processor** attributes in the **task** tag:

```
   <ie:task uri="task1.xml" processor="yyy.taskProcessor" processor=
"xxx.taskProcessor" processor="zzz.taskProcessor" />
```

Notice that the third processor named is the "zzz.taskProcessor." It is only used if both of the other processors are not available.

resumable

Indicates whether the VDB state must be saved before running the subtask named in this **task** tag. This attribute is only used when guaranteed task execution has been enabled and the **task** tag is executed from a task.

If you omit the attribute (or set it to `false`) and guaranteed task execution is enabled, then Info*Engine saves the state of the VDB before running the task. When this is done, the VDB can be restored in the case where the task must be rerun.

If you set the attribute to “true” and guaranteed task execution is enabled, then Info*Engine does not save the state of the VDB before running the task. Set **resumable** to `true` when the task can be run without causing side effects that would prevent the task from producing the same results if it were run again.

uri

Specifies a URI that is the location of the XML task file to execute. The URI can be a relative or absolute URI:

- Relative URIs reference files that reside under the root file system directory that is defined for the local Info*Engine task processor.
- Absolute URIs reference files that reside in the local file system, reside on a remote HTTP server, or are referenced through an accessible LDAP directory.

This attribute is required.

Example

The following example declares that the page or task uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example **task** tag executes the `task1.xml` file that is in the root file directory defined for the local task processor. The nested parameter sets the FORM variable **attr** to “ENAME”:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:task uri="task1.xml">
   <ie:param name="attr" data="ENAME"/>
</ie:task>
```

unit

The **unit** tag allows you to group a sequence of webjects, tasks, or Java code so that the group is executed as a unit. Within the unit, you can supply the main body and can also define the following nested tags to explicitly code special parts of the unit:

- The **init** tag block supplies the initialization for the unit. The code between the start and end **init** tag executes first in the unit.
- The **success** tag block provides a way to specify code that executes only when the code in the body of the unit has completed successfully.
- The **failure** tag block provides a way to specify code that executes only when the code in the body of the unit has failed. You can include multiple **failure** tag blocks in which you can specify error processing for specific errors.

The main body of a unit consists of all webjects, tasks, and Java code within the **unit** start and end tags, but outside of the nested **init**, **success**, and **failure** tag blocks.

The placement of the **init**, **success**, and **failure** tag blocks in the unit have no significance. When the unit executes, the code in the **init** tag block executes first regardless of where it is in the unit. If it completes successfully, then the code in the body of the unit executes in the order it is presented in the code, starting at the beginning of the unit. If the body completes successfully, then the code in the **success** tag block executes. If any code in the body fails, then the **failure** tags in the unit are checked to determine if processing for that error has been provided. If you provide error processing for the error that has occurred, then that code executes.

If an exception occurs in a **success** block, the exception is thrown and the unit is not processed. If an exception occurs in the body of the unit, then it is processed by a **failure** tag (if one is defined for the exception) or by a general **failure** tag. If an error in the body of a unit is not caught through a **failure** tag, the error is not handled.

You can nest this tag in other **unit** tag blocks and in **init**, **success**, and **failure** tag blocks.

You can nest one **init** tag block, one **success** tag block, and one or more **failure** tag blocks in this tag block. You can also nest multiple **webject**, **task**, **unit**, and **parallel** tags in this tag block.

For additional information about the **init**, **success**, and **failure** tags, see the section that corresponds to each tag.

Scriptlets that are nested in **unit**, **init**, **success**, and **failure** tag blocks are not processed on JSP pages the same way they are processed in the Info*Engine task processor. Use the following guidelines to determine when you can nest scriptlets in these tags:

- You can nest scriptlets within tag blocks in a standalone task.
- You should not nest scriptlets in any tag block on a JSP page. Instead, create standalone tasks that contain the scriptlets. You can execute these tasks from the JSP page by using the Info*Engine **task** tag.

For additional information about scriptlets, see Scriptlets on page 246.

Note

Embedded HTML is not supported within **unit** tags in JSPs.

Syntax

```
<ie:unit>
```

.
. (**webject**, **task**, **init**, **unit**, **parallel**, **success**, and **failure** tag blocks.)
.
`</ie:unit>`

Example

The following example declares that the page uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example **unit** tag groups the main body containing the Query-Objects webject with a **success** and **failure** block. The Query-Objects webject executes and, if it completes successfully, then the **success** block executes. If it does not complete successfully, then the **failure** block executes:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:unit>

   <ie:webject name="Query-Objects" type="OBJ">
      <ie:param name="INSTANCE" data="adapter"/>
      <ie:param name="CLASS" data="salesemp"/>
      <ie:param name="WHERE" data="()"/>
      <ie:param name="GROUP_OUT" data="sales">
   </ie:webject>

<ie:success>
  <ie:webject name="Display-Table" type="DSP">
     <ie:param name="GROUP_IN" data="sales"/>
    <ie:param name="ATTRIBUTE" data="ename,phone,title" DELIM=","/>
    <ie:param name="HEADER" data="Name,Telephone,Title" DELIM=","/>
  </ie:webject>
</ie:success>

<ie:failure>
  <ie:webject name="Create-Group" type="GRP">
     <ie:param name="ELEMENT" data="FAILURE=query failed"/>
     <ie:param name="GROUP_OUT" data="failure"/>
  </ie:webject>
  <ie:webject name="Display-Table" type="DSP">
     <ie:param name="GROUP_IN" data="failure"/>
     <ie:param name="ATTRIBUTE" data="FAILURE"/>
  </ie:webject>
</ie:failure>

</ie:unit>
```

webject

The **webject** tag identifies the webject you want to execute.

On JSP pages, you can name any defined webject in the **webject** tag.

You can nest this tag in **unit**, **init**, **success**, **failure**, and **parallel** tag blocks.

For webjects that require parameters, you specify the parameters by nesting the **param** tag in this tag block.

Note

In standalone tasks, display webjects are not allowed.

Display webjects are not allowed to be embedded within **parallel** tag blocks.

Syntax

When there are parameters, you can use the following syntax:

```
<ie:webject name="Webject-Name"
      type="TYPE_CONSTANT"
      resumable="[true|false]"
      use="webject_class_path">
  .
  . (Nest webject parameters using param tags)
  .
</ie:webject>
```

When there are no parameters, you can use the following syntax:

```
<ie:webject name="Webject-Name"
      type="TYPE_CONSTANT"
      resumable="[true|false]"
      use="webject_class_path"/>
```

Attribute Descriptions

Required attributes: **name** and **type**

name

Specifies an Info*Engine webject name. This attribute is required.

resumable

Indicates whether the VDB state must be saved before running the webject named in this **webject** tag. This attribute is only used when guaranteed task execution has been enabled and the webject is an action webject (TYPE= ACT) that resides in a task.

If you omit the attribute (or set it to `false`) and guaranteed task execution is enabled, then Info*Engine saves the state of the VDB before running the action webject. When this is done, the VDB can be restored in the case where the webject must be rerun.

If you set the attribute to `true` and guaranteed task execution is enabled, then Info*Engine does not save the state of the VDB before running the webject in a task. Set **resumable** to `true` when the webject can be run without causing side effects that would prevent the webject from producing the same results if it were run again.

type

Indicates the type of webject you want to use. The type determines which package is searched for the webject class file. Info*Engine type constants are defined for the following types of webjects:

Type Constant	Webject Type
DSP	Display
IMG	Image
ACT	Action
OBJ	Query
GRP	Group
EXT	External
MGT	Management
MSG	Messaging
WES	Web Event Service

This attribute is required.

Note

Display webjects (type=DSP) cannot be used in Info*Engine tasks.

The Messaging (type=MSG) and Web Event Service (type=WES) webjects were developed for use in Info*Engine tasks. These types of webjects should not be used directly within a JSP page, although no exception is thrown. The webjects should be called from a JSP page using the task tag with a **processor** attribute.

use

Specifies the path to the class containing the external webject when the webject named is not a class within.

This attribute is required when the **type** attribute is set to EXT.

Example

The following example declares that the page or task uses tags from the Info*Engine core tag library and that the tags have the `ie` prefix. The example **webject** tag names the Copy-Group webject, which is a group webject that has two parameters:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<ie:webject name="Copy-Group" type="GRP">
   <ie:param name="GROUP_IN" data="grp1"/>
   <ie:param name="GROUP_OUT" data="grp2"/>
</ie:webject>
```

Directory Library Tags

To use tags in the directory tag library, you must include a `taglib` directive similar to the following:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>
```

The syntax for these tags assumes that you have specified the `iedir` prefix in the `taglib` directive. If you specify a different prefix, use the prefix you specified in place of `iedir` in the tag syntax.

Note

The directory library tags can only be used in JSP pages; they are not recognized in standalone tasks.

createObjects

The **createObjects** tag creates new entries in the specified LDAP directory service.

Before executing this tag, you must create an Info*Engine group that contains elements that define the LDAP entries. Each element must have the following attributes:

- **dn**—The distinguished name of the entry.
- **objectClass**—The object class to associate with the entry. Determines valid values from the directory schema configured for the LDAP server.

You can include additional attributes in each element, if they are required by the object class.

You specify the required Info*Engine input group by nesting the **param** tag in this tag block and providing the GROUP_IN parameter value. For more information, see the param on page 270 tag.

You also specify the required parameter by nesting the **param** tag in this tag block.

Syntax

```
<iedir:createObjects uri="service_URL">
   <iedir:param name="GROUP_IN" data="group_name"/>
</iedir:createObjects>
```

Attribute Descriptions

Required Attributes: **uri**

uri

Specifies an LDAP URL that identifies the directory service to use when creating the new entries. For example, entering the following URL identifies the host and the base entry as “myHost.myState.myCompany.com”:

```
ldap://myHost.myState.myCompany.com
```

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine directory and core tag libraries and that the tags have the `iedir` and `ie` prefixes. The **task** tag block creates the group of elements used as input and the **createObjects** tag block creates LDAP entries in the “myHost.myCompany.com” directory service:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>

<ie:task uri="ldaptask.xml">
   <ie:param name="GROUP_OUT" data="ldap_group"/>
</ie:task>

<iedir:createObjects
       uri="ldap://myHost.myCompany.com">
   <iedir:param name="GROUP_IN" data="ldap_group"/>
</iedir:createObjects>
```

deleteObject

The **deleteObject** tag deletes one LDAP leaf entry from the LDAP directory service.

Syntax

`<iedir:deleteObject uri="entry_URL"/>`

Attribute Descriptions

Required Attributes: **uri**

uri

Specifies an LDAP URL that identifies one LDAP leaf entry to delete.

The format of this LDAP URL is:

`ldap://`*`hostname:port/search_base`*

Replace each part of the URL with the appropriate value:

- *`hostname:port`* locates the LDAP directory.
- *`search_base`* is the distinguished name identifying the leaf entry to delete.

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine directory tag library and that the tags have the `iedir` prefix. The **deleteObject** tag deletes the "com.mycompany.myHost.jAdpater" entry from the directory service located at "myCompany.com":

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>
<iedir:deleteObject
uri="ldap://myCompany.com/dc=jAdapter,dc=myHost,dc=myCompany,dc=com"/>
```

listObjects

The **listObjects** tag creates an Info*Engine group containing elements that are the relative distinguished names of the LDAP directory entries located directly under the specified base directory entry. Each element in the group consists of one attribute which is the distinguished name relative to the base entry.

If there are no entries directly under the specified base entry, the group returned is empty.

You name the Info*Engine group that is created using the GROUP_OUT parameter on a nested **param** tag. For more information, see the param on page 270 tag.

You also specify the required parameter by nesting the **param** tag in this tag block.

Syntax

```
<iedir:listObjects uri="base_level_URL">
   <iedir:param name="GROUP_OUT" data="group_name"/>
</iedir:listObjects>
```

Attribute Descriptions

Required Attributes: **uri**

uri

Specifies an LDAP URL that identifies the base directory entry to use when searching for entries. For example, entering the following URL identifies the “myCompany.com” host sets the base entry at “myHost.myState.myCompany.com”:

```
ldap://myCompany.com/dc=myHost,dc=myState,dc=
myCompany,dc=com
```

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine directory tag library and that the tags have the `iedir` prefix. The **listObjects** tag block creates the “ldap_myHost” group from entries in the directory service located at “myCompany.com” under the “dc=myHost,dc=myCompany,dc=com” base entry:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>

<iedir:listObjects
       uri="ldap://myCompany.com/dc=myHost,dc=myCompany,dc=com">

   <iedir:param name="GROUP_OUT" data="ldap_myHost"/>
</iedir:listObjects>
```

queryObjects

The **queryObjects** tag creates an Info*Engine group containing elements that are the LDAP directory entries matching the search criteria specified. Each element in the group consists of the attributes found in each entry. The names and values of the attributes in each element correspond to the names and values of the LDAP directory entry attributes. At a minimum, every element has the following attributes:

Attribute	Description
dn	The distinguished name of the entry.
objectClass	The object class to associate with the entry.

If there are no entries found using the specified search criteria, the group returned is empty.

You name the Info*Engine group that is created using the GROUP_OUT parameter on a nested **param** tag. For more information, see the param on page 270 tag.

Syntax

```
<iedir:queryObjects uri="query_URL">
      <iedir:param name="GROUP_OUT" data="group_name"/>
</iedir:queryObjects>
```

Attribute Descriptions

Required Attributes: **uri**

uri

Specifies an LDAP URL that identifies the search criteria to use in locating LDAP directory entries.

For the general format of this LDAP URL and examples, see Specifying URIs and URLs on page 68.

This attribute is required.

Example

The following example declares that the page uses tags from the Info*Engine directory tag library and that the tags have the `iedir` prefix. The **queryObjects** tag block creates the "ldap_query" group that contains all LDAP entries that reside in the directory service located at "myCompany.com" under the "dc=myHost,dc=myCompany,dc=com" base entry and that have an **objectClass** attribute:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>

<iedir:queryObjects
       uri="ldap://myCompany.com/dc=myHost,dc=myCompany,dc=com??sub?objectClass=*">
   <iedir:param name="GROUP_OUT" data="ldap_query"/>
</iedir:queryObjects>
```

updateObjects

The **updateObjects** tag updates existing entries in an LDAP directory service. Using this tag, you can add new attributes to existing LDAP entries, replace existing attributes, or delete existing attributes. Name the Info*Engine group that contains the LDAP attributes that you are updating by using the GROUP_IN parameter on a nested **param** tag.

Before executing this tag, you must create an Info*Engine group that contains elements that define the LDAP entries to be modified. Each element must have the **dn** attribute, the distinguished name of the entry. The additional attributes in each element identify names and values of the LDAP attributes to modify.

Syntax

```
<iedir:updateObjects uri="service_URL" modification="type">
    <iedir:param name="GROUP_IN" data="group_name"/>
</iedir:updateObjects>
```

Attribute Descriptions

Required Attributes: **uri**

uri

Specifies an LDAP URL that identifies the directory service and the base entry to use when updating entries. For example, entering the following URL identifies the “myCompany.com” directory service:

```
ldap://myCompany.com/
```

This attribute is required.

modification

Specifies the type of modification to do. Enter one of the following types:

Modification Type	Description
`ADD`	Adds the LDAP attributes specified in the group element to the LDAP entry identified by the **dn** group element attribute. If an LDAP attribute that is specified in a group element already exists in the LDAP entry, the tag returns an error.
`DELETE`	Deletes the LDAP attributes other than **dn** that are named in group element attributes for the LDAP entry. Each LDAP entry is identified by the **dn** group element attribute. To delete an entire LDAP entry, use the **deleteObject** tag. If an LDAP attribute that is specified in a group element does not exist in the LDAP entry, the tag may return an error or may complete without an error, depending on the LDAP server in use.
`REPLACE`	Replaces the values of existing LDAP attributes that are specified in the group element. The LDAP entry where the replacement is done is identified by the **dn** group element attribute. If an LDAP attribute that is specified in the group element does not exist in the LDAP entry, the attribute and its corresponding value are added to the entry.

The default for this optional attribute is `REPLACE`.

Example

The following example declares that the page uses tags from the Info*Engine directory and core tag libraries and that the tags have the `iedir` and `ie` prefixes. The **task** tag block creates the group of elements used as input and the **updateObjects** tag block updates LDAP entries in the “myCompany.com” directory service at the “dc=myHost,dc=myCompany,dc=com” base entry:

```
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
           prefix="ie" %>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/directory"
           prefix="iedir" %>

<ie:task uri="ldaptask.xml">
   <ie:param name="GROUP_OUT" data="ldap_update"/>
```

```
</ie:task>
<iedir:updateObjects
      uri="ldap://myCompany.com/dc=myHost,dc=myCompany,dc=com">
   <iedir:param name="GROUP_IN" data="ldap_update"/>
</iedir:updateObjects>
```

Supplied Library Tags (JSTL)

This tag library provides basic JSTL-like functionality for Info*Engine tasks. This includes the ability to set variables, iterate over lists and use conditional logic.

These tags are included in the `/com/infoengine/tlds/iejstl.tld` tag library. To use these tags, you must include a `taglib` directive in your task similar to the following:

```
<%@ taglib uri="/com/infoengine/tlds/iejstl.tld"
           prefix="c" %>
```

The syntax for these tags assumes that you have specified the `c` prefix in the `taglib` directive. If you specify a different prefix, use the prefix you specified in place of `c` in the tag syntax.

choose

This tag allows for conditional evaluation with an alternative for evaluation failure (if/else). It requires embedded children for when/otherwise evaluations.

Syntax

```
<c:choose>
   <c:when test="CONDITION">
     <!-- processed when CONDITION evaluates to true -->
   </c:when>
   <c:otherwise>
     <!-- processed when CONDITION evaluates to false -->
   </c:otherwise>
</c:choose>
```

Example

```
<c:choose>
  <c:when test="${group == null  || group.elementCount==0}">
    <!-- do when group named "group" is empty -->
  </c:when>
  <c:otherwise>
    <!-- do when group contains elements -->
  </c:otherwise>
</c:choose>
```

forEach

This tag supports iteration over an iterable, group, or array of objects.

Syntax

```
<c:forEach var="VARIABLE" var="LIST">
   <!-- invoked once for each element in LIST -->
</c:forEach>
```

Attribute Descriptions

Required attributes: **list** and **var**.

list

The list to iterate over. This must evaluate to an iterable, group, or array.

This attribute is required, and is also an expression.

var

The variable name to use for individual list elements during iteration. The value of **var** specifies the name of an implicit variable that is defined (and which must be unique within its scope within your task). In addition, a tasklet context variable is set with this name for each iteration so that it can be reused within other expressions. If the list being iterated over is an Info*Engine group, then the context variable is an Info*Engine element.

In addition to setting the variable, an element group named with the **var** attribute value is placed in the VDB (similar to the **forEach** tag functionality supplied with the Info*Engine core tag library).

Multi-valued attributes can also be iterated over using an expression such as `${myGroup[0]att[*]}`. This attribute is required.

Example

```
<c:forEach var="one" list="${output}">
  <log:info message="attribute A has a value of ${one[0]A[0]}" />
</c:forEach>
```

if

The **if** tag allows for conditional evaluation of its body.

Syntax

```
<c:if test="CONDITIONAL">
   <!-- invoked if CONDITIONAL evaluates to true -->
</c:if>
```

Attribute Descriptions

Required attributes: **test**

test

Contains the conditional statement to be evaluated. The test attribute supports operators [==, !=, <, >, <=, and >=]. The numeric operators require that the operands be of a numeric data type for comparison. Where possible, if the operands are of different data types an attempt is made to coerce one to the other to attempt comparison. If one of the operands is a constant, then an attempt is made to coerce the variable to the constant. Conditional expressions of arbitrary complexity can be specified. Portions of expressions to be evaluated together must be grouped with ( ) and expressions can then be joined by [||, &&].

This attribute is required and is also an expression.

Note

&& must be encoded as '&amp;&amp;'.

Example

```
<c:if test="${@FORM[]bool[]==true}">
  <!-- conditional code -->
</c:if>

<c:if test="${this==that || (this!=that&amp;&amp;a<b)}">
  <!-- conditional code -->
</c:if>
```

otherwise

This tag is related to the **choose** and **when** tags. It must be a child of the **choose** tag, and have a sibling **when** tag. If the **when** tag evaluates to `false`, then the content of the **otherwise** tag is invoked and executed (represents the "else" of if/else logic).

Syntax

```
<c:choose>
    <c:when test="CONDITION">
      <!-- processed when CONDITION evaluates to true -->
    </c:when>
    <c:otherwise>
      <!-- processed when CONDITION evaluates to false -->
    </c:otherwise>
</c:choose>
```

Example

```
<c:choose>
  <c:when test="${group == null  || group.elementCount==0}">
    <!-- do when group named "group" is empty -->
  </c:when>
  <c:otherwise>
    <!-- do when group contains elements -->
  </c:otherwise>
</c:choose>
```

return

Terminates task invocation.

Syntax

```
<c:return/>
```

Example

```
<c:if test="${input == null}">
  <c:return />
</c:if>
```

set

The **set** tag sets a variable in either the tasklet context or request context.

Syntax

```
<set var="VARIABLE" value="VALUE"/>
```

Attribute Descriptions

Required attributes: **var** and **value**.

var

The variable to set.

This attribute is required.

value

The value to set. The **value** attribute supports more complex expressions representing mathematical operations on two operands of the same type. The supported operands are [+, -, *, /, %, &, |]. Within a task the ‘&’ operator must be escaped like ‘&amp;’.

This attribute is required and can also be an expression.

Note

Where one operand is a constant and the other a variable, if required and possible the variable is coerced to the data type of the constant. Supported data types for mathematical operations are **int** and **float**.

requestScope

Boolean specifying whether the variable should be task or request scope. The default is set to `false`.

Example

```
<c:set var="onePlusOne" value="${one+1}"/>
```

when

Similar to **if**, but within a **choose** parent tag (represents the “if” of if/else logic). Provides JSTL support for Info*Engine tasks.

Syntax

```
<c:choose>
    <c:when test="CONDITION">
      <!-- processed when CONDITION evaluates to true -->
    </c:when>
    <c:otherwise>
      <!-- processed when CONDITION evaluates to false -->
    </c:otherwise>
</c:choose>
```

Attribute Descriptions

Required attribute: **test**.

test

Contains the conditional statement to be evaluated. The test attribute supports operators [==, !=, <, >, <=, and >=]. The numeric operators require that the operands be of a numeric data type for comparison. Where possible, if the operands are of different data types then an attempt is made to coerce one to the other to attempt comparison. If one of the operands is a constant, then an attempt is made to coerce the variable to the constant. Conditional expressions of arbitrary complexity can be specified. Portions of expressions to be evaluated together must be grouped with ( ) and expressions can then be joined by [||, &&].

Note

&& must be encoded as ‘&amp;&amp;’.

This attribute is required and is also an expression.

Example

```
<c:choose>
  <c:when test="${group == null  || group.elementCount==0}">
    <!-- do when group named "group" is empty -->
  </c:when>
  <c:otherwise>
    <!-- do when group contains elements -->
  </c:otherwise>
</c:choose>
```

Supplied Library Tags (Logging)

This tag library adds support for ease of use of log4j logging from within Info*Engine tasks without the need to use scriptlets. It is important to note that this tag library is not a custom extension, but is actually natively supported by the Info*Engine task compiler.

The following tags are included in the `http://www.ptc.com/infoengine/taglib/log` tag library. To use these tags, you must include a `taglib` directive in your task similar to the following:

```
   <%@ taglib uri="http://www.ptc.com/infoengine/taglib/log"
              prefix="log" %>
```

The syntax for these tags assumes that you have specified the `log` prefix in the `taglib` directive. If you specify a different prefix, use the prefix you specified in place of `log` in the tag syntax.

setLogger

Sets the log4j logger to be used. If this tag is not used, then the default logger name for a task is based on its location within the task root using its parent directory name prefixed with `com.infoengine.tasks`. For example, if your task is "/org/myCompany/MyTask.xml" then the default logger name is "com.infoengine.tasks.org.myCompany."

In general, the **setLogger** tag is seldom used since the default logger names should be sufficiently granular for adequate logging control. This tag is useful if you would your task to reuse another existing logger in use by your application, thereby allowing you to control logging for your application under a custom logger hierarchy.

Syntax

`<log:setLogger logger="LOGGER_NAME"/>`

Attribute Descriptions

Required attribute: **logger**

logger
    The logger name to use. This attribute is required and is also an expression.

Example

```
<log:setLogger logger="org.myorg.MyApplicationLogger"/>
```

info, debug, warn, error, trace

There is one tag for each corresponding log4j log level. Each tag issues a log message to the corresponding **log** method.

Syntax

`<log:LEVEL message="log message"/>`

Attribute Descriptions

Required attribute: **message**.

message
: The log messages to be issued. This attribute is required and is also an expression.

Example

```
<log:info message="INFO Message" />
<log:warn message="WARN Message" />
<log:debug message="Group g cotains ${g.elementCount} elements." />
<log:error message="ERROR Message" />
<log:trace message="TRACE Message" />
```

Supplied Library Tags (XSL)

This tag library adds basic support for XSL transformation of Info*Engine groups onto the client's output stream.

These tags are included in the `/com/infoengine/tlds/xsl.tld` tag library. To use these tags, you must include a `taglib` directive in your task similar to the following:

```
<%@ taglib uri="/com/infoengine/tlds/xsl.tld"
           prefix="xsl" %>
```

The syntax for these tags assumes that you have specified the `xsl` prefix in the `taglib` directive. If you specify a different prefix, use the prefix you specified in place of `xsl` in the tag syntax.

param

Specifies a value for an XSL parameter to be supplied for the transformation. This tag is embedded within a **transform** tag and is optional. Provides XSL transformation tag support.

Syntax

`<xsl:param name="PARAM_NAME" data="PARAM_DATA"/>`

Attribute Descriptions

Required attributes: **name** and **data**.

name
The XSL parameter name. This attribute is required and is also an expression.

data
The XSL parameter value. This attribute is required and is also an expression.

Example

```
<xsl:transform group="${toTransform}"
              xsl="/transform/to/xcel.xsl"
              fileName="my.xls"
              contentType="application/vnd.ms-excel">
  <xsl:param name="a" data="b" />
</xsl:transform>
```

transform

Performs XSL transformation on an input group and streams the result to the task's output stream. Supports embedded **param** tags. Provides XSL transformation tag support.

Syntax

```
<xsl:transform xsl="XSL" group="IE GROUP"
      fileName="FILENAME" contentType="CONTENT_TYPE">
  <xsl:param name="PARAM_NAME" data="PARAM_DATA"/>
</xsl:transform>
```

Attribute Descriptions

Required attributes: **xsl**.

xsl
Specifies the path to the XSL style sheet to apply. This attribute is required and is also an expression.

filename
Specifies the filename for the Content-Disposition HTTP header (default: '`transformed.xml`'). This attribute is also an expression.

contentType
Specifies the Content-Type for the Content-Disposition (default: `text/xml; charset="UTF-8"`) This attribute is also an expression.

group
The Info*Engine Group object to translate (not just its name, but the group). This attribute is also an expression.

Example

```
<xsl:transform group="${toTransform}"
```

```
              xsl="/transform/to/xcel.xsl"
              fileName="my.xls"
              contentType="application/vnd.ms-excel">
  <xsl:param name="a" data="b" />
</xsl:transform>
```

13

Display Webject Reference

The following topics contain information about Info*Engine display webjects. Info*Engine supports HTML and JPEG formats for display, and the webjects for each are grouped in separate sections. Each individual listing contains the webject name, a description of its use, details of its syntax, descriptions of all parameters, and, in most cases, an example.

Before their descriptions, parameters are grouped in the following categories:

- Required — The parameter is always required.
- Select — A relationship between the specified parameter and another parameter exists. For example, the HYPERLINK and TEMPLATE_URL parameters of the Display-Table webject are select because if you specify the HYPERLINK parameter, you must also specify the TEMPLATE_URL parameter.

  A parameter is also listed in the “Select” column when there is a relationship between the values in the specified parameter and the values in another parameter. For example, in the Display-Value webject the value entered for the TYPE parameter determines the format of the value you can enter for the SIZE parameter. Therefore, both the TYPE and SIZE parameters are listed in the “Select” column.
- Optional — The parameter is always optional and is not related to another parameter.

Display Webjects for HTML

The following webjects can be used to display output in an HTML format.

- Apply-XSL on page 300
- Display-Object on page 305
- Display-Resource on page 316
- Display-Selection on page 318
- Display-Table on page 327
- Display-Value on page 333
- Display-XML on page 341
- Echo-Request on page 342

Apply-XSL

DESCRIPTION

Applies an XSL stylesheet to an Info*Engine group of objects to generate a textual representation of the group. The generated output is determined by the contents of the XSL stylesheet.

SYNTAX

```
<ie:webject name="Apply-XSL" type="DSP">
  <ie:param name="CONTENT_TYPE" data="mimetype"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="XSL_PARAM" data="name_value_pair"/>
  <ie:param name="XSL_URL" data="url_location"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
XSL_URL		CONTENT_TYPE
		DBUSER
		GROUP_IN
		PASSWD
		XSL_PARAM

CONTENT_TYPE

Specifies the MIME content type to be associated with the result being displayed. Using html text as the content type returns an HTML-formatted document to the calling application or web browser.

The default for this parameter is `text/html`, which produces an XML-formatted document. This parameter is optional.

DBUSER

Specifies the username to be used to authenticate to XSL_URL. If the XSL templates to be used by the webject reside on a remote HTTP server, then this parameter should be used in conjunction with the PASSWD attribute.

This parameter is optional.

GROUP_IN

Identifies the name of the group to be used as the input source. The group can be a VDB group or a Context group. For further information about groups, see Info*Engine Data Management on page 31.

The default for this parameter is to use the last group defined in the VDB. This parameter is optional.

PASSWD

Specifies the password corresponding to DBUSER.

This parameter is optional.

XSL_PARAM

Defines XSL parameters that are then passed to the XSL stylesheet named in the XSL_URL parameter. You enter the value for the XSL_PARAM parameter in the form `XSL_`*`name`*`=XSL_`*`value`* where `XSL_`*`name`* is the name of a parameter in the XSL stylesheet and `XSL_`*`value`* is the value you want set for the parameter.

The default for XSL_PARAM is that no parameters are passed to the stylesheet. Multiple values can be specified for this parameter. This parameter is optional.

XSL_URL

Identifies the location of an XSL stylesheet to apply to the default output group. A relative URL or a fully qualified URL can be specified. Relative URLs are relative to the Info*Engine server task template root.
Fully qualified URLs are dereferenced using **Auth-Map** context group data. The **Auth-Map** context group is searched for a username and password based on the domain name found in the fully qualified URL. For example, assume that the fully qualified URL is:

```
http://machine.com/Windchill/infoengine/servlet/IE/
tasks/ com/company/createGroupData.xsl
```

The **Auth-Map** context group is searched for a username and password with `http://machine.com` as an INSTANCE name. If a username and password are found, BASIC authentication information is used when accessing the URL. If no username and password is found, no authentication information is sent to the remote web server.

If the data value contains the string `://` it is assumed to be a fully qualified Internet URL. If the data value does not contain the string, it is assumed to be a local file relative to the current task root directory.

This parameter is required.

Browser Example

Example webject

The following Apply-XSL webject formats the output from a task, then displays the output in a browser:

```
<ie:webject name="Apply-XSL" type="DSP">
  <ie:param name="CONTENT_TYPE" data="text/html"/>
  <ie:param name="XSL_URL"
               data="com/company/ApplyXsl.xsl"/>
</ie:webject>
```

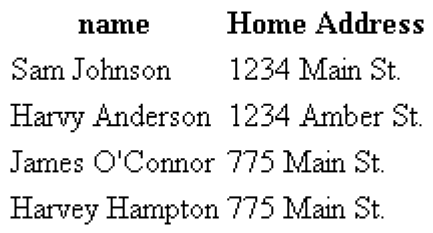
XSL stylesheet

The following XSL stylesheet is applied to an input group with a CLASS of “EmployeeData” with each element containing attributes NAME and ADDRESS attributes:

```
<?xml version='1.0'?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
    xmlns:wc="http://www.ptc.com/infoengine/1.0">
<xsl:template match="/wc:COLLECTION/EmployeeData">
<html>
<head><title>Apply-XSL Example</title></head>
<body>
<table>
<tr>
  <th>name</th><th>Home Address</th>
</tr>
<xsl:for-each select="wc:INSTANCE">
  <tr>
    <td><xsl:value-of select="NAME"/></td>
    <td><xsl:value-of select="ADDRESS"/></td>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Browser display

The output is generated using HTML by specifying the CONTENT_TYPE as `text/html`. The resulting web browser display consists of the following table of names and addresses:



name	Home Address
Sam Johnson	1234 Main St.
Harvy Anderson	1234 Amber St.
James O'Connor	775 Main St.
Harvey Hampton	775 Main St.

Excel Example

Example JSP Page

The following JSP page contains the Apply-XSL webject. The webject defines XSL parameters, then uses a stylesheet to format the output from a task for display in Excel:

```
<%@page language="java" session="false" errorPage="../IEError.jsp"
contentType="application/ms-excel"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
prefix="ie"%>

<%
/********************************************
 *
 * DESCRIPTION
 * Build html table for import Microsoft Word, Excel, Powerpoint 2000.
 *
 ********************************************/
 // execute create group task to get group info
%>

<ie:task uri="com/company/CreateGroup.xml"/>

<%
 // call Apply-XSL Webject to translate data into excel data
%>

<ie:webject name="Apply-XSL" type="DSP">
  <ie:param name="XSL_URL" data="com/company/ExcelWorkbook.xsl"/>
  <ie:param name="XSL_PARAM" data="CLASS='EmployeeData'"/>
  <ie:param name="XSL_PARAM"
            data="HEADERS='NAME=Employee,ADDRESS=Home Address,'"/>
  <ie:param name="XSL_PARAM" data="USE-COLUMNS='NAME,ADDRESS,EMAIL'"/>
</ie:webject>
```

XSL stylesheet

In the example JSP page, the following XSL parameters are passed to the XSL stylesheet:

```
CLASS='EmployeeData'
HEADERS='NAME=Employee,ADDRESS=Home Address,'
USE-COLUMNS='NAME,ADDRESS,EMAIL'
```

The parameters change the class, columns used, and headers defined in the stylesheet so that the stylesheet can manipulate and format the data.

The Apply-XSL webject on the JSP page calls the following XSL stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:wc="http://www.ptc.com/infoengine/1.0">

<xsl:output omit-xml-declaration="yes"/>

<xsl:param name="CLASS">EMP</xsl:param>
<xsl:param name="USE-COLUMNS">EMPNO,ENAME,POS,</xsl:param>
<xsl:param name="HEADERS">EMPNO=E,ENAME=N,POS=P,</xsl:param>

<xsl:template match="/wc:COLLECTION">
  <table border='0' cellpadding='0' cellspacing='0' width='627'
         style='border-collapse:collapse;table-layout:fixed;width:350pt'>
    <col width='200'
         style='mso-width-source:userset;mso-width-alt:5400;width:100pt'/>
    <col width='200'
         style='mso-width-source:userset;mso-width-alt:5400;width:100pt'/>
    <col width='200'
         style='mso-width-source:userset;mso-width-alt:6700;width:175pt'/>

    <xsl:for-each select="//*[name()=$CLASS]">
      <tr height='17' style='height:12.75pt'>
        <xsl:call-template name="table-headers"/>
      </tr>
      <xsl:for-each select="wc:INSTANCE">
      <tr>
          <xsl:for-each select="*">
            <xsl:call-template name="html-table-value"/>
          </xsl:for-each>
      </tr>
      </xsl:for-each>
    </xsl:for-each>
  </table>

</xsl:template>

<xsl:template name="html-table-value">
  <xsl:variable name="TARGET" select="name()"/>
  <xsl:if test="contains( $USE-COLUMNS, name() )">
    <td height='68' class='xl27' style='height:20.0pt'>
      <xsl:value-of select="."/>
    </td>
  </xsl:if>
</xsl:template>

<xsl:template name="table-headers">
  <xsl:for-each select="child::wc:INSTANCE[position()=1]/child::*">
    <xsl:variable name="EQUALS" select="'='"/>
    <xsl:variable name="DELIM" select="','"/>
    <xsl:if test="contains( $USE-COLUMNS, name() )">

      <xsl:message>
        <xsl:text> Column Header </xsl:text>
        <xsl:value-of select="$HEADERS"/>
        <xsl:text> Name </xsl:text>
        <xsl:value-of select="name()"/>
      </xsl:message>
```

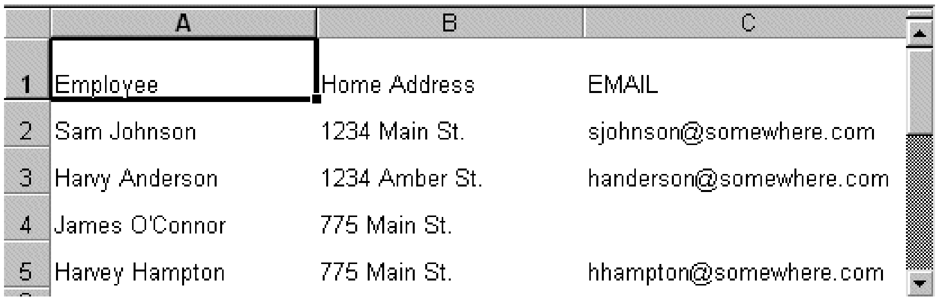
```
       <td height='17' class='xl24' width='100' style='height:25.75pt
;width:100pt'>
           <xsl:if test="contains( $HEADERS, name() )">
             <xsl:variable name="TEMP"
                                 select="substring-after( $HEADERS, name
() )"/>
             <xsl:variable name="TEMP1"
                                 select="substring-after( $TEMP, $EQUALS
)"/>
             <xsl:variable name="VALUE" select="substring-before(
$TEMP1, $DELIM)"/>
             <xsl:value-of select="$VALUE"/>
           </xsl:if>
           <xsl:if test="not(contains( $HEADERS, name() ))">
             <xsl:value-of select="name()"/>
           </xsl:if>
         </td>

       </xsl:if>
     </xsl:for-each>
   </xsl:template>

   </xsl:stylesheet>
```

MS Excel display

To display the output directly in MS Excel, execute the URL for the page from the Excel **Open** dialog. For example, if the `ExcelWorkbook.jsp` page were saved to `<Windchill>/codebase/com/company/ExcelWorkbook.jsp` you might enter the following URL::

`http://host/Windchill/com/company/ExcelWorkbook.jsp`

The following picture displays the resulting Excel columns. Notice that only two of the three headers were supplied. The EMAIL header comes from the column name.



	A	B	C
1	Employee	Home Address	EMAIL
2	Sam Johnson	1234 Main St.	sjohnson@somewhere.com
3	Harvy Anderson	1234 Amber St.	handerson@somewhere.com
4	James O'Connor	775 Main St.	
5	Harvey Hampton	775 Main St.	hhampton@somewhere.com

Display-Object

DESCRIPTION

Displays a group of objects in a general way, allowing the insertion of markup language before and after objects and attributes. If the BORDER parameter is not specified, no formatting is done. If BORDER is specified, a simple table is generated.

SYNTAX

```
<ie:webject name=" Display-Object" type="DSP">
  <ie:param name="ATTRIBUTE" data="attribute"/>
  <ie:param name="ATTRIBUTE_SEPARATOR" data="separator"/>
  <ie:param name="BORDER" data="[pixels | 1]"/>
  <ie:param name="CAPTION" data="text"/>
  <ie:param name="CELLPADDING" data="pixels"/>
  <ie:param name="CELLSPACING" data="pixels"/>
  <ie:param name="DISPLAY_ATTRIBUTE_NAME" data="[TRUE |
                                                      FALSE]"/>
  <ie:param name="DISPLAY_ATTRIBUTE_VALUE" data="[TRUE |
                                                      FALSE]"/>
  <ie:param name="FOOTER" data="text"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="HEADER_SEPARATOR" data="separator"/>
  <ie:param name="MAX" data="maximum"/>
  <ie:param name="OBJECT_SEPARATOR" data="separator"/>
  <ie:param name="POST_ATTRIBUTE_NAME_TEXT" data="text"/>
  <ie:param name="POST_ATTRIBUTE_TEXT" data="text"/>
  <ie:param name="POST_CAPTION_TEXT" data="text"/>
  <ie:param name="POST_FOOTER_TEXT" data="text"/>
  <ie:param name="POST_HEADER_TEXT" data="text"/>
  <ie:param name="POST_OBJECT_TEXT" data="text"/>
  <ie:param name="POST_TABLE_TEXT" data="text"/>
  <ie:param name="POST_TITLE_TEXT" data="text"/>
  <ie:param name="POST_VALUE_TEXT" data="text"/>
  <ie:param name="PRE_ATTRIBUTE_NAME_TEXT" data="text"/>
  <ie:param name="PRE_ATTRIBUTE_TEXT" data="text"/>
  <ie:param name="PRE_CAPTION_TEXT" data="text"/>
  <ie:param name="PRE_FOOTER_TEXT" data="text"/>
  <ie:param name="PRE_HEADER_TEXT" data="text"/>
  <ie:param name="PRE_OBJECT_TEXT" data="text"/>
  <ie:param name="PRE_TABLE_TEXT" data="text"/>
  <ie:param name="PRE_TITLE_TEXT" data="text"/>
  <ie:param name="PRE_VALUE_TEXT" data="text"/>
  <ie:param name="START" data="starting_element"/>
  <ie:param name="TITLE" data="text"/>
  <ie:param name="UNDEFINED" data="string"/>
  <ie:param name="VALUE_SEPARATOR" data="separator"/>
</ie:webject>
```

PARAMETERS

Note

All parameters are optional.

ATTRIBUTE

Specifies which attributes are shown as each object is displayed. For this webject, you can specify two types of attributes: actual attributes and pseudo-attributes.

Actual attributes are explicitly stated as the value of the parameter. For example, specifying `data="ename"` includes the “ename” attribute when the object is displayed. Actual attribute values are always displayed.

Pseudo-attributes, unlike actual attributes, are usually obtained from a previous WML card. They often look like this in a webject:

```
data="$(attribute)"
```

A pseudo-attribute value can be a combination of text and variable references. Variable substitution is made from the attributes on the current object. For example, assume that the group named in the GROUP_IN parameter contains the two objects: `name=Sam sal=200` and `name=Chen sal=300`. Then, the parameter:

```
   <ie:param name="ATTRIBUTE" data="Employee $(name) is paid $(sal)"/>
```

generates the display text:

```
   Employee Sam is paid 200
  Employee Chen is paid 300
```

Notice the double quotation marks around the value of the parameter shown previously. These quotes indicate that the values of the parameter should be HTML-encoded to allow for any special characters to be properly rendered in the wireless device during display. Single quotes around the outside of the value could also have been used. If double quotes are to be used within the value for a parameter, use single quotation marks around the outside of the value. If single quotations (for example, an apostrophe) are to be used within the value for a parameter, use double quotation marks around the outside of the value.

For example, to generate the following name:

```
   NAME='Sam Johnson';
```

either of the following parameter combinations would be correct in the Display-Object webject:

```
   <ie:param name="ATTRIBUTE" data="NAME
```

```
                    =&#39;$(namecode)&#39;"/>

   <ie:param name="ATTRIBUTE" data="NAME='$(NAME)';"/>
```

This parameter must be specified if DISPLAY_ATTRIBUTE_NAME is to be used. The default for this parameter is to display all attributes for each object. Multiple values can be specified for this parameter. This parameter is optional.

ATTRIBUTE_SEPARATOR

Specifies the text or HTML to be used between attributes. When there is a border, the default for this parameter is `\n`; when there is no border, the default is `""`. This parameter is optional.

BORDER

Determines the width, in pixels, of the HTML table border around the group of information to be displayed.

If the BORDER parameter is specified with a value other than `""`, HTML table tags are generated by default using other parameters. The default settings display objects as rows in a table and display attributes as columns in a table.

The following table shows the default values of the parameters that are affected by whether or not there is a border specified. There are two sets of defaults:

- Defaults when there is a border (BORDER is specified with a value other than `""`).
- Defaults when there is no border (BORDER is not specified or is set to `""`).

Parameter	HTML Default Value Used with a Border	HTML Default Value Used with No Border
PRE_TABLE_ TEXTPOST _TABLE_TEXT	`"<table border=BORDER cellspacing=CELLSPACING cellpadding=CELLPADDING> \n""</table>\n"`	`""""`
PRE_HEADER_TEXT HEADER_SEPARATOR POST HEADER_TEXT	`"<th>""""</th>"`	`""""""`
PRE_OBJECT_TEXT POST_OBJECT_TEXT	`"<tr>""</tr>\n"`	`""""`
PRE_ATTRIBUTE_ TEXT POST_ATTRIBUTE_ TEXT	`"<td>""</td>"`	`""""`

Parameter	HTML Default Value Used with a Border	HTML Default Value Used with No Border
PRE_CAPTION_TEXT POST_CAPTION_TEXT	`"<caption>""</caption>\n"`	`""""`
VALUE_SEPARATOR ATTRIBUTE_ SEPARATOR OBJECT_SEPARATOR	`"<br>""\n""\n"`	`""""""`
PRE_TITLE_TEXT_ TITLE POST_TITLE_ TEXT	`"""""\n"`	`""""""`

The default value of BORDER is `""`. Default values for parameters affected by whether or not there is a border can be overridden by explicitly setting the parameters. This parameter is optional.

CAPTION

Specifies the text for a table caption. The default for this parameter is `""`. This parameter is optional.

CELLPADDING

Specifies the amount of space, in pixels, between the border of a cell and its contents. The default for this parameter is `""`. This parameter is optional.

CELLSPACING

Specifies the amount of space, in pixels, around the outside of the cells in a table and between the cells in a table. The default for this parameter is `""`. This parameter is optional.

DISPLAY_ATTRIBUTE_NAME

Determines if the attribute name is shown. This parameter only applies if specific attributes are specified on an ATTRIBUTE parameter. The default for this parameter is FALSE. Use TRUE to enable. This parameter is optional.

DISPLAY_ATTRIBUTE_VALUE

Determines if the attribute value is shown. The default for this parameter is TRUE. Use FALSE to disable. This parameter is optional.

FOOTER

Specifies the text for a table footer. The default for this parameter is `""`. This parameter is optional.

GROUP_IN

Identifies the name of the group to be used as an input source. The group can be a VDB group or a Context group. For further information about groups, see Info*Engine Data Management on page 31.

The default for this parameter is to use the last group defined in the VDB. This parameter is optional.

HEADER_SEPARATOR

Specifies the text or HTML to be used between column headers. The default for this parameter is `""`. This parameter is optional.

MAX

Defines the maximum number of elements to display. Specifying a value for this parameter controls how much data is returned through the webject.

By using parameter value substitution for the MAX and START parameters, a series of webjects can be coded that allow data in any group to be displayed in manageable result sets. For more information, see Dynamic Parameter Value Substitution on page 44.

The default for this parameter is to display all remaining elements in the group, starting with the element identified in the START parameter. This parameter is optional.

OBJECT_SEPARATOR

Specifies the text or HTML to be used between table rows. When there is a border, the default for this parameter is `\n`; when there is no border, the default is `""`. This parameter is optional.

POST_ATTRIBUTE_NAME_TEXT

Specifies the text or HTML generated after each attribute name. The default for this parameter is `</b>`. This parameter is optional.

POST_ATTRIBUTE_TEXT

Specifies the text or HTML generated after each attribute value. When there is a border, the default for this parameter is `</td>`; when there is no border, the default is `""`. This parameter is optional.

POST_CAPTION_TEXT

Specifies the text or HTML generated after each caption. When there is a border, the default for this parameter is `</caption>\n`; when there is no border, the default is `""`. This parameter is optional.

POST_FOOTER_TEXT

Specifies the text or HTML generated after a table footer. The default for this parameter is `\n`. This parameter is optional.

POST_HEADER_TEXT

Specifies the text or HTML generated after each column header. When there is a border, the default for this parameter is `</th>`; when there is no border, the default is `""`. This parameter is optional.

POST_OBJECT_TEXT

Specifies the text or HTML generated after each object. When there is a border, the default for this parameter is `</tr>\n`; when there is no border, the default is `""`. This parameter is optional.

POST_TABLE_TEXT

Specifies the text or HTML generated after each group of objects. When there is a border, the default for this parameter is `</table>\n`; when there is no border, the default is `\n</pre>\n` . This parameter is optional.

POST_TITLE_TEXT

Specifies the text or HTML generated after a title. When there is a border, the default for this parameter is `\n`; when there is no border, the default is `""`. This parameter is optional.

POST_VALUE_TEXT

Specifies the text or HTML generated after each value of a multi-valued attribute. The default for this parameter is `""`. This parameter is optional.

PRE_ATTRIBUTE_NAME_TEXT

Specifies the text or HTML generated before each attribute name. The default for this parameter is `<b>`. This parameter is optional.

PRE_ATTRIBUTE_TEXT

Specifies the text or HTML generated before each attribute value. When there is a border, the default for this parameter is `<tr>`; when there is no border, the default is `""`. This parameter is optional.

PRE_CAPTION_TEXT

Specifies the text or HTML generated before each caption. When there is a border, the default for this parameter is `<caption>`; when there is no border, the default is `""`. This parameter is optional.

PRE_FOOTER_TEXT

Specifies the text or HTML generated before a table footer. The default for this parameter is `""`. This parameter is optional.

PRE_HEADER_TEXT

Specifies the text or HTML generated before each column header. When there is a border, the default for this parameter is `<th>`; when there is no border, the default is `""`. This parameter is optional.

PRE_OBJECT_TEXT

Specifies the text or HTML generated before each object. When there is a border, the default for this parameter is `<tr>`; when there is no border, the default is `""`. This parameter is optional.

PRE_TABLE_TEXT

Specifies the text or HTML generated before each group of objects. When there is a border, the default for this parameter is `<table border=BORDER cellspacing=CELLSPACING cellpadding=CELLPADDING>\n`; when there is no border, the default is `<pre>`. This parameter is optional.

PRE_TITLE_TEXT

Specifies the text or HTML generated before a title. The default for this parameter is `""`. This parameter is optional.

PRE_VALUE_TEXT

Specifies the text or HTML generated before each value of a multi-valued attribute. The default for this parameter is `""`. This parameter is optional.

START

Specifies the number of the first element in the group to display. Use this parameter in conjunction with the MAX parameter to control how much data is returned through the webject.

By using parameter value substitution for the MAX and START parameters, a series of webjects can be coded that allow data in any group to be displayed in manageable result sets. For more information, see Dynamic Parameter Value Substitution on page 44.

The default for this parameter is to start with the first record in the group. This parameter is optional.

TITLE

Specifies the text for a title. The default for this parameter is `""`. This parameter is optional.

VALUE_SEPARATOR

Specifies the text or HTML to be used between values of attributes if the table contains multi-valued attributes. When there is a border, the default for this parameter is `<br>`; when there is no border, the default is “`,`”. This parameter is optional.

UNDEFINED

Sets the value to display for an undefined value. An undefined value is either a nonexistent attribute or an attribute that has a null value. An attribute containing an empty string (`""`) is not interpreted as being undefined. The default for this parameter is `""`. This parameter is optional.

EXAMPLES

The following Display-Object webject examples assume an input group containing multiple elements; each with a name, address and email attribute value:

DEFAULT DISPLAY

Webject

```
<ie:webject name="Display-Object" type="DSP"/>
```

Output

```
Sam Johnson1234 Main St.sjohnson@somewhere.comHarvy Anderson1234
Amber St.handerson@somewhere.comJames O'Connor775 Main St.Harvey
Hampton775 Main St.hhampton@somewhere.com
```

COMMA SEPARATED LIST OF ALL ATTRIBUTES

Webject

```
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="ATTRIBUTE_SEPARATOR" data=","/>
</ie:webject>
```

Output

```
Sam Johnson,1234 Main St.,sjohnson@somewhere.comHarvy Anderson,1234
Amber St.,handerson@somewhere.comJames O'Connor,775 Main St.,Harvey
Hampton,775 Main St.,hhampton@somewhere.com
```

COMMA SEPARATED LIST OF SOME ATTRIBUTES, ONE ELEMENT PER LINE

Webject

```
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="ATTRIBUTE" data="name,address" delim=","/>
  <ie:param name="ATTRIBUTE_SEPARATOR" data=","/>
  <ie:param name="OBJECT_SEPARATOR" data="<br>"/>
</ie:webject>
```

Output

```
Sam Johnson,1234 Main St.
Harvy Anderson,1234 Amber St.
James O'Connor,775 Main St.
Harvey Hampton,775 Main St.
```

FREE-FORM SUBSTITUTION, ONE ELEMENT PER LINE

Webject

```
<ie:webject name="Display-Object" type="DSP">
```

```
  <ie:param name="ATTRIBUTE" data="Employee $[name] lives at
                                             $[ADDRESS]  &amp;"/>
  <ie:param name="ATTRIBUTE" data="$[NAME]'s email address is
                                              &lt;$[email]&gt;"/>
  <ie:param name="ATTRIBUTE_SEPARATOR" data=" "/>
  <ie:param name="OBJECT_SEPARATOR" data="<br>"/>
</ie:webject>
```

Output

```
Employee Sam Johnson lives at 1234 Main St. & Sam Johnson's email address is
<sjohnson@somewhere.com>
Employee Harvy Anderson lives at 1234 Amber St. & Harvy Anderson's email
address is <handerson@somewhere.com>
Employee James O'Connor lives at 775 Main St. & James O'Connor's email
address is <>
Employee Harvey Hampton lives at 775 Main St. & Harvey Hampton's email
address is <hhampton@somewhere.com>
```

ESCAPE CHARACTERS

Webject

```
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="ATTRIBUTE" data="User's <name> = $[EMAIL]"/>
  <ie:param name="OBJECT_SEPARATOR" data="<br>"/>
</ie:webject>
```

Note

Output displays with no `<name>` because the < and > signs are interpreted as HTML tags. If "<" and ">" are desired in the output, use the special characters `&lt` and `&gt`.

Output

```
User's = sjohnson@somewhere.com
User's = handerson@somewhere.com
User's =
User's = hhampton@somewhere.com
```

Note

View source to verify unquoted < and > characters.

Generated HTML

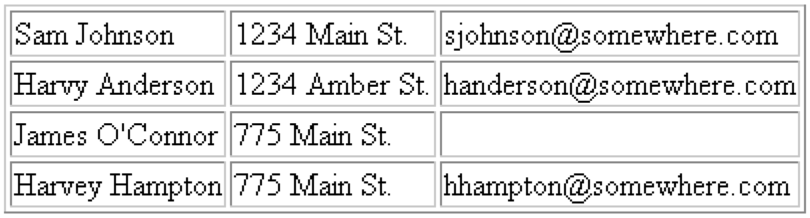
```
User's <name> = sjohnson@somewhere.com<br>
User's <name> = handerson@somewhere.com<br>
User's <name> = <br>
User's <name> = hhampton@somewhere.com
```

DEFAULT TABLE

Webject

```
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="BORDER" data="1"/>
</ie:webject>
```

Output



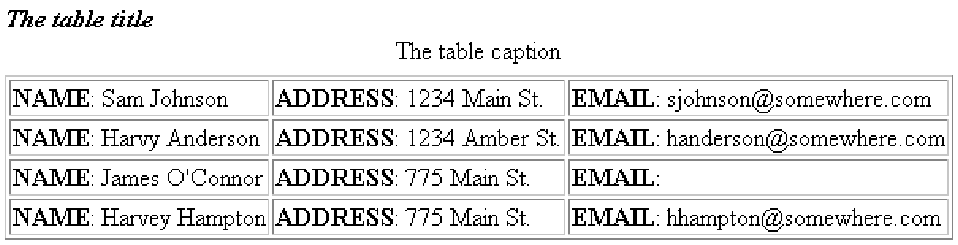
Sam Johnson	1234 Main St.	sjohnson@somewhere.com
Harvy Anderson	1234 Amber St.	handerson@somewhere.com
James O'Connor	775 Main St.	
Harvey Hampton	775 Main St.	hhampton@somewhere.com

DISPLAY TABLE WITH TITLE AND CAPTION

Webject

```
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="BORDER" data="1"/>
  <ie:param name="DISPLAY_ATTRIBUTE_NAME" data="TRUE"/>
  <ie:param name="CAPTION" data="The table caption"/>
  <ie:param name="PRE_TITLE_TEXT" data="<b><i>"/>
  <ie:param name="TITLE" data="The table title"/>
  <ie:param name="POST_TITLE_TEXT" data="</i></b>"/>
</ie:webject>
```

Output



The table title

The table caption

NAME: Sam Johnson	**ADDRESS**: 1234 Main St.	**EMAIL**: sjohnson@somewhere.com
NAME: Harvy Anderson	**ADDRESS**: 1234 Amber St.	**EMAIL**: handerson@somewhere.com
NAME: James O'Connor	**ADDRESS**: 775 Main St.	**EMAIL**:
NAME: Harvey Hampton	**ADDRESS**: 775 Main St.	**EMAIL**: hhampton@somewhere.com

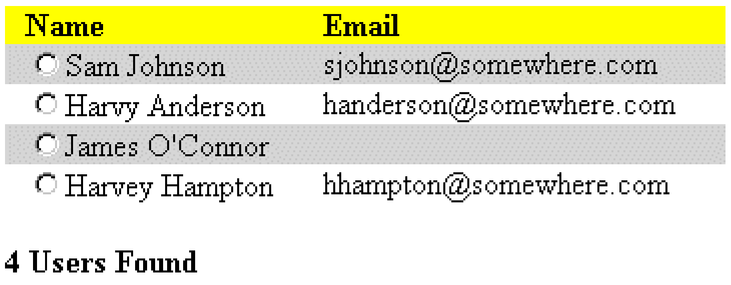
JAVASCRIPT

The following example shows how JavaScript can be incorporated into the Display-Object webject.

Webject

```
<script> var itemcnt=0; </script>
<table border=0 cellspacing=0 cellpadding=0>
   <th bgcolor=yellow align=left> &nbsp; </th>
   <th bgcolor=yellow align=left> Name </th>
   <th bgcolor=yellow align=left> &nbsp; </th>
   <th bgcolor=yellow align=left> Email </th>
   <th bgcolor=yellow align=left> &nbsp; </th>
  <ie:webject name="Display-Object" type="DSP">
    <ie:param name="PRE_OBJECT_TEXT" data="<script> itemcnt++;
                   if (itemcnt % 2 == 0) document.write('<tr
                   bgcolor=#FFFFFF>'); else document.write('<tr
                   bgcolor=#D3D3D3>'); </script>"/>
    <ie:param name="ATTRIBUTE" data="<td width=10>
                   &nbsp; </td>"/>
    <ie:param name="ATTRIBUTE" data="<td align=left
                   valign=middle> <input type=radio
                   name=rowid>$[NAME]</td>"/>
    <ie:param name="ATTRIBUTE" data="<td width=25>
                   &nbsp; </td>"/>
    <ie:param name="ATTRIBUTE" data="<td align=left
                   valign=middle>$[EMAIL]</td>"/>
    <ie:param name="ATTRIBUTE" data="<td width=25>
                   &nbsp; </td>"/>
    <ie:param name="POST_OBJECT_TEXT" data="</tr>"/>
  </ie:webject>
</table>
```

Output



Name	Email
○ Sam Johnson	sjohnson@somewhere.com
○ Harry Anderson	handerson@somewhere.com
○ James O'Connor	
○ Harvey Hampton	hhampton@somewhere.com

4 Users Found

Display-Resource

DESCRIPTION

Extracts and displays a localized string from either a group created with the Get-Resource webject or directly from a resource bundle.

SYNTAX

```
<ie:webject name="Display-Resource" type="DSP">
  <ie:param name="BUNDLE" data="bundle_name"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="KEY" data="key"/>
  <ie:param name="PARAM" data="text"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
KEY	GROUP_IN	PARAM
	BUNDLE	

BUNDLE

Indicates the Java class resource bundle from which the localized string is extracted. This parameter is required if no GROUP_IN is supplied.

GROUP_IN

Indicates the bundle group from which the message is to be extracted. This parameter is required if no BUNDLE is supplied.

KEY

Indicates the key into the resource bundle. The key can be either the number or the actual Java variable reference name. This parameter is required.

PARAM

Specifies the text to be inserted into the localized message. For example, if the extracted string contains a variable place holder for text, such as:

```
The {0} webject has failed.
```

Then the text specified for PARAM replaces the place holder. For example if the value specified for PARAM is “Return-Group”, then the resulting string becomes:

```
The Return-Group webject has failed.
```

Multiple values can be specified for this parameter. The default for this parameter is `""`, resulting in no substitution being performed. This parameter is optional.

EXAMPLES

For more information on the Get-Resource webject used below, see Get-Resource on page 427.

Webject

```
<ie:webject name="Get-Resource" type="MGT">
  <ie:param name="BUNDLE"
                      data="com.infoengine.util.IEResource"/>
```

```
    <ie:param name="GROUP_OUT" data="IEResource"/>
  </ie:webject>

  <b>
  <ie:webject name="Display-Resource" type="DSP">
    <ie:param name="GROUP_IN" data="IEResource"/>
    <ie:param name="KEY" data="19"/>
  </ie:webject>
  </b>
```

Output

The displayed text is a localized message pulled from an Info*Engine resource bundle.

Display-Selection

DESCRIPTION

Generates HTML form elements such as OPTION and SELECT with attributes of NAME, SIZE, and MULTIPLE, producing a selectable list in the form of a drop-down menu or a list box in a browser window.

Note

The FORM element is not automatically generated by this webject. To display the lists correctly, include the `<FORM>` and `</FORM>` tags as shown in the examples.

SYNTAX

```
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="attribute"/>
  <ie:param name="ATTRIBUTE_VALUE" data="attribute_value"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="LEADER" data="leader_char"/>
  <ie:param name="LIST_HEADER" data="addl_item"/>
  <ie:param name="LIST_HEADER_VALUE" data="addl_item_value"/>
  <ie:param name="MULTIPLE" data="[TRUE | FALSE]"/>
  <ie:param name="NAME" data="name"/>
  <ie:param name="SELECTED" data="preselected_value"/>
  <ie:param name="SIZE" data="display_height"/>
  <ie:param name="WIDTH" data="element_width"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	LIST_HEADER	ATTRIBUTE
	LIST_HEADER_VALUE	ATTRIBUTE_VALUE
	MULTIPLE	GROUP_IN
	SELECTED	LEADER
		NAME
		SIZE
		WIDTH

ATTRIBUTE

Specifies the name of the object attribute whose value is used as the value attribute in the generated HTML OPTION element and as the value displayed, and whose attribute name is used as the name value for the generated HTML SELECT element.

If ATTRIBUTE_VALUE is specified, then the name value of the HTML SELECT element and the value attribute of the HTML OPTION element is determined by ATTRIBUTE_VALUE, while the value displayed by the HTML OPTION element remains the value specified by ATTRIBUTE.

The default for this parameter is to use and display all attributes. Multiple values can be specified for this parameter. This parameter is optional.

ATTRIBUTE_VALUE

Specifies the name of the webject attribute whose value is used as the value attribute in the generated HTML OPTION element, as well as the name value for the generated HTML SELECT element; the value displayed remains the ATTRIBUTE value.

If ATTRIBUTE is not specified, then ATTRIBUTE_VALUE determines the name value for the generated HTML SELECT element and the value attribute in the generated HTML OPTION element, while the value displayed is determined by the ATTRIBUTE default.

The default for this parameter is to use the same attribute for the value submitted as for the value displayed. This parameter is optional.

GROUP_IN

Identifies the name of the group to be used as an input source. The group can be a VDB group or a Context group. The default for this parameter is to use the last group defined in the VDB. This parameter is optional.

For further information about groups, see Info*Engine Data Management on page 31.

LEADER

Specifies a string value to place between multiple attribute values. If no width parameter is specified, the LEADER value becomes a separator between the attribute values. The LEADER value can consist of one or more characters. If width is specified, the attribute value is padded with the LEADER value until the width is met or exceeded. The default for this parameter is `"_"`. This parameter is optional.

LIST_HEADER

Specifies an additional selectable item to be displayed at the top of the drop-down list or list box. LIST_HEADER and LIST_HEADER_VALUE parameters are treated as a pair. If LIST_HEADER is specified, LIST_HEADER_VALUE must also be specified. Multiple values can be specified for this parameter.

LIST_HEADER_VALUE

Specifies the value of the LIST_HEADER item that is to be returned when the form is submitted, if the LIST_HEADER item is selected by the user. The LIST_HEADER parameter must be specified for the value of this parameter to be valid. Multiple values can be specified for this parameter.

MULTIPLE

Acts as a flag, allowing multiple objects to be selected from the selectable list. The default for this parameter is FALSE. Specify TRUE to enable multiple selections. You must also set the parameter to TRUE to be able to preselect all of the list values using the SELECTED parameter.

NAME

Sets the NAME value of the generated HTML SELECT element. The default for this parameter is to use the attribute name specified by ATTRIBUTE; or, if ATTRIBUTE_VALUE is specified, then the default for this parameter is to use the attribute name specified by ATTRIBUTE_VALUE. If neither ATTRIBUTE nor ATTRIBUTE_VALUE are specified, then the default for this parameter is to use the name of the first attribute. This parameter is optional.

SELECTED

Specifies which attribute value to use as the default selection in a drop-down menu or list box. You can select no values, a single attribute value, or all values. The default for this parameter is `""`. To set the default selection to all the values in the list, set the value of this parameter to `"*"`. The MULTIPLE parameter must be enabled for all values to be preselected.

SIZE

Sets the number of options to be displayed at one time in the generated list box. If no value is specified, the default for this parameter is one item. If the default is used, or one item is specified, a drop-down menu is generated. If two or more items are specified, then a list box is displayed; scroll bars display when more items are returned than the specified SIZE parameter can display. This parameter is optional.

WIDTH

Sets the width of the OPTION element in the drop-down menu or list box. The default for this parameter is the size of the attribute. If more than one attribute is specified, then each attribute can have a WIDTH parameter specified. If more attributes than WIDTH parameters are specified, then the WIDTH parameters are applied on a one-to-one basis beginning with the first attribute, with the remaining attributes using the default value. This parameter is optional.

EXAMPLES

The following Display-Selection webject examples assume an input group containing multiple elements; each with a name, address and email attribute value:

DEFAULT DISPLAY

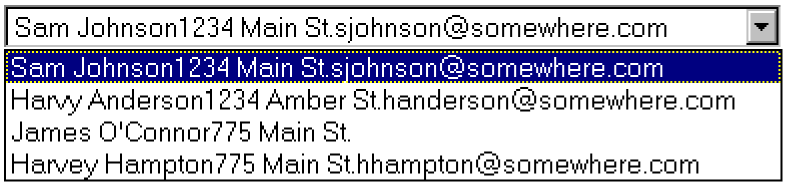
Note

The default for the SIZE parameter results in a drop-down menu.

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP"/>
</form>
```

Output



Sam Johnson1234 Main St.sjohnson@somewhere.com

Sam Johnson1234 Main St.sjohnson@somewhere.com
Harvy Anderson1234 Amber St.handerson@somewhere.com
James O'Connor775 Main St.
Harvey Hampton775 Main St.hhampton@somewhere.com

Generated HTML

```
<form>
<select name='NAME'>
<option value='Sam Johnson'>Sam Johnson1234 Main
                             St.sjohnson@somewhere.com</option>
<option value='Harvy Anderson'>Harvy Anderson1234 Amber
                            St.handerson@somewhere.com</option>
<option value='James O'Connor'>James O'Connor775 Main
                                                  St.</option>
<option value='Harvey Hampton'>Harvey Hampton775 Main
                             St.hhampton@somewhere.com</option>
```

```
</select>
</form>
```

DISPLAY A SINGLE ATTRIBUTE

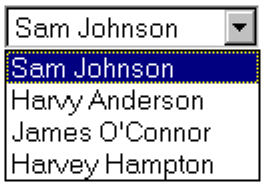
Note

Specifying a value of "1" for the SIZE parameter results in a drop-down menu.

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="SIZE" data="1"/>
</ie:webject>
</form>
```

Output



Generated HTML

```
<form>
<select name='NAME' size=1>
<option value='Sam Johnson'>Sam Johnson</option>
<option value='Harvy Anderson'>Harvy Anderson</option>
<option value='James O'Connor'>James O'Connor</option>
<option value='Harvey Hampton'>Harvey Hampton</option>
</select>
</form>
```

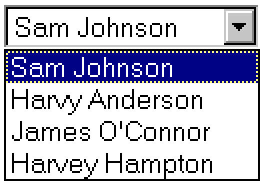
DISPLAY SINGLE ATTRIBUTE WITH DIFFERENT OPTION TAG VALUE

The following example shows the list that is generated when the ATTRIBUTE_VALUE parameter specified differs from the attribute named in the ATTRIBUTE parameter. In this case the NAME attribute is displayed in a list box, while the value returned when the form is submitted is the ADDRESS attribute.

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE_VALUE" data="ADDRESS"/>
</ie:webject>
</form>
```

Output



Generated HTML

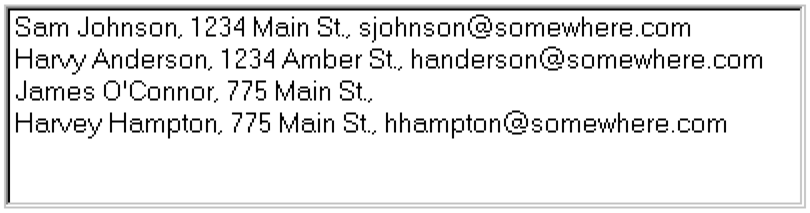
```
<form>
<select name='ADDRESS'>
<option value='1234 Main St.'>Sam Johnson</option>
<option value='1234 Amber St.'>Harvy Anderson</option>
 <option value='775 Main St.'>James O'Connor</option>
<option value='775 Main St.'>Harvey Hampton</option>
</select>
</form>
```

DISPLAY MULTIPLE ATTRIBUTES WITH SEPARATOR

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE_VALUE" data="ADDRESS"/>
  <ie:param name="LEADER" data = ", "/>
  <ie:param name="SIZE" data="6"/>
</ie:webject>
</form>
```

Output



```
Sam Johnson, 1234 Main St., sjohnson@somewhere.com
Harvy Anderson, 1234 Amber St., handerson@somewhere.com
James O'Connor, 775 Main St.,
Harvey Hampton, 775 Main St., hhampton@somewhere.com
```

Generated HTML

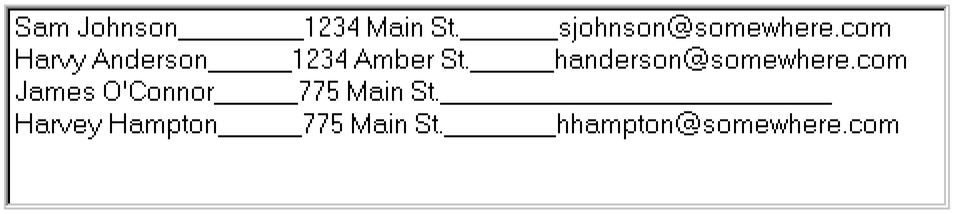
```
<form>
<select name='ADDRESS' size=6>
<option value='1234 Main St.'>Sam Johnson, 1234 Main St.,
                                sjohnson@somewhere.com</option>
<option value='1234 Amber St.'>Harvy Anderson, 1234 Amber St.,
                               handerson@somewhere.com</option>
<option value='775 Main St.'>James O'Connor, 775 Main St.,
                                                      </option>
<option value='775 Main St.'>Harvey Hampton, 775 Main St.,
                                hhampton@somewhere.com</option>
</select>
</form>
```

DISPLAY MULTIPLE ATTRIBUTES WITH SEPARATOR AND WIDTHS

Webject

```
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE_VALUE" data="ADDRESS"/>
  <ie:param name="LEADER" data = "_"/>
  <ie:param name="WIDTH" data="20"/>
  <ie:param name="WIDTH" data="20"/>
  <ie:param name="WIDTH" data="20"/>
  <ie:param name="SIZE" data="6"/>
</ie:webject>
</form>
```

Output



```
Sam Johnson_________1234 Main St._______sjohnson@somewhere.com
Harvy Anderson______1234 Amber St.______handerson@somewhere.com
James O'Connor______775 Main St.______________________________
Harvey Hampton______775 Main St.________hhampton@somewhere.com
```

Note

Each attribute is padded with the LEADER to meet or exceed the specified WIDTH.

Generated HTML

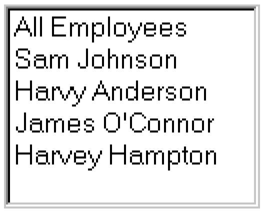
```
<form>
<select name='ADDRESS' size=6>
<option value='1234 Main St.'>Sam Johnson_________1234 Main
                      St._______sjohnson@somewhere.com</option>
<option value='1234 Amber St.'>Harvy Anderson______1234 Amber
                      St.______handerson@somewhere.com</option>
<option value='775 Main St.'>James O'Connor______775 Main
                       St.____________________________</option>
<option value='775 Main St.'>Harvey Hampton______775 Main
                     St.________hhampton@somewhere.com</option>
</select>
</form>
```

DISPLAY SINGLE ATTRIBUTE WITH CONSTANT VALUE

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE_VALUE" data="ADDRESS"/>
  <ie:param name="LIST_HEADER" data="All Employees"/>
  <ie:param name="LIST_HEADER_VALUE" data="*"/>
  <ie:param name="SIZE" data="6"/>
</ie:webject>
</form>
```

Output



Generated HTML

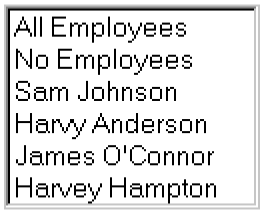
```
<form>
<select name='ADDRESS' size=6>
<option value='*'>All Employees</option>
<option value='1234 Main St.'>Sam Johnson</option>
<option value='1234 Amber St.'>Harvy Anderson</option>
<option value='775 Main St.'>James O'Connor</option>
<option value='775 Main St.'>Harvey Hampton</option>
</select>
</form>
```

DISPLAY SINGLE ATTRIBUTE WITH MULTIPLE CONSTANT VALUES

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE_VALUE" data="ADDRESS"/>
  <ie:param name="LIST_HEADER" data="All Employees,No
                                         Employees" delim=","/>
  <ie:param name="LIST_HEADER_VALUE" data="*,-" delim=","/>
  <ie:param name="SIZE" data="6"/>
</ie:webject>
</form>
```

Output



Generated HTML

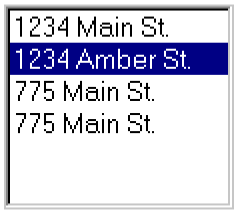
```
<form>
<select name='ADDRESS' size=6>
<option value='*'>All Employees</option>
<option value='-'>No Employees</option>
<option value='1234 Main St.'>Sam Johnson</option>
<option value='1234 Amber St.'>Harvy Anderson</option>
<option value='775 Main St.'>James O'Connor</option>
<option value='775 Main St.'>Harvey Hampton</option>
</select>
</form>
```

DISPLAY SINGLE ATTRIBUTE WITH PRE-SELECTED OPTION

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="ADDRESS"/>
  <ie:param name="SIZE" data="6"/>
  <ie:param name="SELECTED" data="1234 Amber ST."/>
</ie:webject>
</form>
```

Output



```
1234 Main St.
1234 Amber St.
775 Main St.
775 Main St.
```

Generated HTML

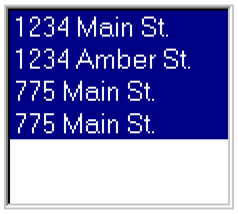
```
<form>
<select name='ADDRESS' size=6>
<option value='1234 Main St.'>1234 Main St.</option>
<option selected value='1234 Amber St.'>1234 Amber St.</option>
<option value='775 Main St.'>775 Main St.</option>
<option value='775 Main St.'>775 Main St.</option>
</select>
</form>
```

DISPLAY SINGLE ATTRIBUTE WITH ALL OPTIONS PRE-SELECTED

Webject

```
<form>
<ie:webject name="Display-Selection" type="DSP">
  <ie:param name="ATTRIBUTE" data="ADDRESS"/>
  <ie:param name="SIZE" data="6"/>
  <ie:param name="SELECTED" data="*"/>
  <ie:param name="MULTIPLE" data="TRUE"/>
</ie:webject>
</form>
```

Output



```
1234 Main St.
1234 Amber St.
775 Main St.
775 Main St.
```

Generated HTML

```
<form>
<select name='ADDRESS' size=6 multiple>
<option selected value='1234 Main St.'>1234 Main St.</option>
```

```
        <option selected value='1234 Amber St.'>1234 Amber St.</option>
        <option selected value='775 Main St.'>775 Main St.</option>
        <option selected value='775 Main St.'>775 Main St.</option>
        </select>
        </form>
```

Display-Table

DESCRIPTION

Displays an Info*Engine group as an HTML table.

SYNTAX

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="ATTRIBUTE" data="attribute"/>
  <ie:param name="BORDER" data="[pixels | 1]"/>
  <ie:param name="FOOTER" data="anytext"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="HEADER" data="[header_text | NONE]"/>
  <ie:param name="HYPER_VALUE" data="attribute_value"/>
  <ie:param name="HYPERLINK" data="[attribute | NONE]"/>
  <ie:param name="MAX" data="maximum"/>
  <ie:param name="PATH_INFO" data="attribute_value"/>
  <ie:param name="START" data="starting_element"/>
  <ie:param name="TEMPLATE_URL" data="url"/>
  <ie:param name="WIDTH" data="[character_width | 11]"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	HYPERLINK	ATTRIBUTE
	HYPER_VALUE	BORDER
	PATH_INFO	FOOTER
	TEMPLATE_URL	GROUP_IN
		HEADER
		WIDTH
		MAX
		START

ATTRIBUTE

Specifies the name of the database attributes you want to display. The HEADER parameter takes precedence over ATTRIBUTE when determining the header titles displayed above each column. The default for this parameter is to display all attributes contained in the first element of the GROUP_IN parameter. If the first element doesn't contain the desired column, multiple ATTRIBUTE parameters must be used to specify the columns to display. Multiple values can be specified for this parameter. This parameter is optional.

BORDER

Determines the width, in pixels, of the HTML table border around the entire group of information to be displayed. If BORDER is specified, lines and rows are created. If BORDER is specified with a value of `""`, no border is drawn. If BORDER is not specified, the default value is 1. This parameter is optional.

FOOTER

Specifies the information to be displayed at the bottom of the table. If specified, the value of this parameter can be `anytext`. The default behavior for this parameter is that no footer is displayed. This parameter is optional.

GROUP_IN

Identifies the name of the group to be used as an input source. The group can be a VDB group or a Context group. For further information about groups, see Info*Engine Data Management on page 31.

The default for this parameter is to use the last group defined in the VDB. This parameter is optional.

HEADER

Specifies the header title or list of header titles to be displayed above each column. This parameter takes precedence over the attribute name in determining the header titles. If HEADER is set to NONE, then no headers are displayed.

While multiple values can be specified for this parameter, the number of HEADER values must match the number of ATTRIBUTE values. This parameter is optional.

HYPER_VALUE

Specifies the attributes used to generate CGI-data for the hyperlinks that are generated for the attributes specified by the HYPERLINK parameter. All the attributes are added to all the hyperlink values generated. Multiple values can be specified. If no values are specified, the default is to use the attributes specified by the HYPERLINK parameter. Specifying an attribute causes a CGI-data string to be added to the generated hyperlink of the form `<attribute_name>=<attribute_value>`. HYPERLINK must be specified in order to use this parameter.

HYPERLINK

Specifies a list of attributes that should have hyperlinks generated for them. If any HYPERLINK parameters are specified, you must specify a TEMPLATE_URL for each attribute specified by this parameter.

The default for this parameter is that no hyperlinks are generated. Multiple values can be specified for this parameter.

MAX

Defines the maximum number of elements to display. Specifying a value for this parameter allows control over how much data is returned through the webject.

By using parameter value substitution for the MAX and START parameters, a series of webjects can be coded that allow the data in any group to be displayed in manageable result sets. For more information, see Dynamic Parameter Value Substitution on page 44.

The default for this parameter is to display all remaining elements in the group, starting with the element identified in the START parameter. This parameter is optional.

PATH_INFO

Specifies additional path information to insert at the end of a URL specified with the TEMPLATE_URL parameter. This string must be manually CGI-encoded to prevent the generation of invalid URLs. If no values are specified, no additional text is added to the generated URLs. The number of values specified for this parameter must equal the number of attributes specified with the HYPERLINK parameter. An empty string can be specified. The values of this parameter are associated with the HYPERLINK parameter values in the order encountered. HYPERLINK must be specified in order to use this parameter. Multiple values can be specified for this parameter.

START

Specifies the number of the first element in the group to display. Use this parameter in conjunction with the MAX parameter to control how much data is returned through the webject.

By using parameter value substitution for the MAX and START parameters, a series of webjects can be coded that allow the data in any group to be displayed in manageable result sets. For more information, see Dynamic Parameter Value Substitution on page 44.

The default for this parameter is to start with the first record in the group. This parameter is optional.

TEMPLATE_URL

Specifies the base URL or list of base URLs to be used when generating hyperlinks for the attributes specified with the HYPERLINK parameter. There must be a URL for every attribute specified with the HYPERLINK parameter. The values of this parameter are associated with the HYPERLINK parameter values in the order encountered. If a TEMPLATE_URL parameter value contains the string “`mailto:`”, then the associated HYPERLINK parameter value is assumed to be an email address, and an HTML `mailto` URL is constructed.

HYPERLINK must be specified in order to use this parameter. Multiple values can be specified for this parameter.

WIDTH

Specifies a list of the widths of each attribute column. Unless you use a pre-formatted text table, HTML does not allow a length longer than the longest data. The default value of this optional parameter is 11 characters. Multiple values can be specified for this parameter. This parameter is optional.

EXAMPLES

The following Display-Table webject examples assume an input group containing multiple elements; each with a name, address and email attribute value:

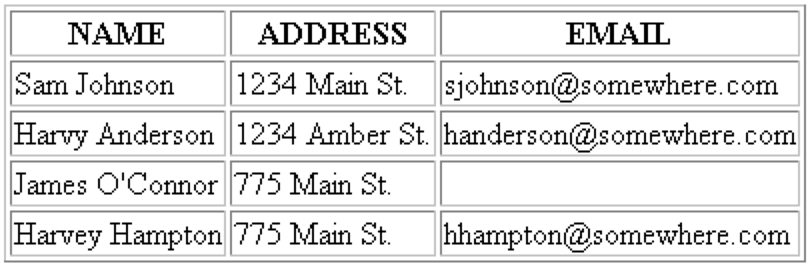
DEFAULT DISPLAY

In the following example, all of the information is displayed using the default values for the table:

Webject

```
<ie:webject name="Display-Table" type="DSP"/>
```

Output



NAME	ADDRESS	EMAIL
Sam Johnson	1234 Main St.	sjohnson@somewhere.com
Harvy Anderson	1234 Amber St.	handerson@somewhere.com
James O'Connor	775 Main St.	
Harvey Hampton	775 Main St.	hhampton@somewhere.com

Generated HTML

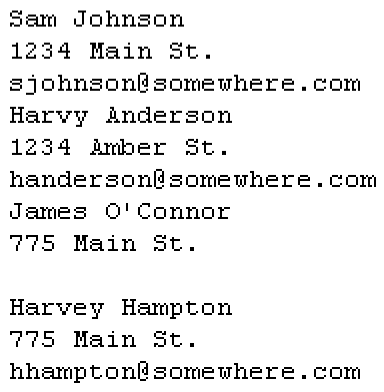
```
<table border=1>
  <tr><th>NAME </th><th>ADDRESS </th><th>EMAIL</th></tr>
  <tr><td>Sam Johnson</td><td>1234 Main
                   St.</td><td>sjohnson@somewhere.com</td></tr>
  <tr><td>Harvy Anderson</td><td>1234 Amber
                  St.</td><td>handerson@somewhere.com</td></tr>
  <tr><td>James O'Connor</td><td>775 Main
                                   St.</td><td>&nbsp;</td></tr>
  <tr><td>Harvey Hampton</td><td>775 Main
                   St.</td><td>hhampton@somewhere.com</td></tr>
</table>
```

NO BORDERS, NO HEADERS

Webject

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="BORDER" data=""/>
  <ie:param name="HEADER" data="none"/>
</ie:webject>
```

Output



```
Sam Johnson
1234 Main St.
sjohnson@somewhere.com
Harvy Anderson
1234 Amber St.
handerson@somewhere.com
James O'Connor
775 Main St.

Harvey Hampton
775 Main St.
hhampton@somewhere.com
```

Generated HTML

```
<pre>
Sam Johnson
1234 Main St.
sjohnson@somewhere.com
Harvy Anderson
1234 Amber St.
handerson@somewhere.com
James O'Connor
775 Main St.
&nbsp;
Harvey Hampton
775 Main St.
hhampton@somewhere.com
</pre>
```

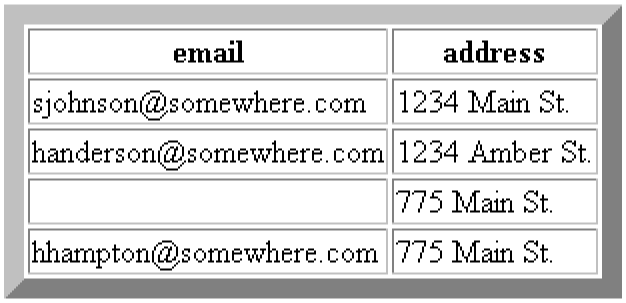
MULTIPLE COLUMNS

The following example shows a table with multiple columns. The ATTRIBUTE parameter specifies the information to display with a comma serving as the delimiter. The BORDER parameter overrides the default value.

Webject

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="BORDER" data="10"/>
  <ie:param name="ATTRIBUTE" data="email,address" delim=","/>
</ie:webject>
```

Output



email	address
sjohnson@somewhere.com	1234 Main St.
handerson@somewhere.com	1234 Amber St.
	775 Main St.
hhampton@somewhere.com	775 Main St.

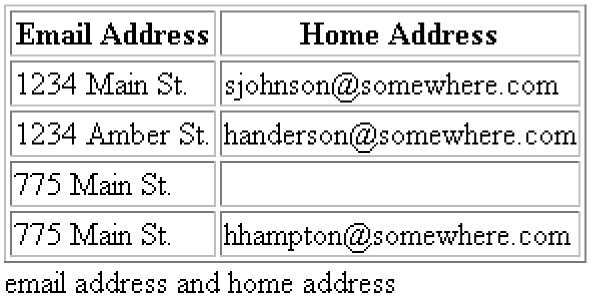
MULTIPLE COLUMNS WITH ALTERNATE NAMES

The following example shows the header with a different name. The WIDTH parameter overrides the default. A comma is used as a delimiter, and FOOTER text follows the table.

Webject

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="ATTRIBUTE" data="address,email" delim=","/>
  <ie:param name="HEADER"
            data="Email Address,Home Address" delim=","/>
  <ie:param name="WIDTH" data="20,20" delim=","/>
  <ie:param name="FOOTER" data="email address and home address"/>
</ie:webject>
```

Output



Email Address	Home Address
1234 Main St.	sjohnson@somewhere.com
1234 Amber St.	handerson@somewhere.com
775 Main St.	
775 Main St.	hhampton@somewhere.com

email address and home address

HYPERLINKED COLUMNS

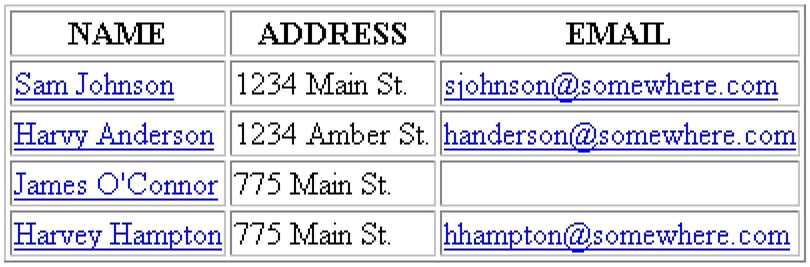
In the following example, the HYPERLINK parameter identifies the columns that contain hyperlinks. The WIDTH parameter overrides the default.

Webject

```
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="HYPERLINK" data="email"/>
  <ie:param name="HYPERLINK" data="name"/>
  <ie:param name="TEMPLATE_URL"
                            data="mailto:abc@somecompany.com"/>
  <ie:param name="TEMPLATE_URL"
```

```
                       data="http://machine/somecompany.html"/>
  <ie:param name="PATH_INFO" data=""/>
  <ie:param name="PATH_INFO" data="class=employees"/>
  <ie:param name="HYPER_VALUE" data="Name,Address,Email"
                                                    delim=","/>
</ie:webject>
```

Output



NAME	ADDRESS	EMAIL
Sam Johnson	1234 Main St.	sjohnson@somewhere.com
Harvy Anderson	1234 Amber St.	handerson@somewhere.com
James O'Connor	775 Main St.	
Harvey Hampton	775 Main St.	hhampton@somewhere.com

Display-Value

DESCRIPTION

Displays the value of an attribute, or set of attributes, of an Info*Engine group as HTML elements.

SYNTAX

```
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="attributes"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="NAME" data="name"/>
  <ie:param name="POST_TABLE_TEXT" data="text"/>
  <ie:param name="PRE_TABLE_TEXT" data="text"/>
  <ie:param name="PROMPT" data="somecomment"/>
  <ie:param name="SIZE" data="[columnsXrows
                                 |windowsize/maxsize | size]"/>
  <ie:param name="TYPE" data="[HTML | RADIO | FILE | PLAIN |
              TEXT | TEXTAREA| HIDDEN | CHECKBOX | PASSWORD]"/>
  <ie:param name="UNDEFINED" data="somevalue"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
ATTRIBUTE	NAME	GROUP_IN
	PROMPT	PRE_TABLE_TEXT

Required	Select	Optional
	SIZE	POST_TABLE_TEXT
	TYPE	UNDEFINED

ATTRIBUTE

Identifies the name of the database attributes to display. The order in which the attributes are specified determines the order in which they are displayed. Refer to the table in the TYPE parameter description for the TYPE values with which this parameter can be used. Multiple values can be specified for this parameter. This parameter is required.

GROUP_IN

Identifies the name of the group to be used as an input source. The group can be a VDB group or a Context group. For further information about groups, see Info*Engine Data Management on page 31.

The default for this parameter is to use the last group defined in the VDB. This parameter is optional.

NAME

Assigns a form name to the generated FORM element. You can refer to this name in the next JSP page to supply a value (such as within a WHERE clause). The default for this parameter is the attribute name of the database value displayed. Refer to the table in the TYPE parameter description for the TYPE values with which this parameter can be used.

POST_TABLE_TEXT

Specifies the text or HTML generated after each group of objects. The default for this parameter is `""`. This parameter is optional.

PRE_TABLE_TEXT

Specifies the text or HTML generated before each group of objects. The default for this parameter is `""`. This parameter is optional.

PROMPT

Specifies a descriptive string of text that you want to display to the left of the attribute value. The default for this parameter is the attribute name. Multiple values can be specified for this parameter. Refer to the table in the TYPE parameter description for the TYPE values with which this parameter can be used.

SIZE

Sets the length of the text field used to display the attribute value.

If the SIZE parameter is used to specify the size of a TEXTAREA in the TYPE parameter, then the value of the SIZE parameter must be stated using the format `data="`*`columnsXrows`*`"` (for example, `data="10x4"`).

If you use the SIZE parameter to limit the maximum length of a text input field, such a TEXT or PASSWORD in the TYPE parameter, then you must specify the value using the format `data="`*`windowsize/maximumsize`*`"`, where the *`windowsize`* is the maximum number of characters to display for that element and the *`maximumsize`* is the maximum number of characters to accept for that element. For example, assume that your *`windowsize`* is 10 and your *`maximumsize`* is 20; the resulting parameter would be:

```
<ie:param name="SIZE" data="10/20"/>
```

Refer to the table in the TYPE parameter description for the TYPE values with which this parameter can be used. Multiple values can be specified for this parameter. The default for this parameter is 11 characters.

TYPE

Specifies how to display the value of the ATTRIBUTE parameter. Valid values are HTML, PLAIN, TEXT, TEXTAREA, HIDDEN, and PASSWORD:

- CHECKBOX – Displays data with a checkbox to the left. Must be used in conjunction with a form.
- FILE – Allows the user to select files in order to submit their contents with a form. Must be used in conjunction with a form.
- HIDDEN – Creates a hidden form element which displays nothing, but passes the value of the attribute. Must be used in conjunction with a form.
- HTML – Encodes an output string in HTML.
- PASSWORD – Similar to TEXT. However, what is typed is not visible to the user. Must be used in conjunction with a form.
- PLAIN – Displays an attribute value as text that cannot be edited.
- RADIO – Displays data with a radio button to the left. Radio buttons work like checkboxes; however radio buttons may be mutually exclusive. Must be used in conjunction with a form.
- TEXT – Displays an attribute value in a text box and is editable. Must be used in conjunction with a form.
- TEXTAREA – Displays multiple rows. However, the number of rows and columns must be specified in the SIZE parameter. TEXTAREA must be used in conjunction with a form.

The default for this optional parameter is PLAIN.

The following table shows which parameters are usable depending on the TYPE:

	ATTRIBUTE	NAME	PROMPT	SIZE
CHECKBOX	X	X		
FILE		X		
HIDDEN	X	X		
HTML			X	
PASSWORD	X	X	X	X
RADIO	X	X		

	ATTRIBUTE	NAME	PROMPT	SIZE
TEXT	X	X	X	X
TEXTAREA		X	X	X

UNDEFINED

Sets the value to display for an undefined value. An undefined value is either a nonexistent attribute or an attribute that has a null value. An attribute containing an empty string ("") is not interpreted as being undefined. The default for this parameter is "". This parameter is optional.

EXAMPLES

The following Display-Value webject examples assume an input group containing multiple elements; each with a name, address and email attribute value:

Note

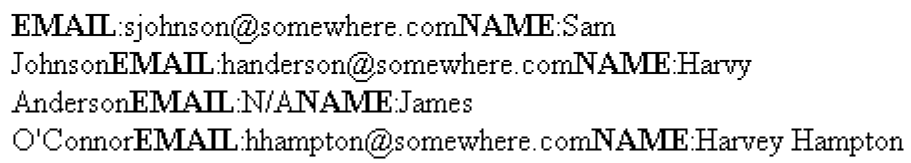
The FORM element is not automatically generated by this webject. To display the lists correctly, include the `<FORM>` and `</FORM>` tags as shown in the examples.

DEFAULT DISPLAY OF MULTIPLE ATTRIBUTES

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="ATTRIBUTE" data="EMAIL,NAME" delim=","/>
  <ie:param name="UNDEFINED" data="N/A"/>
</ie:webject>
</form>
```

Output



EMAIL:sjohnson@somewhere.com**NAME**:Sam Johnson**EMAIL**:handerson@somewhere.com**NAME**:Harvy Anderson**EMAIL**:N/A**NAME**:James O'Connor**EMAIL**:hhampton@somewhere.com**NAME**:Harvey Hampton

Generated HTML

```
<form>
<b>EMAIL</b>:sjohnson@somewhere.com<b>NAME</b>:Sam Johnson
<b>EMAIL</b>:handerson@somewhere.com<b>NAME</b>:Harvy Anderson
<b>EMAIL</b>:N/A<b>NAME</b>:James O'Connor
<b>EMAIL</b>:hhampton@somewhere.com<b>NAME</b>:Harvey Hampton
</form>
```

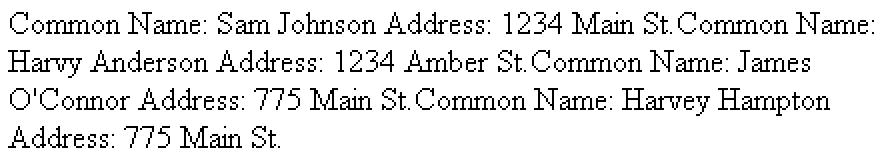
MULTIPLE ATTRIBUTES WITH PROMPTS AND NON-DEFAULT NAMES

The following example displays the NAME and ADDRESS attributes after their PROMPT value.

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim=","/>
  <ie:param name="PROMPT"
                   data="Common Name: , Address: " delim=","/>
  <ie:param name="TYPE" data="PLAIN"/>
</ie:webject>
</form>
```

Output



Common Name: Sam Johnson Address: 1234 Main St.Common Name: Harvy Anderson Address: 1234 Amber St.Common Name: James O'Connor Address: 775 Main St.Common Name: Harvey Hampton Address: 775 Main St.

Generated HTML

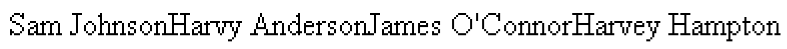
```
<form>
<pre>
NamePrompt <input type="text" name="NameField" value="Sam
            Johnson" size="20">AddressPrompt <input type="text"
           name="AddressField" value="1234 Main St." size="20">
NamePrompt <input type="text" name="NameField" value="Harvy
           Anderson" size="20">AddressPrompt <input type="text"
          name="AddressField" value="1234 Amber St." size="20">
NamePrompt <input type="text" name="NameField" value="James
           O'Connor" size="20">AddressPrompt <input type="text"
            name="AddressField" value="775 Main St." size="20">
NamePrompt <input type="text" name="NameField" value="Harvey
            Hampton" size="20">AddressPrompt <input type="text"
            name="AddressField" value="775 Main St." size="20">
</pre>
</form>
```

SINGLE ATTRIBUTE, TYPE=PLAIN

Webject

```
<form>
ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="PROMPT" data=""/>
  <ie:param name="TYPE" data="PLAIN"/>
</ie:webject>
</form>
```

Output



Sam JohnsonHarvy AndersonJames O'ConnorHarvey Hampton

Generated HTML

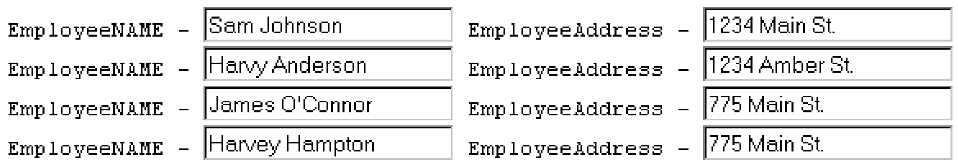
```
<form>
Sam JohnsonHarvy AndersonJames O'ConnorHarvey Hampton
</form>
```

MULTIPLE ATTRIBUTES WITH PROMPTS, DEFAULT NAMES, AND TYPE=TEXT

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim=","/>
  <ie:param name="PROMPT" data="EmployeNAME - ,
                              EmployeeAddress - " delim=","/>
  <ie:param name="TYPE" data="text"/>
  <ie:param name="SIZE" data="20"/>
  <ie:param name="CELLPADDING"  data="5"/>
</ie:webject>
</form>
```

Output



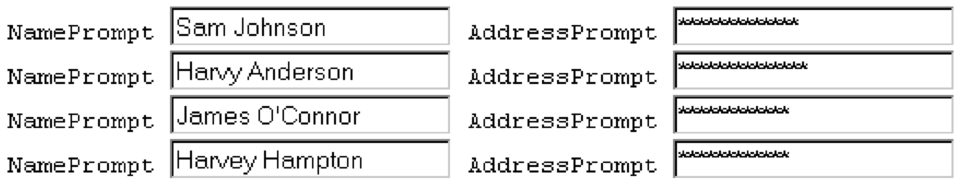
```
EmployeeNAME - Sam Johnson       EmployeeAddress - 1234 Main St.
EmployeeNAME - Harvy Anderson    EmployeeAddress - 1234 Amber St.
EmployeeNAME - James O'Connor    EmployeeAddress - 775 Main St.
EmployeeNAME - Harvey Hampton    EmployeeAddress - 775 Main St.
```

MULTIPLE ATTRIBUTES WITH PROMPTS, SPECIFIED NAMES, AND TYPE=TEXT

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim=","/>
  <ie:param name="NAME" data="NameField,
                                    AddressField" delim=","/>
  <ie:param name="PROMPT" data="NamePrompt ,
                                  AddressPrompt " delim=","/>
  <ie:param name="TYPE" data="text"/>
  <ie:param name="SIZE" data="20/20"/>
  <ie:param name="CELLPADDING"  data="5"/>
</ie:webject>
<form>
```

Output



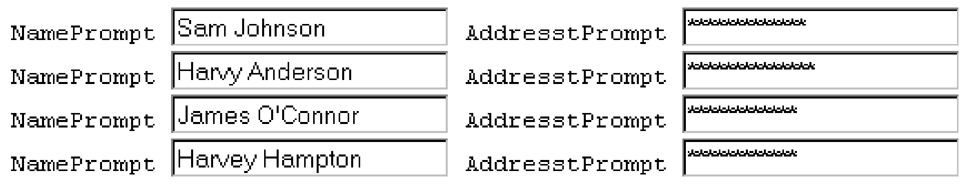
```
NamePrompt Sam Johnson       AddressPrompt *************
NamePrompt Harvy Anderson    AddressPrompt **************
NamePrompt James O'Connor    AddressPrompt ************
NamePrompt Harvey Hampton    AddressPrompt ************
```

MULTIPLE ATTRIBUTES WITH PROMPTS, SPECIFIED NAMES, AND MULTIPLE TYPE VALUES

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim=","/>
  <ie:param name="NAME" data="NameField,AddressField"
                                                  delim=","/>
  <ie:param name="PROMPT" data="NamePrompt , AddresstPrompt "
                                                  delim=","/>
  <ie:param name="TYPE" data="text,password" delim=","/>
  <ie:param name="SIZE" data="20/20,20/20"/>
  <ie:param name="CELLPADDING"  data="5"/>
</ie:webject>
</form>
```

Output



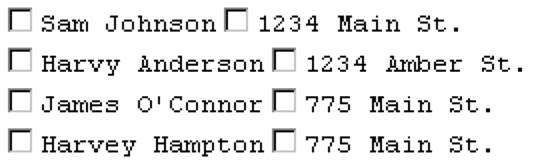
```
NamePrompt  Sam Johnson       AddresstPrompt  *************
NamePrompt  Harvy Anderson    AddresstPrompt  **************
NamePrompt  James O'Connor    AddresstPrompt  ************
NamePrompt  Harvey Hampton    AddresstPrompt  ************
```

MULTIPLE ATTRIBUTES WITH PROMPTS AND SPECIFIED NAMES, TYPE=CHECKBOX

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim=","/>
  <ie:param name="NAME" data="NameField,DepartmentField"
                                                  delim=","/>
  <ie:param name="PROMPT" data="NamePrompt , DepartmentPrompt "
                                                  delim=","/>
  <ie:param name="TYPE" data="checkbox"/>
  <ie:param name="SIZE" data="20"/>
  <ie:param name="CELLPADDING"  data="5"/>
</ie:webject>
</form>
```

Output



```
☐ Sam Johnson ☐ 1234 Main St.
☐ Harvy Anderson ☐ 1234 Amber St.
☐ James O'Connor ☐ 775 Main St.
☐ Harvey Hampton ☐ 775 Main St.
```

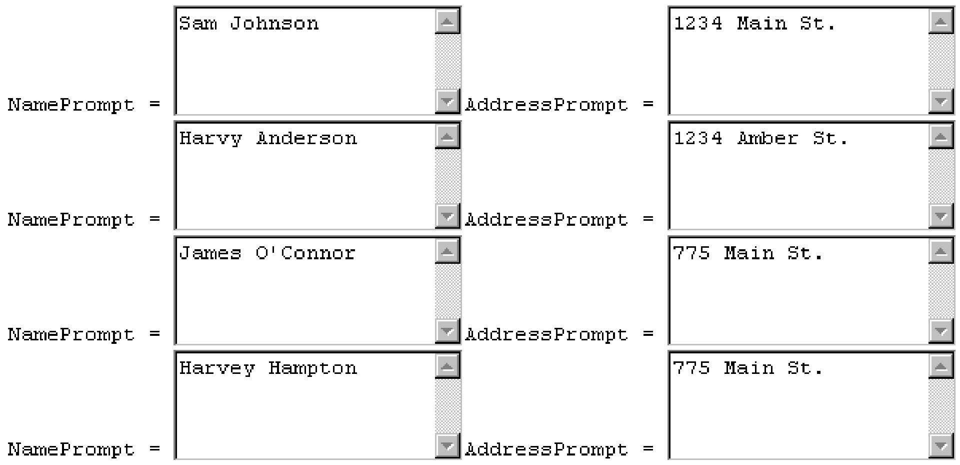
MULTIPLE ATTRIBUTES WITH PROMPTS, TYPE=TEXTAREA

Webject

```
<form>
<ie:webject name="Display-Value" type="DSP">
  <ie:param name="ATTRIBUTE" data="NAME"/>
```

```
    <ie:param name="ATTRIBUTE" data="ADDRESS"/>
    <ie:param name="NAME" data="NameField,AddressField"
                                                  delim=","/>
    <ie:param name="PROMPT" data="NamePrompt = ,AddressPrompt = "
                                                  delim=","/>
    <ie:param name="TYPE" data="TEXTAREA"/>
    <ie:param name="SIZE" data="20x4"/>
    <ie:param name="CELLPADDING"  data="5"/>
  </ie:webject>
  </form>
```

Output



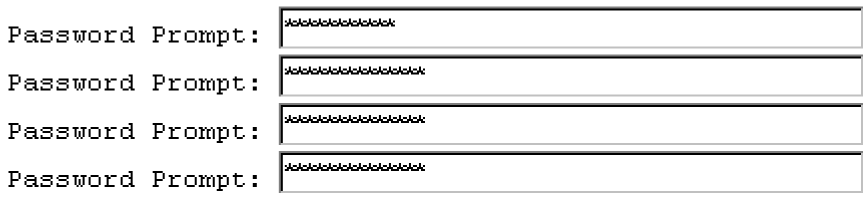
```
NamePrompt = Sam Johnson      AddressPrompt = 1234 Main St.
NamePrompt = Harvy Anderson   AddressPrompt = 1234 Amber St.
NamePrompt = James O'Connor   AddressPrompt = 775 Main St.
NamePrompt = Harvey Hampton   AddressPrompt = 775 Main St.
```

SINGLE ATTRIBUTE WITH PROMPT, TYPE=PASSWORD

Webject

```
  <form>
  <ie:webject name="Display-Value" type="DSP">
    <ie:param name="ATTRIBUTE" data="NAME"/>
    <ie:param name="TYPE" data="password"/>
    <ie:param name="PROMPT" data="Password Prompt: "/>
    <ie:param name="SIZE" data="40"/>
  </ie:webject>
  </form>
```

Output



```
Password Prompt: ************
Password Prompt: ***************
Password Prompt: ***************
Password Prompt: ***************
```

Display-XML

DESCRIPTION

Displays an XML formatted page of the specified group that is currently in the VDB. The webject provides an alias for a direct call to the task processor. The output of the webject is identical to using the URL:

`http://machine/Windchill/servlet/IE/tasks/infoengine/taskname`

This URL assumes that the Info*Engine application URL task prefix is `/Windchill/servlet/IE/tasks`.

Note

Do not include any HTML tags on the JSP page containing this webject, and include only one occurrence of the webject on a page, as additional output from the JSP page corrupts the XML syntax.

SYNTAX

```
<ie:webject name="Display-XML" type="DSP">
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="MODE" data="[FULL | BRIEF]"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		GROUP_IN
		MODE

GROUP_IN

Identifies the name of the group to be used as an input source. The group can be a VDB group or a Context group. For further information about groups, see Info*Engine Data Management on page 31.

Multiple values can be specified for this parameter. The default for this parameter is to use all groups in the VDB. This parameter is optional.

MODE

Specifies whether metadata is included in the generated XML stream. Specifying FULL includes metadata at all levels of the XML stream.

The default for this parameter is BRIEF. This parameter is optional.

EXAMPLES

The following Display-XML webject example assume an input group containing multiple elements; each with a name, address and email attribute value:

For an example that shows the XML output using MODE set to FULL, see the Set-Metadata webject on page 393.

Webject

```
<%@page language="java"
  session="false"
  errorPage="../IEError.jsp"%>
<%@ taglib uri=http://www.ptc.com/infoengine/taglib/core
                                               prefix="ie" %>
<ie:task uri="com/company/CreateGroup.xml"/>
<ie:webject name="Display-XML" type="DSP"/>
```

Generated XML

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="createdgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL>hhampton@somewhere.com</EMAIL>
  </wc:INSTANCE>
</EmployeeData>
</wc:COLLECTION>
```

Echo-Request

DESCRIPTION

Displays the values and metadata of the current groups in an Info*Engine request, in HTML format. All VDB groups and context groups of an Info*Engine request are formatted.

This webject can also be used to verify that the Info*Engine server is functioning properly without connecting to any adapter.

SYNTAX

```
<ie:webject name="Echo-Request" type="DSP"/>
```

PARAMETERS

No parameters are required because the Echo-Request webject does not query or display objects. Instead, it displays the webject parameters after processing, as well as the environment and the form variables. Webject parameters are processed as usual and their resulting values are displayed in a browser window on your screen.

EXAMPLES

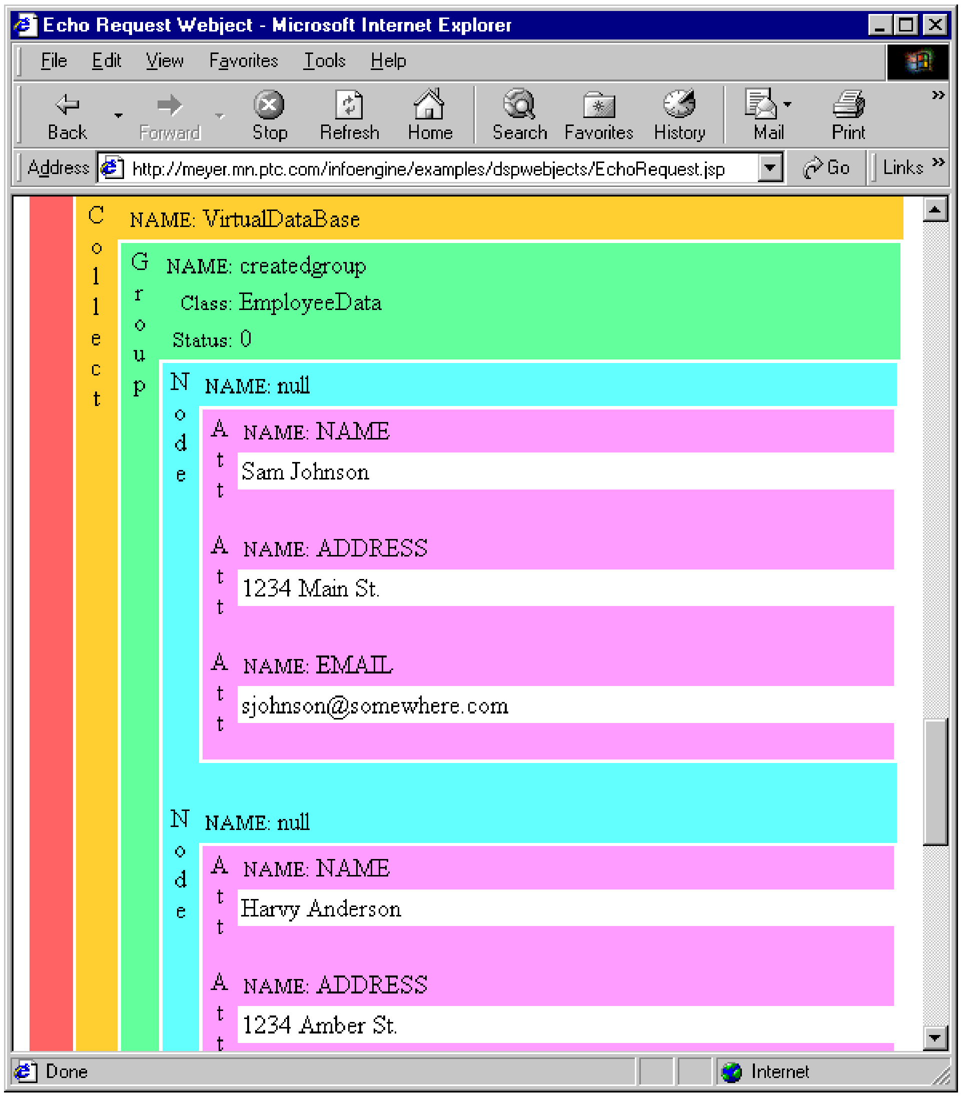
The following Echo-Request webject example assume an input group containing multiple elements; each with a name, address and email attribute value:

Webject

```
    <ie:task uri="/com/company/CreateGroup.xml"/>
    <ie:webject name="Echo-Request" type="DSP"/>
```

Output

The following example is a screen capture of a partial browser window showing the group created from the `com/company/CreateGroup.xml` task.


Echo Request Webject - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address http://meyer.mn.ptc.com/infoengine/examples/dspwebjects/EchoRequest.jsp Go Links
Done Internet

```
Collect  NAME: VirtualDataBase
 Group  NAME: createdgroup
          Class: EmployeeData
        Status: 0
  Node  NAME: null
   Att  NAME: NAME
        Sam Johnson
   Att  NAME: ADDRESS
        1234 Main St.
   Att  NAME: EMAIL
        sjohnson@somewhere.com
  Node  NAME: null
   Att  NAME: NAME
        Harvy Anderson
   Att  NAME: ADDRESS
        1234 Amber St.
```

Image Webjects for JPEG

The following webjects can be used to display images in a JPEG format:

- Bar-Graph on page 345
- Line-Graph on page 348
- Pie-Chart on page 352

Bar-Graph

DESCRIPTION

Takes as input an Info*Engine group containing numeric data to be graphed and produces a JPEG image as output.

SYNTAX

```
<ie:webject name="Bar-Graph" type="IMG">
  <ie:param name="BACKDROP_COLOR " data="html_color"/>
  <ie:param name="BACKGROUND_COLOR " data="html_color"/>
  <ie:param name="COLOR " data="html_color"/>
  <ie:param name="DECIMAL_PLACES" data="integer_places"/>
  <ie:param name="GRAPH" data="column_name"/>
  <ie:param name="GROUP_IN" data="input_groupname"/>
  <ie:param name="HEIGHT" data="integer_height"/>
  <ie:param name="INSET_BOTTOM" data="integer_inset"/>
  <ie:param name="INSET_LEFT" data="integer_inset"/>
  <ie:param name="INSET_RIGHT" data="integer_inset"/>
  <ie:param name="INSET_TOP" data="integer_inset"/>
  <ie:param name="INSETS" data="integer_inset"/>
  <ie:param name="LABEL" data="column_name"/>
  <ie:param name="LABEL_ANGLE" data="integer_angle"/>
  <ie:param name="SCALE_INCREMENT" data="double"/>
  <ie:param name="TEXT_COLOR " data="html_color"/>
  <ie:param name="WIDTH" data="integer_width"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GRAPH		BACKDROP_COLOR
GROUP_IN		BACKGROUND_COLOR
		COLOR
		DECIMAL_PLACES
		HEIGHT
		INSET_BOTTOM
		INSET_LEFT
		INSET_RIGHT
		INSET_TOP
		INSETS
		LABEL
		LABEL_ANGLE
		SCALE_INCREMENT

Required	Select	Optional
		TEXT_COLOR
		WIDTH

BACKDROP_COLOR

Specifies the color, in #RRGGBB format, of backdrop in the generated image. The default value is `lightGray` (#868686).

BACKGROUND_COLOR

Specifies the color, in #RRGGBB format, of the background of the generated image. The default value is white.

COLOR

Specifies the color, in #RRGGBB format, that the generated bars should be rendered in. This parameter may be multi-valued. If it is multi-valued, then the colors specified are used in order, and all values are iterated over until all data in the input group has been graphed. For example, specifying two color parameters as the following:

```
<ie:param name="COLOR" data="#FF0000,#0000FF" delim=","/>
```

causes alternating red and blue.

Optionally each input group can contain a column named “COLOR” to specify the color to use for each data point in the input group. The COLOR parameter overrides data contained in the input group.

The default value is #0000C8.

DECIMAL_PLACES

Specifies an integer that governs how many decimal places are displayed on scales. This parameter is ignored if no value is given for SCALE_ INCREMENT. The default value is 0.

GRAPH

Specifies the name of the column containing the data to be graphed.

GROUP_IN

Specifies the name of the input group containing data to be graphed.

HEIGHT

Specifies the integer height of the generated image. The default value is 400.

INSET_BOTTOM

Specifies an integer value to use as insets to pad the bottom of the image. The default value is 10.

INSET_LEFT

Specifies an integer value to use as insets to pad the left side of the image. The default value is 10.

INSET_RIGHT

Specifies an integer value to use as insets to pad the right side of the image. The default value is 10.

INSET_TOP

Specifies an integer value to use as insets to pad the top of the image. The default value is 10.

INSETS

Specifies an integer value to use as insets to pad the outside of the image. If this parameter is specified it overrides the presence of INSET_LEFT, INSET_RIGHT, INSET_TOP and INSET_BOTTOM. The default value is 10.

LABEL

Specifies the name of the column containing strings used to label each bar generated. The default is not to display labels.

LABEL_ANGLE

Specifies the integer angle at which to display the labels. This can be used to avoid overlapping labels when one or more are particularly long. The default value is 0.

SCALE_INCREMENT

Specifies a floating point number governing how to display a scale on the Y axis. If specified, scales start at 0 and appear every SCALE_INCREMENT until the largest data point in the data to be graphed. The default is to not display a scale.

TEXT_COLOR

Specifies the color, in #RRGGBB format, of text placed in the generated image. The default value is black.

WIDTH

Specifies the integer width of the generated image. The default value is 600.

EXAMPLE

Given the following input data (in GROUP_IN named “Employees”):

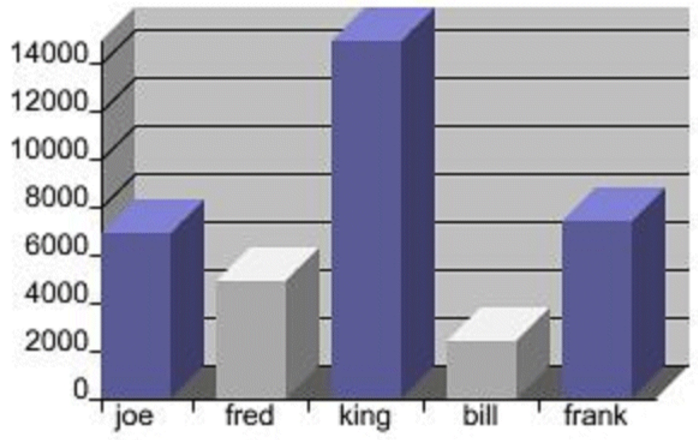
Name	Sales
joe	7000
fred	5000
king	15000
bill	2500
frank	7500

and the following JSP:

```
<%@page language="java" session="true"
```

```
contentType="image/jpeg"
%><%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><ie:webject name="Bar-Graph" type="IMG">
    <ie:param name="GROUP_IN" data="Employees"/>
    <ie:param name="COLOR" data="#5B5B97,#AAAAAA" delim=","/>
    <ie:param name="GRAPH" data="SAL"/>
    <ie:param name="WIDTH" data="300"/>
    <ie:param name="HEIGHT" data="200"/>
    <ie:param name="SCALE_INCREMENT" data="2000"/>
    <ie:param name="LABEL" data="NAME"/>
    <ie:param name="INSET_LEFT" data="20"/>
  </ie:webject>
```

Results in the following image:



Line-Graph

DESCRIPTION

Takes as input an Info*Engine group containing numeric data to be graphed and produces a JPEG image as output.

SYNTAX

```
<ie:webject name="Line-Graph" type="IMG">
  <ie:param name="BACKDROP_COLOR " data="html_color"/>
  <ie:param name="BACKGROUND_COLOR " data="html_color"/>
  <ie:param name="COLOR " data="html_color"/>
  <ie:param name="DECIMAL_PLACES" data="integer"/>
  <ie:param name="GRAPH" data="column_name"/>
  <ie:param name="GROUP_IN" data="input_groupname"/>
  <ie:param name="HEIGHT" data="integer_height"/>
  <ie:param name="INSET_BOTTOM" data="integer_inset"/>
  <ie:param name="INSET_LEFT" data="integer_inset"/>
  <ie:param name="INSET_RIGHT" data="integer_inset"/>
  <ie:param name="INSET_TOP" data="integer_inset"/>
  <ie:param name="INSETS" data="integer_inset"/>
  <ie:param name="MAXY" data="double"/>
  <ie:param name="MINY" data="double"/>
  <ie:param name="TEXT_COLOR " data="html_color"/>
  <ie:param name="WIDTH" data="integer_width"/>
  <ie:param name="X" data="column_name"/>
  <ie:param name="XSCALE_INCREMENT" data="double"/>
  <ie:param name="YSCALE_INCREMENT" data="double"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GRAPH		BACKDROP_COLOR
GROUP_IN		BACKGROUND_COLOR
X		COLOR
		DECIMAL_PLACES
		HEIGHT
		INSET_BOTTOM
		INSET_LEFT
		INSET_RIGHT
		INSET_TOP
		INSETS
		MAXY
		MINY
		TEXT_COLOR
		WIDTH
		XSCALE_INCREMENT
		YSCALE_INCREMENT

BACKDROP_COLOR

Specifies the color, in #RRGGBB format, of the backdrop in the generated image. The default value is `lightGray` (#868686).

BACKGROUND_COLOR

Specifies the color, in #RRGGBB format, of the background of the generated image. The default value is white.

COLOR

Specifies the color, in #RRGGBB format, that the generated bars should be rendered in. This parameter may be multi-valued. If it is multi-valued, then the colors specified are used in order and all values are iterated over until all data in the input group has been graphed. For example, specifying two color parameters as the following:

```
<ie:param name="COLOR" data="#FF0000,#0000FF" delim=","/>
```

causes alternating red and blue.

The default value is #0000C8.

DECIMAL_PLACES

Specifies an integer that governs how many decimal places are displayed on scales. This parameter is ignored if no value is given for YSCALE_INCREMENT or XSCALE_INCREMENT. The default value is 0.

GRAPH

Specifies the name of the column(s) containing the data to be graphed. This parameter can be multi-valued

GROUP_IN

Specifies the name of the input group containing data to be graphed.

HEIGHT

Specifies the integer height of the generated image. The default value is 400.

INSET_BOTTOM

Specifies an integer value to use as insets to pad the bottom of the image. The default value is 10.

INSET_LEFT

Specifies an integer value to use as insets to pad the left side of the image. The default value is 10.

INSET_RIGHT

Specifies an integer value to use as insets to pad the right side of the image. The default value is 10.

INSET_TOP

Specifies an integer value to use as insets to pad the top of the image. The default value is 10.

INSETS

Specifies an integer value to use as insets to pad the outside of the image. If this parameter is specified it overrides the presence of INSET_LEFT, INSET_RIGHT, INSET_TOP and INSET_BOTTOM. The default value is 10.

MAXY

Specifies the largest value found in the data set on the Y axis.

MINY

Specifies the smallest value found in the data set on the Y axis.

TEXT_COLOR

Specifies the color, in #RRGGBB format, of text placed in the generated image. The default value is black.

WIDTH

Specifies the integer width of the generated image. The default value is 600.

X

Specifies the name of the column containing the data to be graphed on the X axis.

XSCALE_INCREMENT

Specifies a floating point number governing how to display a scale on the X axis. If specified, scales start at the smallest X value and appear every XSCALE_INCREMENT until the largest X value. The default is to not display a scale on the X axis.

YSCALE_INCREMENT

Specifies a floating point number governing how to display a scale on the Y axis. If specified, scales start at MINY or the smallest value found in the data set and appear every YSCALE_INCREMENT until MAXY or the largest data point found in the data set. The default is to not display a scale on the Y axis.

EXAMPLE

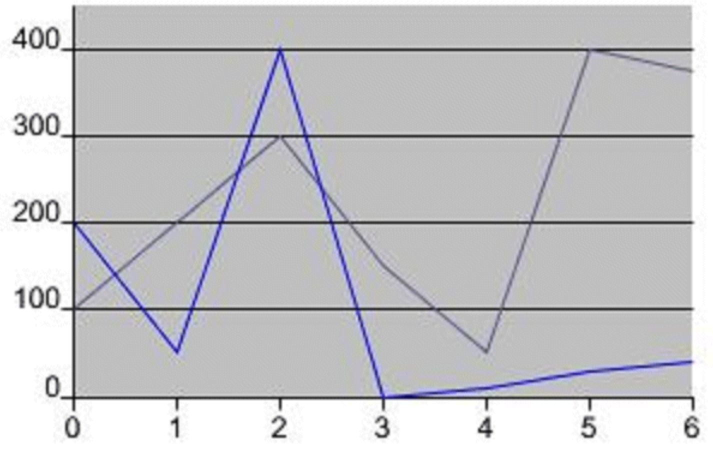
Given the following input data (in GROUP_IN value named “dataSet”):

X	G1	G2
0.0	100.0	200.0
1.0	200.0	50.0
2.0	300.0	400.0
3.0	150.0	0.0
4.0	50.0	10.0
5.0	400.0	30.0
6.0	375.0	40.0

and the following JSP:

```
<%@page language="java" session="true"
contentType="image/jpeg"
%><%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><ie:webject name="Line-Graph" type="IMG">
    <ie:param name="GROUP_IN" data="dataSet"/>
    <ie:param name="COLOR" data=" #5B5B97,#0000FF" delim=","/>
<ie:param name="X" data="X"/>
    <ie:param name="GRAPH" data="G1,G2" delim=","/>
    <ie:param name="WIDTH" data="300"/>    <ie:param name="HEIGHT"
data="200"/>
    <ie:param name="YSCALE_INCREMENT" data="100"/>
    <ie:param name="XSCALE_INCREMENT" data="1"/>
    <ie:param name="MINY" data="0"/>
    <ie:param name="MAXY" data="450"/>
    <ie:param name="INSET_LEFT" data="15"/>
  </ie:webject>
```

Results in the following image:



If titles and a legend are required, they should be generated programatically in the JSP that includes this image.

Pie-Chart

DESCRIPTION

Takes as input an Info*Engine group containing numeric data to be graphed and produces a JPEG image as output.

SYNTAX

```
<ie:webject name="Pie-Chart" type="IMG">
  <ie:param name="BACKGROUND_COLOR " data="html_color"/>
  <ie:param name="COLOR " data="html_color"/>
  <ie:param name="DECIMAL_PLACES" data="integer_places"/>
  <ie:param name="GRAPH" data="column_name"/>
  <ie:param name="GROUP_IN" data="input_groupname"/>
  <ie:param name="HEIGHT" data="integer_height"/>
  <ie:param name="INSET_BOTTOM" data="integer_inset"/>
  <ie:param name="INSET_LEFT" data="integer_inset"/>
  <ie:param name="INSET_RIGHT" data="integer_inset"/>
  <ie:param name="INSET_TOP" data="integer_inset"/>
  <ie:param name="INSETS" data="integer_inset"/>
  <ie:param name="LABEL" data="column_name"/>
  <ie:param name="TEXT_COLOR " data="html_color"/>
  <ie:param name="WIDTH" data="integer_width"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GRAPH		BACKGROUND_COLOR
GROUP_IN		COLOR
		DECIMAL_PLACES
		HEIGHT
		INSET_BOTTOM
		INSET_LEFT
		INSET_RIGHT
		INSET_TOP
		INSETS
		LABEL
		TEXT_COLOR
		WIDTH

BACKGROUND_COLOR

Specifies the color, in #RRGGBB format, of the background of the generated image. The default value is white.

COLOR

Specifies the color, in #RRGGBB format, that the generated slices of pie should be rendered in. This parameter may be multi-valued. If it is multi-valued, then the colors specified are used in order and all values are iterated over until all data in the input group has been graphed. For example, specifying two color parameters such as the following:

```
<ie:param name="COLOR" data="#FF0000,#0000FF" delim=","/>
```

causes alternating red and blue.

Optionally each input group can contain a column named “COLOR” to specify the color to use for each data point in the input group. The COLOR parameter overrides data contained in the input group.

The default value is white.

DECIMAL_PLACES

Specifies an integer that governs how many decimal places are displayed on percent values of labels. This parameter is ignored if no value is given for LABEL. Default value is 0.

GRAPH

Specifies the name of the column containing the data to be graphed.

GROUP_IN
Specifies the name of the input group containing data to be graphed.

HEIGHT
Specifies the integer height of the generated image. The default value is 400.

INSET_BOTTOM
Specifies an integer value to use as insets to pad the bottom of the image. The default value is 10.

INSET_LEFT
Specifies an integer value to use as insets to pad the left side of the image. The default value is 10.

INSET_RIGHT
Specifies an integer value to use as insets to pad the right side of the image. The default value is 10.

INSET_TOP
Specifies an integer value to use as insets to pad the top of the image. The default value is 10.

INSETS
Specifies an integer value to use as insets to pad the outside of the image. If this parameter is specified it overrides the presence of INSET_LEFT, INSET_RIGHT, INSET_TOP and INSET_BOTTOM. The default value is 10.

LABEL
Specifies the name of the column containing strings used to label each slice of the pie. Labels are of the form `<text>(XX%)` where `<text>` is the value for the data point being graphed and `XX` is the percent of the total dataset the point represents. The default is not to display labels.

TEXT_COLOR
Specifies the color, in #RRGGBB format, of text placed in the generated image. The default value is black.

WIDTH
Specifies the integer width of the generated image. The default value is 600.

EXAMPLE

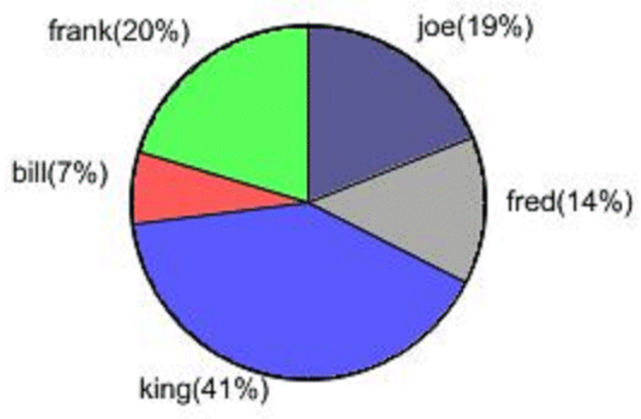
Given the following input data (in GROUP_IN named "Employees"):

Name	Sales
joe	7000
fred	5000
king	15000
bill	2500
frank	7500

and the following JSP:

```
<%@page language="java" session="true"
contentType="image/jpeg"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<ie:webject name="Pie-Chart" type="IMG">
  <ie:param name="GROUP_IN" data="Employees"/>
  <ie:param name="COLOR"
            data=" #5B5B97,#AAAAAA,#5B5BFF,#FF5B5B,#5BFF5B"
            delim=","/>
  <ie:param name="GRAPH" data="SAL"/>
  <ie:param name="WIDTH" data="300"/>
  <ie:param name="HEIGHT" data="200"/>
  <ie:param name="LABEL" data="NAME"/>
</ie:webject>
```

Results in the following image:



14 Task Webject Reference

The following topics contain information about Info*Engine task webjects. Each individual listing contains the webject name, a description of its use, details of its syntax, descriptions of all parameters, and, in most cases, an example.

Before their descriptions, parameters are grouped in the following categories:

- Required — The parameter is always required.
- Select — A relationship between the specified parameter and another parameter exists. For example, the ELEMENT and XML_URL parameters of the Create-Group webject are select because at least one of the parameters is required.

  A parameter is also listed in the “Select” column when there is a relationship between the values in the specified parameter and the use of another parameter. For example, in the Create-Object webject, the value entered for the TYPE parameter determines which PUT_ parameters can be used. Therefore, both the TYPE and PUT_ parameters are listed in the “Select” column.
- Optional — The parameter is always optional and is not related to another parameter.

Administrative Webjects

The following webjects provide a quick way to perform specific administrative tasks, such as gathering simple statistics or causing service properties to reload at runtime:

- Get-Statistics on page 358
- Reload-Properties on page 359

All administrative webjects use the `ADM` **type** attribute value in the **webject** tag.

Get-Statistics

DESCRIPTION

Returns a group containing simple statistical information related to the specified process. The returned group contains the following statistical information:

- **serviceName** — Name of the service
- **serviceType** — Type of service. Valid service types are task processor or adapter.
- **activeThreads** — Number of actively running threads, not including the current thread.
- **handledRequests** — Number of requests handled.
- **averageResponse** — Average time in milliseconds it has taken to respond to a request.

  Requests to execute administrative webjects are not included in the statistics.

Note

This webject is only supported by processes listening on a port, such as an out-of-process adapter or a task processor.

SYNTAX

```
<ie:webject name="Get-Statistics" type="ADM">
  <ie:webject name="INSTANCE" data="instance"/>
</ie:webject>
```

PARAMETERS

Required	Select	Option-al
INSTANCE		

INSTANCE

Specifies the name of the process in which this webject is to be run. An adapter that is running out-of-process or a task processor are both processes.

This parameter is required.

EXAMPLE

```
<ie:webject name="Get-Statistics" type="ADM">
  <ie:param name="INSTANCE"
 data="com.myCompany.myHost.server.taskProcessor"/>
</ie:webject>
```

Reload-Properties

DESCRIPTION

Reloads properties associated with either the currently running `serviceName` or a specific `serviceName`. This webject can be executed in out-of-process adapters, task processors, or JSP pages. If the webject is executed locally in a JSP page, the properties are reloaded by a service running within the servlet JVM, such as **IeServlet**, **SoapRPCRouter**, or **JSP** properties.

SYNTAX

```
<ie:webject name="Reload-Properties" type="ADM">
  <ie:param name="INSTANCE" data="instance"/>
  <ie:param name="SERVICE_NAME" data="service_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		INSTANCE
		SERVICE_NAME

INSTANCE

Specifies the name of the process in which this webject is to be run. An adapter that is running out-of-process or a task processor are both processes.

This parameter is optional.

SERVICE_NAME

Specifies the name of the service whose properties are to be reloaded. If this parameter is omitted, then the webject reloads the properties of the currently running service. If the value specified for SERVICE_NAME is not a running service, then `com.infoengine.au.NoSuchServiceException` is thrown.

This parameter is optional.

Group Webjects

The following webjects compare, combine, or sort one or more existing groups of data that have been generated as a result of query, action, or other group webjects:

- Change-Group on page 360
- Concat-Groups on page 362
- Copy-Group on page 364
- Create-Group on page 366
- Diff-Groups on page 371
- Extract-Group on page 374
- Format-Group on page 376
- Intersect-Groups on page 379
- Join-Groups on page 382
- Merge-Groups on page 386
- Return-Groups on page 388
- Set-Identity on page 391
- Set-Metadata on page 393
- Sort-Group on page 397
- Subset-Group on page 402
- Summarize-Groups on page 405
- Translate-Group on page 406
- Union-Groups on page 412
- XOR-Groups on page 414

All group webjects use the `GRP` **type** attribute value in the **webject** tag.

Change-Group

DESCRIPTION

Changes the name of one or more attributes in a group. The resulting group contains the changed attributes and all attributes not affected by the change.

SYNTAX

```
<ie:webject name="Change-Group" type="GRP">
  <ie:param name="GROUP_IN" data="value"/>
  <ie:param name="GROUP_OUT" data="group_out_name"/>
  <ie:param name="RENAME" data="'old_name'='new_name'"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		GROUP_OUT
RENAME		

GROUP_IN

Specifies the name of the group with information to be renamed. This parameter is required.

GROUP_OUT

Specifies the name of the resulting group with renamed information. If GROUP_OUT is specified, the original group is replaced by the output group and is no longer available in the VDB. The default for this parameter is for the attribute changes to be made in the original group. This parameter is optional.

RENAME

Specifies the attribute name to be changed. Specific syntax is required when renaming an attribute. For example, to change the attribute name *objectClass* to *XYZclass*, the following RENAME parameter would be specified:

```
<ie:param name="RENAME" data="'objectClass'='XYZclass'">
```

As shown above, the value of the RENAME parameter must be enclosed in double quotes. The old name and the new name of the attribute both must be enclosed in single quotes. Attribute names are case insensitive, therefore case is ignored when the parameter searches for the existing attribute name.

This parameter is required.

EXAMPLE

The webject changes the attribute names in the input group named “createdgroup” and adds the “results” group to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Creates input group named createdgroup. -->

<ie:task uri="/com/company/CreateGroup.xml"/>

<!-- Renames attributes in a group. -->

<ie:webject name="Change-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="RENAME" data="'ADDRESS'='HomeAddress'"/>
```

```
    <ie:param name="RENAME" data="'NAME'='FullName'"/>
    <ie:param name="RENAME" data="'EMAIL'='email'"/>
    <ie:param name="GROUP_OUT" data="results"/>
  </ie:webject>
```

The XML output from executing the example is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="results" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <FullName>Sam Johnson</FullName>
    <HomeAddress>1234 Main St.</HomeAddress>
    <email>sjohnson@somewhere.com</email>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <FullName>Harvy Anderson</FullName>
    <HomeAddress>1234 Amber St.</HomeAddress>
    <email>handerson@somewhere.com</email>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <FullName>James O&apos;Connor</FullName>
    <HomeAddress>775 Main St.</HomeAddress>
    <email></email>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <FullName>Harvey Hampton</FullName>
    <HomeAddress>775 Main St.</HomeAddress>
    <email>hhampton@somewhere.com</email>
  </wc:INSTANCE>
</EmployeeData>
</wc:COLLECTION>
```

Concat-Groups

DESCRIPTION

Links two groups of data together in a series, allowing duplicate information to appear from both groups.

For example, group A contains the elements u, v, and x and group B contains the elements x, y, and z. The Concat-Groups webject links the elements of groups A and B together, one after the other, to form group C. Group C would then contain the elements u, v, x, x, y, and z.

SYNTAX

```
<ie:webject name="Concat-Groups" type="GRP">
```

```
  <ie:param name="CLASS" data="class"/>
  <ie:param name="GROUP_IN" data="group_name1"/>
  <ie:param name="GROUP_IN" data="group_name2"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CLASS
GROUP_OUT		

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example, if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

GROUP_IN

Specifies the two or more groups that are used in the concatenation. This parameter is required.

GROUP_OUT

Specifies the name of the resulting group. This parameter is required.

EXAMPLE

The following example links the two specified groups together:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Form a group by concatenating two groups -->

<ie:task uri="/com/company/CreateGroup.xml"/>
<ie:task uri="/com/company/CreateGroupHr.xml"/>

<ie:webject name="Concat-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createhrgroup"/>
  <ie:param name="CLASS" data="ConcatGroup"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

Copy-Group

DESCRIPTION

Copies a group to a new group of a different name. For example, the webject might copy all the data in group “A”, creating a new group called “B” with exactly the same information.

SYNTAX

```
<ie:webject name="Copy-Group" type="GRP">
  <ie:param name="GROUP_IN" data="group_in_name"/>
  <ie:param name="GROUP_OUT" data="group_out_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		
GROUP_OUT		

GROUP_IN
Specifies the name of the group to be copied. This parameter is required.

GROUP_OUT
Specifies the name of the new group that is created. This name must not be identical to the GROUP_IN name. This parameter is required.

EXAMPLE

The following example copies the input group named “createdgroup” into the “results” group and returns both groups (using the Return-Groups webject) to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<!-- Copies a group to a new group. -->

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Copy-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="results"/>
</ie:webject>
```

The XML output from executing the example is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="createdgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL>hhampton@somewhere.com</EMAIL>
  </wc:INSTANCE>
</EmployeeData>
<EmployeeData NAME="results" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
```

```
       <ADDRESS>775 Main St.</ADDRESS>
       <EMAIL>hhampton@somewhere.com</EMAIL>
     </wc:INSTANCE>
   </EmployeeData>
   </wc:COLLECTION>
```

Create-Group

DESCRIPTION

Creates a data group from embedded parameters or from an XML source file. If the XML source conforms to the Info*Engine schema, a group can be created from it directly. Otherwise, this webject allows an XSL stylesheet to be applied in order to translate the source form to the Info*Engine schema.

The Create-Group webject accepts an XML_URL parameter. The value of this parameter can be a fully specified HTTP URL. If so, Create-Group creates an HTTP connection to a remote web server, delivers the URL, and then reads the response from the remote server. The response is expected to be an XML stream. Ideally, the remote web server returns an XML stream that is formatted as a collection of Info*Engine groups (the same format as is produced by the Info*Engine task processor).

In cases where the format of the XML stream is not the same as produced by the Info*Engine task processor, Create-Group accepts an XSL_URL parameter specifying an XSL stylesheet that can be applied to translate the XML stream into Info*Engine format. For a list of the XML Info*Engine output requirements, see Understanding XML Output for Info*Engine Groups on page 107.

You can use the Create-Group webject to create an empty group. When you specify only the GROUP_OUT parameter and no ELEMENT parameters, Info*Engine creates an empty group.

SYNTAX

```
<ie:webject name="Create-Group" type="GRP" >
  <ie:param name="CLASS" data="class"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="DELIMITER" data="delimiter_character"/>
  <ie:param name="ELEMENT" data="key_value_pairs"/>
  <ie:param name="GROUP_OUT" data="group_out_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="XML_URL" data="url_of_xml_source"/>
  <ie:param name="XSL_PARAM" data="name_value_pair"/>
  <ie:param name="XSL_URL" data="url_of_xsl_source"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	ELEMENT	CLASS
	GROUP_OUT	DBUSER
	XML_URL	DELIMITER
	XSL_PARAM	PASSWD
	XSL_URL	

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name”. This parameter is optional.

DBUSER

Specifies the username to be used to authenticate to XSL_URL. If the XSL templates to be used by the webject reside on a remote HTTP server, then this parameter should be used in conjunction with the PASSWD attribute.

This parameter is optional.

DELIMITER

Specifies the character used to delimit key and value pairs created in the ELEMENT parameter.

If you are using a substitution string containing an asterisk (*) within the ELEMENT parameter, you should not specify the characters that are used in separating multiple elements or values as the delimiter in the Create-Group webject. If you do use the same character, the resulting elements or attribute values may be different from what you expect them to be. Substitution is described in Dynamic Parameter Value Substitution on page 44.

The default for this parameter is the colon (:). This parameter is optional.

ELEMENT

Specifies a delimited set of key and value pairs to add to the group specified by the GROUP_OUT parameter. Each key and value pair is specified by `key=value`. Each ELEMENT parameter creates an instance in the group with the attribute specified by the `key` value and the value specified by the `value` portion of the key and value pair.

Multiple ELEMENT parameters can be specified to create multiple instances of objects in the group. At least one ELEMENT parameter is required for this webject if no XML_URL parameter is specified. If both XML_URL and ELEMENT parameters are specified, the resulting group is the sum total of the rows created by the XML source file and the ELEMENT parameters specified.

GROUP_OUT

Specifies the name of the resulting group. The GROUP_OUT parameter is required only when the XML_URL parameter is not specified. When the XML_URL parameter is specified, the XML stream being parsed contains one or more groups. The names of these groups can be specified in the XML stream, so it is not necessary to provide a GROUP_OUT parameter to supply a name for the group(s) in this case.

PASSWD

Specifies the password corresponding to DBUSER.

This parameter is optional.

XML_URL

Identifies the location of an XML source file from which to create the group. A relative URL or a fully qualified URL can be specified. Relative URLs are relative to the Info*Engine Server task template root. The XML source is expected to be in Info*Engine format; the same format as is produced by the Info*Engine task processor. If the contents of the source specified by the XML_URL parameter is not in Info*Engine format, the XSL_URL parameter can be used to specify an XSL stylesheet that can be applied to transform the source into Info*Engine format. In any case, the source (possibly transformed by XSL) can contain definitions for more than one Info*Engine group. Create-Group parses all of them and add them to its collection of output groups.

Fully qualified URLs are dereferenced using the **Auth-Map** context group data. The **Auth-Map** is searched for a username and password based on the domain name found in the fully qualified URL. For example, if the fully qualified URL is:

```
http://machine.com/infoengine/servlet/IE/tasks/createGroupData.xml
```

then the **Auth-Map** context group is searched for a username and password that has an INSTANCE name of `http://machine.com`. If a username and password is found, BASIC authentication is used. If no username or password is found, no authentication information is sent to the remote web server.

If the data value contains the `://` string, it is assumed to be a fully qualified internet URL. If the data value does not contain the string, it is assumed to be a local file relative to the current task root directory.

If the data value is `input:` the webject reads an XML object from the task's BLOB input stream. This allows XML objects to be submitted from web pages and converted into groups by Create-Group.

XSL_PARAM

Defines XSL parameters that are then passed to the XSL stylesheet named in the XSL_URL parameter. You enter the value for the XSL_PARAM parameter in the form `XSL_name=XSL_value`, where `XSL_name` is the name of a parameter in the XSL stylesheet and `XSL_value` is the value you want set for the parameter.

The default for XSL_PARAM is that no parameters are passed to the stylesheet. Multiple values can be specified for this parameter. This parameter is optional.

XSL_URL

Identifies the location of an XSL stylesheet to apply to the XML source specified by the XML_URL parameter. A relative URL or a fully qualified URL can be specified. Relative URLs are relative to the Info*Engine Server task root.

Fully qualified URLs are dereferenced using the **Auth-Map** context group data. The **Auth-Map** is searched for a username and password based on the domain name found in the fully qualified URL. For example, if the fully qualified URL is:

```
http://machine.com/infoengine/servlet/IE/tasks/createGroupData.xml
```

the **Auth-Map** context group is searched for a username and password that has an INSTANCE name of `http://machine.com`. If a username and password are found, BASIC authentication information is used when accessing the URL. If no username and password are found, no authentication information is sent to the remote web server.

If the data value contains the `://` string, it is assumed to be a fully qualified internet URL. If the data value does not contain the string, it is assumed to be a local file relative to the current task root directory.

EXAMPLE: INTERNAL GROUP CREATION

In the following example, the webject specifies the elements in the output group named “createdgroup” and adds the group to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Create an internal Group -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Sam
                Johnson:ADDRESS=1234 Main
                St.:EMAIL=sjohnson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=Harvy
                Anderson:ADDRESS=1234 Amber
                St.:EMAIL=handerson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=James
                O'Connor:ADDRESS=775 Main St.:EMAIL="/>
  <ie:param name="ELEMENT" data="NAME=Harvey
                 Hampton:ADDRESS=775 Main
                 St.:EMAIL=hhampton@somewhere.com"/>
  <ie:param name="CLASS" data="EmployeeData"/>
  <ie:param name="GROUP_OUT" data="createdgroup"/>
</ie:webject>
```

The XML output from executing the example is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="createdgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL>hhampton@somewhere.com</EMAIL>
  </wc:INSTANCE>
</EmployeeData>
</wc:COLLECTION>
```

EXAMPLE: EXTERNAL GROUP CREATION

The following example of Create-Group executes a task in a remote Info*Engine Server and adds its output groups to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Create one or more local groups from the output of a remote
                                                     Info*Engine task -->
```

```
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="XML_URL" data="http://remote-ie.acme.com/
                              infoengine/servlet/IE/tasks/report.xml"/>
</ie:webject>
```

Diff-Groups

DESCRIPTION

Computes the difference of two groups and places that difference in an output group.

In general, an element is different for two groups if it is part of the first group but not both groups. For example, if an element x exists in group A and group B, then it is not placed in the output group. Additionally, if an element y exists in group A but not in group B, then it is placed in the output group. However, if an element z exists in group B but not in group A then it is not placed in the output group; only the items unique to group A are placed in the output.

SYNTAX

```
<ie:webject name="Diff-Groups" type="GRP">
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="DIFFBY" data="attribute"/>
  <ie:param name="GROUP_IN" data="group_name1"/>
  <ie:param name="GROUP_IN" data="group_name2"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]">
</ie:webject>
```

PARAMETERS

Required	Select	Optional
DIFFBY		CASE_IGNORE
GROUP_IN		CLASS
GROUP_OUT		COMPARISON
		SORTBY
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS=MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name”. This parameter is optional.

COMPARISON

Describes how to compare the two groups: either ALPHA for an alpha-numeric comparison, or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. This parameter is optional.

DIFFBY

Identifies the name of the attribute that is used in comparing elements to determine whether they are different. If the groups being compared contain elements with attributes of the same name, then one DIFFBY value can be specified. Otherwise, you must specify two DIFFBY parameters to provide the names of the attributes from each respective group that is compared.

The attribute name placed in the output group is the attribute name of the first DIFFBY parameter used. This parameter is required.

GROUP_IN

Specifies the two groups that are compared for differences. This parameter is required.

GROUP_OUT

Specifies the name of the single resulting group comparison. This parameter is required.

SORTBY

Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED

Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

EXAMPLE

The following example uses the Create-Group webject and then finds the difference between the NAME attribute in the groups using the Diff-Groups webject. It creates the output group named “DIFF-RESULTS” and adds the group to the local output group collection:

```
%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Create Group containing the difference between two
                                                  groups -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Sam
                Johnson:ADDRESS=1234 Main
                St.:EMAIL=sjohnson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=Harvy
                Anderson:ADDRESS=1234 Amber
                St.:EMAIL=handerson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=James
                O'Connor:ADDRESS=775 Main St.:EMAIL="/>
  <ie:param name="ELEMENT" data="NAME=Harvey
                Hampton:ADDRESS=775 Main
                St.:EMAIL=hhampton@somewhere.com"/>
  <ie:param name="CLASS" data="EmployeeData"/>
  <ie:param name="GROUP_OUT" data="CREATE-RESULTS"/>
</ie:webject>

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Sam
           Johnson:POSITION=Engineer:PHONE=555-111-1111"/>
  <ie:param name="ELEMENT" data="NAME=Harvy
         Anderson:POSITION=Marketing:PHONE=555-222-2222"/>
  <ie:param name="ELEMENT" data="NAME=James
                O'Connor:POSITION=Management"/>
  <ie:param name="CLASS" data="EmployeeHrData"/>
  <ie:param name="GROUP_OUT" data="CREATE-HR-RESULTS"/>
</ie:webject>

<ie:webject name="Diff-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="CREATE-RESULTS"/>
  <ie:param name="GROUP_IN" data="CREATE-HR-RESULTS"/>
  <ie:param name="DIFFBY" data="NAME"/>
  <ie:param name="CLASS" data="NewEmployees"/>
  <ie:param name="GROUP_OUT" data="DIFF-RESULTS"/>
```

```
   </ie:webject>
```

The XML output from executing the example is the following:

```
   <?xml version="1.0" encoding="UTF-8"?>
   <wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
   <NewEmployees NAME="DIFF-RESULTS" TYPE="Object" STATUS="0">
     <wc:INSTANCE>
       <NAME>Harvey Hampton</NAME>
       <ADDRESS>775 Main St.</ADDRESS>
       <EMAIL>hhampton@somewhere.com</EMAIL>
     </wc:INSTANCE>
   </NewEmployees>
   </wc:COLLECTION>
```

Extract-Group

DESCRIPTION

Searches for one or more attribute names and values and puts them into a new group. This forms a type of subset of a group of attributes. For information on searching for a subset of objects (not attributes on objects), see Subset-Group on page 402.

When used in combination with the EXTRACT parameter, a specific set of attributes are extracted from the specified subset of elements. If the ELEMENT_OFFSET parameter is omitted, its value defaults to 0 (thus, the subset of elements returned begin with the first element of the input group). If the ELEMENT_COUNT parameter is omitted, its value defaults to the size of the input group (thus, the subset of elements returned include all elements of the input group, starting from the one specified by ELEMENT_OFFSET). If ELEMENT_OFFSET + ELEMENT_COUNT exceeds the number of elements in the input group, the output group contains fewer ELEMENT_COUNT elements (it contains as many elements as remain in the input group, beginning with ELEMENT_OFFSET). If the EXTRACT parameter is omitted, all attributes of each selected element are returned. At least one of the EXTRACT, ELEMENT_OFFSET, or ELEMENT_COUNT parameters must be specified. If all three are omitted, an exception is thrown.

SYNTAX

```
<ie:webject name="Extract-Group" type="GRP">
  <ie:param name="CLASS" data="class"/>
  <ie:param name="EXTRACT" data="attribute_name"/>
  <ie:param name="EXTRACT_MODE" data="[MATCH | NOMATCH]"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="GROUP_OUT" data="output_group"/>
 <ie:param name="ELEMENT_COUNT" data="number of elements to extract" />
 <ie:param name="ELEMENT_OFFSET" data="index of the starting element to extract" />
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN	ELEMENT_COUNT	CLASS
GROUP_OUT	ELEMENT_OFFSET	EXTRACT_MODE
	EXTRACT	

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

ELEMENT_COUNT

Allows the webject to return a subset of elements from the input group, thus allowing this webject to be used in simulating page-mode queries in cases where true page-mode is not supported by a back-end information system. This parameter specifies the number of elements to extract. The default value is the size of the input group.

ELEMENT_OFFSET

Allows the webject to return a subset of elements from the input group, thus allowing this webject to be used in simulating page-mode queries in cases where true page-mode is not supported by a back-end information system. This parameter specifies the index of the element at which to begin extraction. The default value is 0.

EXTRACT

Specifies the name of the attribute to be extracted. Multiple values can be specified for this parameter. This parameter is required.

EXTRACT_MODE

Specifies the extract mode. The value can be either MATCH or NOMATCH. When EXTRACT_MODE is MATCH, the specified attributes on the EXTRACT parameter are included in the output group. When EXTRACT_ MODE is NOMATCH, all but the specified attributes on the EXTRACT parameter are included in the output group. This parameter is optional. If not specified, the default mode of MATCH is assumed.

GROUP_IN

Specifies the name of the input group. This parameter is required.

GROUP_OUT

Specifies the name of the extraction of attributes from the input group. This parameter is required.

EXAMPLE

The following example searches for the specified attribute names and values and puts them into a new group named “extractresults”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Extract attributes from a group. -->

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Extract-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="EXTRACT" data="NAME"/>
  <ie:param name="EXTRACT" data="ADDRESS"/>
  <ie:param name="CLASS" data="resultGroup"/>
  <ie:param name="GROUP_OUT" data="extractresults"/>
</ie:webject>
```

Format-Group

DESCRIPTION

Formats specific groups of information by adding special characters to that information. For example, this webject could be used to place the dollar sign at the beginning of each item in a group of salary amounts. This webject can also be used to combine multiple attributes into one long attribute for formatting purposes.

SYNTAX

```
<ie:webject name=Format-Group type=GRP>
  <ie:param name="ATTRIBUTE" data="attribute_name"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="FORMAT" data="format_specifier"/>
  <ie:param name="GROUP_IN" data="input_group_name"/>
  <ie:param name="GROUP_OUT" data="output_group"/>
  <ie:param name="LOCALE" data="language"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
ATTRIBUTE		CLASS
FORMAT		LOCALE
GROUP_IN		
GROUP_OUT		

ATTRIBUTE

Identifies the string of data in a group to be formatted. These define the attributes that are used in creating the GROUP_OUT. Only the attributes explicitly specified in ATTRIBUTE parameters are used in creating the GROUP_OUT.

Multiple values can be specified for this parameter. This parameter is required.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

FORMAT

Specifies how to format the string of data specified in the ATTRIBUTE parameter. The value of the FORMAT parameter must be acceptable to the Java class `java.text.MessageFormat`. Each row of the GROUP_OUT has as many attributes as there are FORMAT parameters.

Each FORMAT parameter can include substitution strings that look like `{n}`, where `n` is a non-negative integer. These strings are replaced in the GROUP_OUT by the next unconsumed attribute whose offset is defined by `n`. The offset is relative to the first unconsumed ATTRIBUTE value as of the start of processing of the FORMAT parameter.

For example, if `n` is 0, the very next unconsumed ATTRIBUTE value is substituted. If `n` is 1, the second unconsumed attribute value is substituted. If `n` is 2, the third unconsumed attribute value is substituted, and so on. When the FORMAT parameter has been processed completely, all ATTRIBUTE values substituted by it are marked as consumed.

Multiple values can be specified for this parameter. This parameter is required.

GROUP_IN

Specifies the group to be formatted. This parameter is required.

GROUP_OUT

Specifies the formatted group. This parameter is required.

LOCALE

A string representation of the Java class locale, such as `en-US` (ISO language name, dash, ISO country code). If not specified, it currently defaults to the default locale defined for the platform on which Info*Engine is running. This parameter is optional.

EXAMPLE

The following example formats data in the current VDB group using substitution strings. The formatted data is placed in an output group named “formatted results” with a specified Java class locale of “en-US”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Format elements of a group. -->

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Format-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE" data="EMAIL"/>
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="ATTRIBUTE" data="EMAIL"/>
  <ie:param name="ATTRIBUTE" data="EMAIL"/>
  <ie:param name="ATTRIBUTE" data="NAME"/>
  <ie:param name="FORMAT" data="{0}"/>
  <ie:param name="FORMAT" data="{0}"/>
  <ie:param name="FORMAT" data="Name: {0} lives at {1}"/>
  <ie:param name="FORMAT" data="Email: {0} --- Name: {1}"/>
  <ie:param name="CLASS" data="resultGroup"/>
  <ie:param name="GROUP_OUT" data="formatted results"/>
  <ie:param name="LOCALE" data="en-US"/>
</ie:webject>
```

Intersect-Groups

DESCRIPTION

Identifies the intersection of two groups and places the result in a new group. For example, say that group A contains the elements u, v, and x and group B contains the elements x, y, and z. This webject identifies the common elements of both groups and uses the results of an intersection to form group C. Group C in this example would contain only the element x.

SYNTAX

```
<ie:webject name="Intersect-Groups" type="GRP">
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]/>
  <ie:param name="GROUP_IN" data="input_groups"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="INTERSECTBY" data="attribute"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
  <ie:param name="CASE_IGNORE" data="[ TRUE | FALSE]"/>
  </ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
INTERSECTBY		COMPARISON
		SORTBY
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

COMPARISON

Describes how to compare the two groups: either ALPHA for an alpha-numeric comparison, or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. This parameter is optional.

GROUP_IN

Specifies the names of two input groups that are used in computing an intersection.

To specify two group names, you can include two lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
```

This parameter is required.

GROUP_OUT

Specifies the name of the results of computing the intersection of the two groups. This parameter is required.

INTERSECTBY

Identifies the attribute to be used for comparisons. When an intersection is performed, the named attribute of the first named group is compared with the named attribute of the second named group. If the values are the same, then the element from the first group is placed in the resulting output group. Otherwise, it is discarded.

The attribute name placed in the output group is the attribute name of the first INTERSECTBY parameter used. If all the group INTERSECTBY parameters are the same name, then only one need be specified here. If any of the group INTERSECTBY parameters have different names, then all of them must be specified here.

This parameter is required.

SORTBY

Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED

Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

EXAMPLE

The following example creates two groups using the **CreateGroup** and **CreateGroupHr** task tags, and then finds the intersection between the NAME attribute in the groups using the Intersect-Groups webject. It creates the output group named “results” and adds the group to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!--Create a group named createdgroup. -->
<ie:task uri="com/company/CreateGroup.xml"/>

<!--Create a group named createdhrgroup. -->
<ie:task uri="com/company/CreateGroupHr.xml"/>

<!-- Form a group by intersecting the two groups. -->

<ie:webject name="Intersect-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createhrgroup"/>
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="INTERSECTBY" data="NAME"/>
  <ie:param name="CASE_IGNORE" data="YES"/>
  <ie:param name="COMPARISON" data="ALPHA"/>
  <ie:param name="SORTBY" data="NAME"/>
  <ie:param name="SORTED" data="ASC"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

The XML output from executing the example is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<Unknown-Class-Name NAME="results" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <POSITION>Marketing</POSITION>
    <PHONE>555-222-2222</PHONE>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <POSITION>Management</POSITION>
    <PHONE></PHONE>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <POSITION>Engineer</POSITION>
    <PHONE>555-111-1111</PHONE>
  </wc:INSTANCE>
</Unknown-Class-Name>
</wc:COLLECTION>
```

Join-Groups

DESCRIPTION

Joins similar information from two different groups into one group. By setting the JOIN_TYPE parameter to MAX, dissimilar information within both groups is also included in the new group.

Note

The difference between the Join-Groups webject and the Merge-Groups webject depends on how duplicate information is treated. The results of joining groups is that duplicated information is eliminated. Merging groups of data allows duplicates to be preserved.

SYNTAX

```
<ie:webject name="Join-Groups" type="GRP">="GRP">
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="GROUP_IN" data="input_groups"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="JOIN_TYPE" data="MAX"/>
  <ie:param name="JOINBY" data="attribute"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
  <ie:param name="UNDEFINED" data="somevalue"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
JOINBY		COMPARISON
		JOIN_TYPE
		SORTBY
		SORTED
		UNDEFINED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example, if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is "Unknown-Class-Name." This parameter is optional.

COMPARISON

Describes how to compare the two groups: either ALPHA for an alpha-numeric comparison, or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. This parameter is optional.

GROUP_IN

Specifies the names of the two groups to be used in computing a join. To specify two group names, you can include two lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
```

The order of the groups listed in the parameter determines the order in which the groups are joined.

This parameter is required.

GROUP_OUT

Specifies the name of the results of computing the join of the two groups. This parameter is required.

JOINBY

Identifies the attribute or column name to be used for comparisons. When a join is performed, an element of the first named group is joined with an element of the second named group if they contain attributes named by the JOINBY parameter(s) with matching values.

If the attributes used for the comparison in both groups have the same name, then include only one JOINBY parameter. If the attributes used for the comparison do not have the same name, then include two JOINBY parameters. The attribute name placed in the output group is the attribute name of the first JOINBY parameter used.

This parameter is required.

JOIN_TYPE

Specifies that a MAX join is to be performed. Without JOIN_TYPE specified, any information that is not similar within the two groups being joined are omitted from the output. When JOIN_TYPE is set to MAX, the webject joins all information from both groups being joined. For fields without values, the UNDEFINED value is used. For more information, see Join-Groups on page 384.

This parameter is optional.

SORTBY

Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED

Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

UNDEFINED

Sets the value to use if no attribute value exists. The default value for this parameter is `""`. This parameter is optional.

EXAMPLES

There are two examples for the Join-Groups webject. The first is an example task, and the second supplements the definition of the JOIN_TYPE parameter.

The following example creates two groups and then joins them by the NAME attribute using the Join-Groups webject. It creates an output group named “results” and adds the group to the local output group collection:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!--Create a group named createdgroup. -->
<ie:task uri="com/company/CreateGroup.xml"/>

<!--Create a group named createdhrgroup. -->
<ie:task uri="com/company/CreateGroupHr.xml"/>

<!-- Form a group by joining two groups -->

<ie:webject name="Join-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createhrgroup"/>
  <ie:param name="JOINBY" data="NAME"/>
  <ie:param name="JOIN_TYPE" data="MAX"/>
  <ie:param name="SORTBY" data="NAME"/>
  <ie:param name="SORTED" data="ASC"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

This example shows the results of combining Group A with Group B on C3 with the Join-Groups webject. The first two tables show the groups that are joined. Notice that the information in Column 3, Row 3 is not the same in these two groups.

Table 1 Group A

	C1	C2	C3
Row 1	a1	b1	c1
Row 2	a2	b2	c2
Row 3	a3	b3	c3

Table 2 Group B

	C3	C4	C5
Row 1	c1	d1	e1
Row 2	c2	d2	e2
Row 3	c4	d4	e4

Assume that the following webject executes:

```
<ie:webject name="Join-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="Group A"/>
  <ie:param name="GROUP_IN" data="Group B"/>
  <ie:param name="JOINBY" data="C3"/>
  <ie:param name="GROUP_OUT" data="Results"/>
</ie:webject>
```

Table 3 Results Without the JOIN_TYPE Parameter

	C1	C2	C3	C4	C5
Row 1	a1	b1	c1	d1	e1
Row 2	a2	b2	c2	d2	e2

This webject does not use the JOIN_TYPE parameter. Notice that the columns from Group A precedes the columns from Group B and that information from Row 3 in Groups A and B is not included.

```
<ie:webject name="Join-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="Group A"/>
  <ie:param name="GROUP_IN" data="Group B"/>
  <ie:param name="JOINBY" data="C3"/>
  <ie:param name="JOIN_TYPE" data="MAX"/>
```

```
    <ie:param name="GROUP_OUT" data="Results"/>
  </ie:webject>
```

The result when the JOIN_TYPE parameter is specified is shown below. All rows of information are included.

In this example, Group A (which had values for Row 3 in C1, C2, and C3) has those values included in Row 3. Group B (which had values for Row 3 in C3, C4, and C5) has those values included in Row 4. The default value that is included for information that is dissimilar between the groups is `""`.

Table 4 Results With the JOIN_TYPE Parameter

	C1	C2	C3	C4	C5
Row 1	a1	b1	c1	d1	e1
Row 2	a2	b2	c2	d2	e2
Row 3	a3	b3	c3	""	""
Row 4	""	""	c4	d4	e4

Joining Group B with Group A results in the same data joined in a different order. The C3 column is derived from Group B. Data from Group B is first, followed by the data from Group A. If Group B is joined with Group A, you get the following:

Table 5 Results With the JOIN_TYPE Parameter

	C3	C4	C5	C1	C2
Row 1	c1	d1	e1	a1	b1
Row 2	C2	d2	e2	a2	b2
Row 3	C3	""	""	a3	b3
Row 4	C4	d4	e4	""	""

Merge-Groups

DESCRIPTION

Combines two groups into one and allows duplication of information contained in one or both groups.

Note

The difference between the Join-Groups webject and the Merge-Groups webject depends on how duplicate information is treated. The results of joining groups is that duplicated information is eliminated. Merging groups of data allows duplicates to be preserved.

SYNTAX

```
<ie:webject name="Merge-Groups "type="GRP">
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="GROUP_IN" data="input_groups"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
		COMPARISON
		SORTBY
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

COMPARISON

Describes how to compare the two groups: either ALPHA for an alpha-numeric comparison or NUMERIC for a strictly numeric comparison. The default for this optional parameter is ALPHA.

GROUP_IN

Specifies the names of the two groups to be used in computing a merge. To specify two group names, you can include two lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
```

This parameter is required.

GROUP_OUT
Identifies the name of the resulting merge of the two groups specified using the GROUP_IN and MERGEBY parameters. This parameter is required.

SORTBY
Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED
Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

EXAMPLE

The following example combines the two specified GROUP_INs, sorts the data by the NAME attribute in ascending order, and places the resulting merged and sorted data in an output group named “results”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Form a group by merging two groups -->

<ie:task uri="com/company/CreateGroup.xml"/>
<ie:task uri="com/company/CreateGroupHr.xml"/>

<ie:webject name="Merge-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createhrgroup"/>
  <ie:param name="SORTBY" data="NAME"/>
  <ie:param name="SORTED" data="ASC"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

Return-Groups

DESCRIPTION

Returns multiple groups from a task. These groups can then be used by calling tasks or JSP pages.

By default, only the last group that is created is available to the caller of a task.

SYNTAX

```
<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="group_names"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_ IN		

GROUP_IN

Specifies the names of the multiple groups to return. The special value " * " can be specified to return all groups produced by a task.

To specify multiple group names, you can include multiple lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
<ie:param name="GROUP_IN" data="group3"/>
```

This parameter is required.

EXAMPLE

The following example creates two groups using the **CreateGroup** and **CreateGroupHr** task tags and then returns the groups using the Return-Groups webject:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<!--Create a group named createdgroup. -->
<ie:task uri="com/company/CreateGroup.xml"/>

<!--Create a group named createdhrgroup. -->
<ie:task uri="com/company/CreateGroupHr.xml"/>

<!-- Return multiple groups. -->

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="Group_in" data="createdgroup"/>
  <ie:param name="Group_in" data="createhrgroup"/>
</ie:webject>
```

The XML output from executing the `ReturnGroups.xml` file is the following:

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

```
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<EmployeeData NAME="createdgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <ADDRESS>1234 Main St.</ADDRESS>
    <EMAIL>sjohnson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <ADDRESS>1234 Amber St.</ADDRESS>
    <EMAIL>handerson@somewhere.com</EMAIL>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL></EMAIL>
    </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvey Hampton</NAME>
    <ADDRESS>775 Main St.</ADDRESS>
    <EMAIL>hhampton@somewhere.com</EMAIL>
  </wc:INSTANCE>
</EmployeeData>
<EmployeeHrData NAME="createhrgroup" TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <NAME>Sam Johnson</NAME>
    <POSITION>Engineer</POSITION>
    <PHONE>555-111-1111</PHONE>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>Harvy Anderson</NAME>
    <POSITION>Marketing</POSITION>
    <PHONE>555-222-2222</PHONE>
  </wc:INSTANCE>
  <wc:INSTANCE>
    <NAME>James O&apos;Connor</NAME>
    <POSITION>Management</POSITION>
    <PHONE></PHONE>
  </wc:INSTANCE>
</EmployeeHrData>
</wc:COLLECTION>
```

Set-Identity

DESCRIPTION

Adds attributes and meta-attributes to every element of a group. These attributes specify the UFID (unique federation identifier) and object class of each element. Task delegation can then be easily applied to the group by providing it as the value of the GROUP_IN parameter of the Dispatch-Tasks webject.

SYNTAX

```
<ie:webject name="Set-Identity" type="GRP">
  <ie:param name="GROUP_IN" data="input_group_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN	DOMAIN	CLASS
	DOMAIN_ATTRIBUTE	CLASS_ATTRIBUTE
	UFID	GUID
		GUID_ATTRIBUTE
		LOCAL_ID
		LOCAL_ID_ATTRIBUTE
		UFID_ATTRIBUTE

CLASS

Specifies a logical object class name. This value is added to each element of the input group as the value of an attribute whose name is specified by CLASS_ATTRIBUTE. This parameter may be specified more than once. If the number of values is less than the number of elements in the input group, the last value specified is reused as many times as necessary to accommodate the remaining elements. If this parameter is omitted, no class name attribute is added to any elements of the input group. This parameter is optional.

CLASS_ATTRIBUTE

Specifies the name of the attribute in which logical class names specified by CLASS is stored. The default value of this parameter is `class`. If the CLASS parameter is omitted, CLASS_ATTRIBUTE is ignored. This parameter is optional.

DOMAIN

Specifies the domain name of an information repository. This value is combined with LOCAL_ID and GUID to form a UFID (unique federation identifier) value that is added to each element of the input group. This parameter may be specified more than once. If the number of values is less than the number of elements in the input group, the last value specified is reused as many times as necessary to accommodate the remaining elements. If this parameter is omitted, then either the DOMAIN_ATTRIBUTE or UFID parameter must be specified.

DOMAIN_ATTRIBUTE
Specifies the name of an attribute that each element of the input group contains. The value of each such attribute identifies the domain name of an information repository. Each value is combined with LOCAL_ID and GUID to form a UFID (unique federation identifier) value that is added to each element of the input group. If this parameter is omitted, then either the DOMAIN or UFID parameter must be specified.

GROUP_IN
Specifies the name of a group to which class and UFID attributes are added, which is modified in place (this webject has no GROUP_OUT parameter). This parameter is required.

GUID
Specifies the globally unique identifier of an information repository. This value is combined with LOCAL_ID and DOMAIN to form a UFID (unique federation identifier) value that is added to each element of the input group. This parameter may be specified more than once. If the number of values is less than the number of elements in the input group, the last value specified is reused as many times as necessary to accommodate the remaining elements. If this parameter is omitted, and GUID_ATTRIBUTE is also omitted, the globally unique identifier is derived from the DOMAIN value. This parameter is optional.

GUID_ATTRIBUTE
Specifies the name of an attribute contained in each element of the input group. Each such value provides the globally unique identifier of an information repository. This value is combined with LOCAL_ID and DOMAIN to form a UFID (unique federation identifier) value that is added to each element of the input group. If this parameter is omitted, and GUID is also omitted, the globally unique identifier is derived from the DOMAIN value. This parameter is optional.

LOCAL_ID
Specifies the repository-specific identifier of the object represented by an element of the input group. This value is combined with GUID and DOMAIN to form a UFID (unique federation identifier) value that is added to each element of the input group. This parameter may be specified more than once. If the number of values specified is less than the number of elements in the input group, then only those elements with assigned values have a UFID. If this parameter is omitted, and both LOCAL_ID_ATTRIBUTE and UFID are also omitted, then no UFID attribute is added to any elements of the input group. This parameter is optional.

LOCAL_ID_ATTRIBUTE

Specifies the name of an attribute contained in each element of the input group. Each such value provides the repository-specific identifier of the object represented by the element that contains it. This value is combined with GUID and DOMAIN to form a UFID (unique federation identifier) value that is added to each element of the input group. If this parameter is omitted, and both LOCAL_ID and UFID are also omitted, then no UFID attribute is added to any elements of the input group. This parameter is optional.

UFID

Directly specifies the value that is added as a UFID to each element of the input group. This parameter may be specified more than once. If the number of values specified is less than the number of elements in the input group, then only those elements with assigned values have a UFID. If this parameter is omitted, and both LOCAL_ID and LOCAL_ID_ATTRIBUTE are also omitted, then no UFID attribute is added to any elements of the input group. This parameter is optional.

UFID_ATTRIBUTE

Specifies the name of the attribute that is added to each element of the input group to identify each element's UFID. The default value of this parameter is `obid`. This parameter is optional.

Set-Metadata

DESCRIPTION

Registers metadata values for groups, for individual elements within a group, and for attributes within elements. This metadata can then be read by higher layer applications, which can affect the processing of group, element, or attribute data.

Note

The Set-Metadata webject does not generate a new output group. Instead, it modifies the group specified by its GROUP_IN parameter.

SYNTAX

```
<ie:webject name="Set-Metadata" type="GRP">
  <ie:param name="ATTRIBUTE" data="name_of_attribute"/>
  <ie:param name="ELEMENT" data="index_of_element"/>
  <ie:param name="GROUP_IN" data="input_group_name"/>
  <ie:param name="NAME" data="name_of_meta_attribute"/>
  <ie:param name="SCOPE" data="[GROUP | ELEMENT | ATTRIBUTE]"/>
  <ie:param name="VALUE" data="value_of_meta_attribute"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_ IN	SCOPE	VALUE
NAME	ELEMENT	
	ATTRIBUTE	

ATTRIBUTE

Identifies the attribute for which the metadata is set.

If SCOPE is ATTRIBUTE, then you must specify both the ELEMENT and ATTRIBUTE parameters in order to identify the attribute against which the metadata is set.

ELEMENT

Identifies the element for which the metadata is set. The value of the parameter can be a simple integer that specifies the index of the element within the group, or can be specified as `name=value` where `name` is the name of an attribute, and `value` is the corresponding value of the attribute. If a name and value pair are specified, the webject locates the first element that contains an attribute with the specified name and value, and it sets the metadata for that element.

The value of ELEMENT can also be specified as " * ". In this case, all elements of the group are selected.

If the SCOPE parameter is ELEMENT, then you must specify the ELEMENT parameter. If the SCOPE parameter is ATTRIBUTE, then you must specify both the ELEMENT and ATTRIBUTE parameters in order to identify the attribute for which the metadata is set.

GROUP_IN

Specifies the name of the group for which metadata is registered. This parameter is required.

NAME

Specifies the name of the metadata attribute that is set. Multiple NAME and VALUE parameter pairs can be specified in order to set multiple metadata attributes in a single call to the webject. This parameter is required.

VALUE

Specifies the metadata value associated with the corresponding metadata named in the NAME parameter. Multiple NAME and VALUE parameter pairs may be specified in order to set multiple metadata attributes in a single call to the webject.

The default for this parameter is the null character. This parameter is optional.

SCOPE

Specifies the metadata scope. Valid values for this parameter are the following:

- GROUP – Sets the metadata for the group as a whole.
- ELEMENT – Sets the metadata for a particular element within the group. If SCOPE is specified as ELEMENT, then the ELEMENT parameter must also be specified.
- ATTRIBUTE – Sets the metadata for an particular attribute of an element within the group. If SCOPE is specified as ATTRIBUTE, then both the ELEMENT and ATTRIBUTE parameters must also be specified.

The default value for this parameter is GROUP.

EXAMPLE

The following example JSP page registers metadata with a name of `meta_data` and a value of `met_value` on the group named "createdgroup" at the group level:

```
<%@page language="java" session="false"
       errorPage="XML_IEError.jsp" contentType="text/xml"%>

<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Set-Metadata" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="SCOPE"  data="GROUP"/>
  <ie:param name="NAME"   data="meta_name"/>
  <ie:param name="VALUE"  data="met_value"/>
</ie:webject>

<ie:webject name="Display-XML" type="DSP">
  <ie:param name="MODE" data="FULL"/>
</ie:webject>
```

The following `SetMetadataLevels.jsp` file sets metadata for all three SCOPE values and displays the XML output using Display-XML:

```
<%@page language="java" session="false" errorPage="../IEError.jsp"%>

<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="NAME=Sam Johnson:
          ADDRESS=1234 Main St.:EMAIL=sjohnson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=Harvy Anderson:
          ADDRESS=1234 Amber St.:EMAIL=handerson@somewhere.com"/>
  <ie:param name="ELEMENT" data="NAME=&lt;&gt;'&amp;&quot;:
          EMAIL=joconnor@somewhere.com"/>
  <ie:param name="GROUP_OUT" data="CREATE-RESULTS"/>
</ie:webject>

<ie:webject name="SeT-Metadata" type="GRP">
  <ie:param name="GROUP_IN" data="CREATE-RESULTS"/>
  <ie:param name="NAME" data="testattribute"/>
  <ie:param name="VALUE" data="attribute metadata"/>
  <ie:param name="SCOPE" data="ATTRIBUTE"/>
  <ie:param name="ELEMENT" data="*"/>
  <ie:param name="ATTRibute" data="name"/>
</ie:webject>

<ie:webject name="Set-Metadata" type="GRP">
  <ie:param name="GROUP_IN" data="CREATE-RESULTS"/>
  <ie:param name="NAME" data="testelement"/>
  <ie:param name="VALUE" data="element metadata"/>
  <ie:param name="SCOPE" data="ELEMENT"/>
  <ie:param name="ELEMENT" data="0"/>
</ie:webject>

<ie:webject name="Set-Metadata" type="GRP">
  <ie:param name="GROUP_IN" data="CREATE-RESULTS"/>
  <ie:param name="NAME" data="testgroup"/>
  <ie:param name="VALUE" data="group metadata"/>
  <ie:param name="SCOPE" data="GROUP"/>
</ie:webject>

<ie:webject name="Display-Xml" type="DSP">
  <ie:param name="Mode" data="FULL"/>
</ie:webject>
```

The XML output from executing the example is as follows:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
  - <Unknown-Class-Name NAME="CREATE-RESULTS" TYPE="Object" STATUS="0">
    - <wc:INSTANCE>
      - <NAME>
          Sam Johnson
       - <wc:Meta>
          <testattribute>attribute metadata</testattribute>
         </wc:Meta>
       </NAME>
       <ADDRESS>1234 Main St.</ADDRESS>
       <EMAIL>sjohnson@somewhere.com</EMAIL>
       - <wc:Meta>
          <testelement>element metadata</testelement>
         </wc:Meta>
      </wc:INSTANCE>
    - <wc:INSTANCE>
      - <NAME>
          Harvy Anderson
       - <wc:Meta>
          <testattribute>attribute metadata</testattribute>
         </wc:Meta>
       </NAME>
       <ADDRESS>1234 Amber St.</ADDRESS>
       <EMAIL>handerson@somewhere.com</EMAIL>
      </wc:INSTANCE>
    - <wc:INSTANCE>
      - <NAME>
          &lt;&gt;'&amp;&quot;
       - <wc:Meta>
          <testattribute>attribute metadata</testattribute>
         </wc:Meta>
       </NAME>
       <EMAIL>joconnor@somewhere.com</EMAIL>
      </wc:INSTANCE>
      - <wc:Meta>
          <Class>Unknown-Class-Name</Class>
          <testgroup>group metadata</testgroup>
          <Status>0</Status>
        </wc:Meta>
     </Unknown-Class-Name>
  </wc:COLLECTION>
```

Sort-Group

DESCRIPTION

Sorts the information in a group of objects by one or more attributes. For example, in a group containing employee names, numbers, and salaries, the Sort-Group webject can be used to specify an alphanumeric ordering of the information by either name, number, salary, or a combination of any or all of the three attributes.

SYNTAX

```
<ie:webject name="Sort-Group" type="GRP">
  <ie:param name="GROUP_IN" data="input_group_name"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
SORTBY		COMPARISON
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. Multiple values can be specified for this parameter. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name”. This parameter is optional.

COMPARISON

Describes how to compare the attribute values: either ALPHA for an alpha-numeric comparison or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. Multiple values can be specified for this parameter. This parameter is optional.

GROUP_IN

Specifies the group to be sorted. This parameter is required.

GROUP_OUT

Specifies the name of the sorted group. This parameter is required.

SORTBY

Identifies the field or column name to be used for sorting. Note that null values sort before all numbers and letters in ascending sorting and after all numbers and letters in descending sorting.

Multiple values can be specified for this parameter. If more values are specified for SORTBY than for SORTED, COMPARISON, or CASE_IGNORE, then the last value specified for SORTED, COMPARISON, or CASE_IGNORE is used against the remaining SORTBY values. This parameter is required.

SORTED

Describes how to order the output of the two groups: either ASC for ascending order of output or DESC for a descending order of output. The default of this parameter is ASC. Multiple values can be specified for this parameter. This parameter is optional.

EXAMPLE: SINGLE-COLUMN SORTING

The following example sorts the group named “createdgroup” by name, alphabetically, in ascending order, ignoring case, and places the resulting sorted data in an output group named “results”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- Sort the elements in a group. -->

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Sort-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="SORTBY" data="NAME"/>
  <ie:param name="CASE_IGNORE" data="TRUE"/>
  <ie:param name="COMPARISON" data="ALPHA"/>
  <ie:param name="SORTED" data="ASC"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

EXAMPLE: MULTI-COLUMN SORTING

The following Sort-Group webject example shows sorting on multiple columns.

First, a group is created, containing employee last names and department numbers.

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="DEPT=300:NAME=Smith"/>
  <ie:param name="ELEMENT" data="DEPT=300:NAME=Johnson"/>
  <ie:param name="ELEMENT" data="DEPT=200:NAME=Reilly"/>
  <ie:param name="ELEMENT" data="DEPT=100:NAME=Sinclair"/>
  <ie:param name="ELEMENT" data="DEPT=300:NAME=Michaels"/>
```

```
    <ie:param name="ELEMENT" data="DEPT=500:NAME=King"/>
    <ie:param name="GROUP_OUT" data="employees"/>
  </ie:webject>

  <ie:webject name="Display-Table" type="DSP"/>
```

The resulting output group named “employees” would display in the following form:

DEPT	NAME
300	Smith
300	Johnson
200	Reilly
100	Sinclair
300	Michaels
500	King

The group “employees” is then sorted by department number, in ascending order.

```
  <ie:webject name="Sort-Group" type="GRP">
    <ie:param name="GROUP_IN" data="employees"/>
    <ie:param name="SORTBY" data="DEPT"/>
    <ie:param name="SORTED" data="ASC"/>
    <ie:param name="GROUP_OUT" data="results"/>
  </ie:webject>

  <ie:webject name="Display-Table" type="DSP"/>
```

The output group named “results” would display in the following form:

DEPT	NAME
100	Sinclair
200	Reilly
300	Smith
300	Johnson
300	Michaels
500	King

Note

The employee names in Department 300 are in no particular order.

The group “employees” is then sorted by department number and name, both in ascending order.

```
  <ie:webject name="Sort-Group" type="GRP">
```

```
    <ie:param name="GROUP_IN" data="employees"/>
    <ie:param name="SORTBY" data="DEPT,NAME" delim=","/>
    <ie:param name="SORTED" data="ASC"/>
    <ie:param name="GROUP_OUT" data="results"/>
  </ie:webject>

  <ie:webject name="Display-Table" type="DSP"/>
```

The output group named "results" would display in the following form:

DEPT	NAME
100	Sinclair
200	Reilly
300	Johnson
300	Michaels
300	Smith
500	King

Note

The employee names in Department 300 are now listed in ascending alphabetical order.

The group "employees" is then sorted by department number, ascending, and name, descending.

```
  <ie:webject name="Sort-Group" type="GRP">
    <ie:param name="GROUP_IN" data="employees"/>
    <ie:param name="SORTBY" data="DEPT,NAME" delim=","/>
    <ie:param name="SORTED" data="ASC,DESC" delim=","/>
    <ie:param name="GROUP_OUT" data="results"/>
  </ie:webject>

  <ie:webject name="Display-Table" type="DSP"/>
```

The output group named "results" would display in the following form:

DEPT	NAME
100	Sinclair
200	Reilly
300	Smith
300	Michaels

DEPT	NAME
300	Johnson
500	King

Note

The employee names in Department 300 are now listed in reverse alphabetical order.

Subset-Group

DESCRIPTION

Uses pattern matching on single or multiple parameters to see if a string matches a specified pattern as a whole or to see if a substring within a string matches a specified pattern. Matching is done using regular expressions (part of the POSIX Standard). Case can be ignored.

SYNTAX

```
<ie:webject name="Subset-Group" type="GRP">
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="FILTER" data="string_pattern"/>
  <ie:param name="FILTER_MODE" data="[MATCH | NOMATCH]"/>
  <ie:param name="FILTER_TYPE" data="[IE | REGEXP]"/>
  <ie:param name="GROUP_IN" data="input_group_name"/>
  <ie:param name="GROUP_OUT" data=" output_group_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
FILTER		CASE_IGNORE
GROUP_IN		CLASS
GROUP_OUT		FILTER_MODE
		FILTER_TYPE

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored when searching for matches. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is "Unknown-Class-Name." This parameter is optional.

FILTER

Specifies the pattern to which a string or substring must match. The set of wildcard characters you can include in the pattern is determined by the type of filter you specify. For additional information, see the FILTER_TYPE parameter description.

This parameter is required.

FILTER_MODE

Specifies whether to pass the values that match the specified pattern or the values that do not match the specified pattern. Valid values are MATCH and NOMATCH. The default for this parameter is MATCH. This parameter is optional.

FILTER_TYPE

Specifies the type of filter to use in pattern matching. Valid values are IE for Info*Engine or REGEXP for regular expressions.

If IE is specified, the following characters are translated into the corresponding regular expressions:

IE	Regular Expression
?	.
*	.*
pattern	^pattern$

After the translation from IE characters to regular expressions is complete, then pattern matching is performed.

The IE characters listed in the previous table can be used when the required pattern is a relatively simple pattern. If a more complex pattern is required, specify REGEXP as the value for the FILTER_TYPE parameter and include the required regular expression in the FILTER parameter pattern.

The default for this parameter is IE. This parameter is optional.

GROUP_IN

Specifies the group from which to select a particular subset. This parameter is required.

GROUP_OUT

Specifies the name of the output group into which the subset is stored. This parameter is required.

EXAMPLE

The following example uses pattern matching to see if a string matches a specified pattern as a whole or to see if a substring within a string matches a specified pattern. The first Subset-Group webject passes items which match the specified FILTER, placing those items in an output group named “matched”. The second Subset-Group webject passes items which do not match the specified FILTER, placing those items in a output group named “nomatch”.

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Form a new group that is a subset of a group.-->

<ie:task uri="com/company/CreateGroup.xml"/>

<ie:webject name="Subset-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="FILTER" data="NAME='^J'"/>
  <ie:param name="FILTER_TYPE" data="REGEXP"/>
  <ie:param name="FILTER_MODE" data="MATCH"/>
  <ie:param name="CASE_IGNORE" data="TRUE"/>
  <ie:param name="CLASS" data="MATCHEDITEMS"/>
  <ie:param name="GROUP_OUT" data="matched"/>
</ie:webject>

<ie:webject name="Subset-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="FILTER" data="NAME='^J'"/>
  <ie:param name="FILTER_TYPE" data="REGEXP"/>
  <ie:param name="FILTER_MODE" data="NOMATCH"/>
  <ie:param name="CASE_IGNORE" data="TRUE"/>
  <ie:param name="CLASS" data="NONMATCHEDITEMS"/>
  <ie:param name="GROUP_OUT" data="nomatch"/>
</ie:webject>

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="Group_in" data="matched"/>
  <ie:param name="Group_in" data="nomatch"/>
</ie:webject>
```

Summarize-Groups

DESCRIPTION

Provides descriptive information about groups.

SYNTAX

```
<ie:webject name="Summarize-Groups" type="GRP">
  <ie:param name="CLASS" data="class"/>
  <ie:param name="GROUP_IN" data="group_in"/>
  <ie:param name="GROUP_OUT" data="group_out"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CLASS
GROUP_OUT		

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

GROUP_IN

Specifies the groups to be summarized. Multiple values can be specified for this parameter. This parameter is required.

GROUP_OUT

Specifies the name of the results of summarizing the input groups. Each node or row has the following attributes:

NAME – the name of the GROUP_IN

TYPE – the type of the group (Unknown, Object, Status, or Stream)

SIZE – currently 0

COUNT – the number of elements in the group

MIMETYPE – currently an empty string

MESSAGE – the message, if any, associated with the group

STATUS – the integer status associated with the group

This parameter is required.

EXAMPLE

The following example uses Summarize-Groups to provide descriptive information about the specified GROUP_IN and places the resulting data in an output group named “result”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Summarize the contents of a set of groups. -->

<ie:task uri="com/company/CreateGroup.xml"/>
<ie:task uri="com/company/CreateGroupHr.xml"/>

<ie:webject name="Summarize-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createhrgroup"/>
  <ie:param name="CLASS" data="SUMMARIZEGROUPS"/>
  <ie:param name="GROUP_OUT" data="result"/>
</ie:webject>
```

Translate-Group

DESCRIPTION

Allows translation of data from a specified Info*Engine data group in one or more of the following ways:

- Rename attributes within elements in the group;
- Translate data types of attribute values in the output group;
- Remove attributes from elements in the output group;
- Apply an XSL stylesheet to the Info*Engine data group specified on the GROUP_IN parameter.

This webject can be used for generalized schema translation.

SYNTAX

```
<ie:webject name="Translate-Group" type="GRP">
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COPY_UNTRANSLATED" data="[YES | NO]"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="DEFAULT_TABLE" data="group_name"/>
  <ie:param name="GROUP_IN" data="group_in"/>
  <ie:param name="GROUP_OUT" data="GROUP_OUT_name"/>
  <ie:param name="NAME_TRANS_TABLE" data="group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="TYPE_TRANS_TABLE" data="group_name"/>
  <ie:param name="XSL_PARAM" data="name_value_pair"/>
  <ie:param name="XSL_URL" data="url_of_xsl_source"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN	COPY_UNTRANSLATED	CLASS
GROUP_OUT	DEFAULT_TABLE	DBUSER
	NAME_TRANS_TABLE	PASSWD
	TYPE_TRANS_TABLE	XSL_PARAM
		XSL_URL

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name”. This parameter is optional.

COPY_UNTRANSLATED

Specifies the action to be taken against attributes found in GROUP_IN elements that are not matched by attributes of the same names defined in the NAME_TRANS_TABLE or TYPE_TRANS_TABLE parameters. If the value of the parameter is specified as YES, then the name of an attribute of a GROUP_IN element which does not match the name of an attribute in the NAME_TRANS_TABLE or TYPE_TRANS_TABLE is copied to the GROUP_OUT element unmodified. If COPY_UNTRANSLATED is not specified, its value defaults to NO, in which case any such attribute is not included in the GROUP_OUT element.

If you specify the COPY_UNTRANSLATED parameter, then you must also specify either the NAME_TRANS_TABLE or the TYPE_TRANS_TABLE parameter.

DBUSER

Specifies the username to be used to authenticate to XSL_URL. If the XSL templates to be used by the webject reside on a remote HTTP server, then this parameter should be used in conjunction with the PASSWD attribute.

This parameter is optional.

DEFAULT_TABLE

Specifies the name of a group that provides default attribute names and values to the GROUP_OUT group. The group is assumed to contain exactly one element. If more than one element is contained within the group, then the additional elements are ignored. Each attribute of the element specifies a default name and value.

After the NAME_TRANS_TABLE and TYPE_TRANS_TABLE parameters have been applied to each element of the GROUP_IN group to produce an element of the GROUP_OUT group, the GROUP_OUT element is checked for attributes having names matching attributes defined in the DEFAULT_TABLE. For each attribute defined in the DEFAULT_TABLE that does not have a matching name in the GROUP_OUT element, the attribute of the DEFAULT_TABLE is added to the GROUP_OUT element. For example, if the DEFAULT_TABLE contains an attribute named “Factory” and a value named “Unknown”, and after translation a GROUP_OUT element does not contain an attribute named “Factory”, then an attribute with name “Factory” and value “Unknown” are added.

If you specify the DEFAULT_TABLE parameter, then you must also specify either the NAME_TRANS_TABLE or the TYPE_TRANS_TABLE parameter.

GROUP_IN

Specifies the group to convert to XML and to which an XSL stylesheet is applied. This parameter is required.

GROUP_OUT

Specifies the name of the results of translating the input group. This parameter is required.

NAME_TRANS_TABLE

Specifies the name of a group that defines an attribute name translation table. The group is assumed to contain exactly one element. If more than one element is contained within the group, then the additional elements are ignored. The name of each attribute of the element identifies an attribute name to be translated. The value of each attribute specifies the new name. For example, if an attribute in the NAME_TRANS_TABLE group has the name “PartNo” and the value “PartNumber”, then every attribute named “PartNo” in the GROUP_IN group is renamed to “PartNumber” in the GROUP_OUT group.

NAME_TRANS_TABLE and TYPE_TRANS_TABLE can be specified together to change both the names and data types of attributes and their values. When both parameters are specified, it is important to remember that the attribute names specified by TYPE_TRANS_TABLE identify attributes in the GROUP_OUT group, after NAME_TRANS_TABLE has been applied. If TYPE_TRANS_TABLE specifies names of attributes that occur in the GROUP_IN group, but one or more of these are not translated using NAME_

TRANS_TABLE, then these attributes are copied to the GROUP_OUT group with their names unchanged, but their values translated using TYPE_TRANS_TABLE into different data types.

The NAME_TRANS_TABLE and TYPE_TRANS_TABLE parameters can be specified in addition to the XSL_URL parameter. In this case, the XSL-based translation is executed first; the NAME_TRANS_TABLE and/or TYPE_TRANS_TABLE based translations are then applied to the output group produced by the XSL-based translation. Thus, the attribute names specified in the NAME_TRANS_TABLE identify attributes of the group produced by the XSL-based translation, not attributes of the original GROUP_IN group. In all cases, the attribute names specified in TYPE_TRANS_TABLE identify attributes of the final GROUP_OUT group.

PASSWD

Specifies the password corresponding to DBUSER.

This parameter is optional.

TYPE_TRANS_TABLE

Specifies the name of a group that defines an attribute value data type translation table. The group is assumed to contain exactly one element. If more than one element is contained within the group, then the additional elements are ignored. The name of each attribute identifies the name of an attribute in the GROUP_OUT group whose values are to be translated. The value of each attribute specifies the desired data type. For example, if an attribute in the TYPE_TRANS_TABLE group has the name “Quantity” and the value “Integer”, then every value of every attribute named “Quantity” in the GROUP_OUT group is converted to the data type INTEGER. The data types currently supported are:

BYTE – 8-bit value.

DOUBLE – double precision floating point value.

FLOAT – single precision floating point value.

INTEGER – integer value with default range.

LONG – integer value with long range.

SHORT – integer value with short range.

STRING – character string value.

NAME_TRANS_TABLE and TYPE_TRANS_TABLE can be specified together to change both the names and data types of attributes and their values. When both parameters are specified, the attribute names specified by TYPE_TRANS_TABLE identify attributes in the GROUP_OUT group, after NAME_TRANS_TABLE has been applied. If TYPE_TRANS_TABLE specifies names of attributes that occur in the GROUP_IN group, but one or more of these are not translated using NAME_TRANS_TABLE, then these attributes are copied to the GROUP_OUT group with their names unchanged, but their values are translated using TYPE_TRANS_TABLE into different data types.

The NAME_TRANS_TABLE and TYPE_TRANS_TABLE parameters can be specified in addition to the XSL_URL parameter. In this case, the XSL-based translation is executed first. The NAME_TRANS_TABLE and/or TYPE_TRANS_TABLE based translations are then applied to the output group produced by the XSL-based translation. Thus, the attribute names specified in the NAME_TRANS_TABLE identify attributes of the group produced by the XSL-based translation, not attributes of the original GROUP_IN group. In all cases, the attribute names specified in TYPE_TRANS_TABLE identify attributes of the final GROUP_OUT group.

XSL_PARAM

Defines XSL parameters that are then passed to the XSL stylesheet named in the XSL_URL parameter. Enter the value for the XSL_PARAM parameter in the form `XSL_name=XSL_value`, where `XSL_name` is the name of a parameter in the XSL stylesheet and `XSL_value` is the value you want set for the parameter.

The default for XSL_PARAM is that no parameters are passed to the stylesheet. Multiple values can be specified for this parameter. This parameter is optional.

XSL_URL

Identifies the location of an XSL stylesheet to apply to the group specified in the GROUP_IN parameter. A relative URL or a fully qualified URL can be specified. Relative URLs are relative to the Info*Engine Server task template root. The XSL stylesheet should be constructed to apply to the Info*Engine XML format (the format shown in all examples in this guide, and produced by the Display-XML webject). The stylesheet must also produce Info*Engine XML format.

The NAME_TRANS_TABLE and TYPE_TRANS_TABLE parameters can be specified in addition to the XSL_URL parameter. In this case, the XSL-based translation is executed first; the NAME_TRANS_TABLE and/or TYPE_TRANS_TABLE based translations are then applied to the output group produced by the XSL-based translation.

Fully qualified URLs are dereferenced using **Auth-Map** context group data. The **Auth-Map** is searched for a username and password based on the domain name found in the fully qualified URL. For example, if the fully qualified URL is `http://machine.com/servlet/IE/createGroupData.xml`, the **Auth-Map** context group is searched for a username and password that has an INSTANCE name of `http://machine.com`. If a username and password are found, BASIC authentication information is used when accessing the URL. If no username and password are found, no authentication information is sent to the remote web server.

If the data value contains the `://` string, it is assumed to be a fully qualified internet URL. If the data value does not contain the `://` string, the stylesheet is assumed to be a local file relative to the current task template root directory.

This parameter is optional.

EXAMPLE

The following example translates the data from the group named “createdgroup” by renaming the attributes. The resulting data is placed in the group named “RenamedGroup”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Create a test Group -->

<ie:task uri="com/company/CreateGroup.xml"/>

<!-- Create attribute translation Group -->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT"
                  data="NAME=FullName:EMAIL=EmailAddress"/>
  <ie:param name="GROUP_OUT" data="NewNames"/>
</ie:webject>

<!-- Translate the attribute names -->

<ie:webject name="Translate-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="NAME_TRANS_TABLE" data="NewNames"/>
  <ie:param name="GROUP_OUT" data="RenamedGroup"/>
</ie:webject>

<!-- Translate some of the attribute names -->

<ie:webject name="Translate-Group" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="NAME_TRANS_TABLE" data="NewNames"/>
  <ie:param name="COPY_UNTRANSLATED" data="YES"/>
  <ie:param name="GROUP_OUT" data="RenamedGroup"/>
</ie:webject>
```

Union-Groups

DESCRIPTION

Identifies the union of two groups and places the results in a new group. For example, group A contains the elements u, v, and x, and group B contains the elements x, y, and z. The Union-Groups webject identifies the elements in either group A or group B or both and uses the results to form group C. Group C, in this example, would contain the elements u, v, x, y, and z.

SYNTAX

```
<ie:webject name="Union-Groups" type="GRP">
  <ie:param name="CASE_IGNORE" data="[ TRUE | FALSE]"/>
  <ie:param name="CLASS" data=" class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="GROUP_IN" data="input_groups"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
  <ie:param name="UNIONBY" data="attribute"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
UNIONBY		COMPARISON
		SORTBY
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this optional parameter is “Unknown-Class-Name.”

COMPARISON

Describes how to compare the two groups: either ALPHA for an alphanumeric comparison or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. This parameter is optional.

GROUP_IN

Specifies the names of the two groups to be used in computing a union.

To specify two group names, you can include two lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
```

This parameter is required.

GROUP_OUT

Specifies the name of the results of computing the union of the two groups. This parameter is required.

SORTBY

Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED

Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

UNIONBY

Identifies the field or column name to be used for comparison. This parameter is required.

EXAMPLE

The following example finds the union of the two specified input groups by the NAME attribute, compares them alphabetically and places them in an output group named “results”:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Form a new group from the Union of multiple groups -->

<ie:task uri="com/company/CreateGroup.xml"/>
<ie:task uri="com/company/CreateGroupA.xml"/>

<ie:webject name="Union-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createdgroupa"/>
```

```
    <ie:param name="UNIONBY" data="NAME"/>
    <ie:param name="COMPARISON" data="ALPHA"/>
    <ie:param name="GROUP_OUT" data="results"/>
    <ie:param name="CLASS" data="ALLEMPLOYEES"/>
  </ie:webject>
```

XOR-Groups

DESCRIPTION

Identifies the symmetric difference or the 'exclusive or' of two groups. For example, group A contains the elements u, v, and x, and group B contains the elements x, y, and z. The XOR-Groups webject identifies the unique elements of groups A and B and uses the results to form group C. All elements that appear in either group A or group B, but not in both groups, are returned. Group C, in this example, would contain the elements u, v, y, and z.

SYNTAX

```
<ie:webject name="XOR-Group" type="GRP">
  <ie:param name="CASE_IGNORE" data="[TRUE | FALSE]"/>
  <ie:param name="CLASS" data="class"/>
  <ie:param name="COMPARISON" data="[ALPHA | NUMERIC]"/>
  <ie:param name="GROUP_IN" data="input_groups"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="SORTBY" data="attribute"/>
  <ie:param name="SORTED" data="[ASC | DESC]"/>
  <ie:param name="XORBY" data="attribute"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
GROUP_IN		CASE_IGNORE
GROUP_OUT		CLASS
XORBY		COMPARISON
		SORTBY
		SORTED

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case is ignored. If FALSE is specified, then case is significant. The default for this parameter is FALSE. This parameter is optional.

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, then the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

COMPARISON

Describes how to compare the two groups: either ALPHA for an alpha-numeric comparison or NUMERIC for a strictly numeric comparison. The default for this parameter is ALPHA. This parameter is optional.

GROUP_IN

Specifies the names of the two groups to be used in computing the symmetric difference.

To specify two group names, you can include two lines with different values for the GROUP_IN parameter. For example:

```
<ie:param name="GROUP_IN" data="group1"/>
<ie:param name="GROUP_IN" data="group2"/>
```

This parameter is required.

GROUP_OUT

Specifies the name of the results of computing the symmetric difference. This parameter is required.

SORTBY

Specifies the name of the attribute on which the sorting is done. If you do not include this parameter, the results are not sorted. This parameter is optional.

SORTED

Determines how values in the resulting group are sorted. The attribute named in the SORTBY parameter determines which values are sorted. Specify ASC to sort in ascending order or specify DESC to sort in descending order. The default for this parameter is ASC. This parameter is optional.

XORBY

Identifies the attribute name to be used for comparison. If the groups being compared contain elements with attributes of the same name, then one XORBY value can be specified. Otherwise, you must specify two XORBY parameters to provide the names of the attributes from each respective group that are compared.

The attribute name placed in the output group is the attribute name of the first XORBY parameter used.

This parameter is required.

EXAMPLE

The following example finds the `exclusive or` of the two specified input groups, and places the resulting data in an output group named “results”.

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<!-- Form a new group from the "exclusive or" of
                                              two groups -->

<ie:task uri="examples/CreateGroup.xml"/>
<ie:task uri="examples/CreateGroupA.xml"/>

<ie:webject name="XOR-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="createdgroup"/>
  <ie:param name="GROUP_IN" data="createdgroupa"/>
  <ie:param name="XORBY" data="NAME"/>
  <ie:param name="GROUP_OUT" data="results"/>
  <ie:param name="CLASS" data="partial"/>
</ie:webject>
```

Management Webjects

The following webjects provide some common functions, such as getting properties, mapping credentials, and throwing exceptions, that can be useful in managing your JSP pages or tasks:

- Dispatch-Tasks on page 417
- Generate-WSDL on page 422
- Get-Properties on page 425
- Get-Resource on page 427
- List-Repositories on page 428
- Lookup-Services on page 429
- Map-Credentials on page 431
- Query-Schema on page 433
- Query-TypeHierarchy on page 434
- Throw-Exception on page 436
- Write-Log on page 438

All management webjects use the `MGT` **type** attribute value in the **webject** tag.

Dispatch-Tasks

DESCRIPTION

Selects a task that is capable of executing a specified action against objects of the specified type residing at the specified location. The webject facilitates the development of applications that operate upon information distributed across multiple hosts or comprised of multiple object types.

SYNTAX

```
<ie:webject name="Dispatch-Tasks" type="MGT">
 <ie:param name="ACCESS" data="http|soap|internal"/>
  <ie:param name="ACTION" data="action_name"/>
  <ie:param name="CLIMBER" data="class_name"/>
  <ie:param name="DEFAULT_DOMAIN" data="domain_name"/>
  <ie:param name="DEFAULT_TYPE" data="type_name"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="GROUP_VDB" data="group_name"/>
  <ie:param name="ID_ATTRIBUTE" data="attr_name"/>
  <ie:param name="ID_META_ATTRIBUTE" data="meta_name"/>
  <ie:param name="MAX_CONCURRENT" data="integer"/>
  <ie:param name="PARAM" data="name=value"/>
  <ie:param name="TASKS" data="group_name_or_url"/>
  <ie:param name="TYPE_ATTRIBUTE" data="attr_name"/>
  <ie:param name="TYPE_META_ATTRIBUTE" data="meta_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
ACTION	ACCESS	CLIMBER
GROUP_IN	ID_ATTRIBUTE	GROUP_VDB
TASKS	ID_META_ATTRIBUTE	MAX_CONCURRENT
	TYPE_ATTRIBUTE	PARAM
	TYPE_META_ATTRIBUTE	

ACCESS

Specifies the required access configuration of the task that is called as a result of this Dispatch-Tasks invocation. This parameter value is only consulted if the value of the ACTION parameter has been supplied from an external client. It behaves identically to the **access** attribute on a task `page` directive. The value can be a pipe-delimited list of access values. If no value is specified then the default task access of the system is used. For more information about the **access** attribute, see page Directive on page 250.

ACTION

Specifies the name of the logical action that is performed by the tasks that are selected and executed. ACTION is the primary selector used in choosing the tasks to be executed. This parameter is required.

CLIMBER

Identifies the name of a Java class which climbs data type hierarchies in search of task delegates. If specified, the parameter continues to be called by the webject until it either succeeds in finding the appropriate task delegate or reaches the top of the hierarchy. If the top of the hierarchy is reached without finding a task delegate, an exception is thrown. This parameter is optional.

DEFAULT_DOMAIN

Specifies a default domain to use instead of deriving the domain from the name of the local virtual machine in cases where objects don't have attributes or metadata that identifies their domains.

DEFAULT_TYPE

Specifies a default type name to use instead of `com.ptc.object` in cases where objects don't have attributes of metadata that identifies their types.

GROUP_IN

Specifies the name of the group containing the object upon which the logical action specified by ACTION is taken. Multiple values can be specified for this parameter, resulting in more than one group of objects upon which to apply the ACTION. This parameter is required.

GROUP_VDB

Provides the names of one or more groups that are provided to each of the called tasks in their VDBs. These names populate each of the called tasks' VDBs with an initial set of groups other than the groups specified as GROUP_IN parameters.

ID_ATTRIBUTE

Specifies the name of the attribute that every element of the input group contains and that specifies each respective element's unique object identifier. For example, if the value of this parameter is OBID, then it is assumed that every element of the input group contains an attribute named OBID, and that the value of each such attribute identifies the unique object identifier of the element containing it.

If specified, ID_ATTRIBUTE takes precedence over the ID_META_ATTRIBUTE. If neither ID_ATTRIBUTE nor ID_META_ATTRIBUTE is specified, an internal API is called to obtain the element's unique object identifier. If an element is a Windchill type instance, then the API returns the Windchill type identifier of the object.

If either ID_ATTRIBUTE or ID_META_ATTRIBUTE is specified and fail to resolve an element's unique object identifier an internal API is used to obtain the unique object identifier.

ID_META_ATTRIBUTE

Specifies the name of the metadata item that every element of the input group contains and that specifies each respective element's unique object identifier. For example, if the value of this parameter is `com.infoengine.obid`, then it is assumed that every element of the input group contains a metadata item named `com.infoengine.obid`, and the value of each such attribute identifies the unique object identifier of the element associated with it.

If both ID_META_ATTRIBUTE and ID _ATTRIBUTE are specified, then ID_ATTRIBUTE takes precedence over the ID_META_ATTRIBUTE. If neither ID_ATTRIBUTE nor ID_META_ATTRIBUTE is specified, an internal API is called to obtain the element's unique object identifier. If an element is a Windchill type instance, then the API returns the Windchill type identifier of the object.

If either ID_ATTRIBUTE or ID_META_ATTRIBUTE is specified and fails to resolve an element's unique object identifier, an internal API is used to obtain the unique object identifier.

MAX_CONCURRENT

Specifies the maximum number of selected tasks that the webject is allowed to execute concurrently. If this parameter is not specified and multiple tasks are selected, they are executed sequentially. The default for this parameter is 1. This parameter is optional.

PARAM

Specifies additional parameters to be passed to the tasks that are selected and executed. Each value of this parameter is specified as a `name=value` pair, where `name` is the name of the additional parameter to be passed to each selected task, and `value` is the value of the parameter.

If PARAM is not specified, no additional parameters are passed to the selected task. Multiple values may be specified for this parameter. This parameter is optional.

TASKS

Specifies either an LDAP URL or the name of a group.

If an LDAP URL is specified, then the URL identifies a directory server and the node within the associated directory that establishes the root of a tree of task definitions. Tasks are selected from the tree on the basis of action name (from the ACTION parameter), object type (obtained per object based on TYPE_ATTRIBUTE or TYPE_META_ATTRIBUTE), and object location (obtained from the domain component of the object identifier of each object based on ID_ATTRIBUTE or ID_META_ATTRIBUTE).

If a group name is specified, then the group represents a table of task definitions. Each element of the group specifies one task definition, and each such definition is assumed to contain attributes named the following:

- ACTION – Specifies the logical name of the action supported by this definition. This attribute is compared to the ACTION parameter of the webject. If they do not match, then the definition does not apply to this invocation of the webject.
- TYPE – Specifies the object type to which this definition applies. This attribute is compared to the object type obtained from each element of the input group. If they do not match, then the definition does not apply to the element. If the value of this attribute is "*", the definition applies to all object types.
- DOMAIN – Specifies the fully qualified domain name to which this definition applies. This attribute is compared to the domain component obtained from the unique object identifier of each element of the input group. If they do not match, then the definition does not apply to the element. If the value of this attribute is "*", the definition applies to all domains.
- TASK – Specifies the URL of the Info*Engine task that implements the action associated with this definition.
- PROCESSOR – Specifies the name of the Info*Engine task processor that is capable of executing the task identified by the TASK attribute. If this attribute is not specified in a task definition, the task implementation identified by the TASK attribute can be accessed and executed anywhere.

  This parameter is required.

TYPE_ATTRIBUTE

Specifies the name of the attribute that every element of the input group contains and that specifies each respective element's object type. For example, if the value of this parameter is CLASS, then it is assumed that every element of the input group contains an attribute named CLASS, and the value of each such attribute identifies the object type of the element containing it.

If both TYPE_ATTRIBUTE and TYPE_META_ATTRIBUTE are specified, then TYPE_ATTRIBUTE takes precedence over the TYPE_META_ATTRIBUTE. If neither TYPE_ATTRIBUTE nor TYPE_META_ATTRIBUTE is specified, then an internal API is called to obtain the type identifier. If the element is a Windchill type instance, this API returns the Windchill type identifier of the object.

If either TYPE_ATTRIBUTE or TYPE_META_ATTRIBUTE is specified and fail to resolve an element's type identifier an internal API is used to obtain the type identifier.

TYPE_META_ATTRIBUTE

Specifies the name of the metadata item that every element of the input group contains and that specifies each respective element’s object type. For example, if the value of this parameter is `com.infoengine.objectType`, then it is assumed that every element of the input group has an associated metadata item named `com.infoengine.objectType`, and the value of each such metadata object identifies the object type of the element associated with it.

If both TYPE_META_ATTRIBUTE and TYPE _ATTRIBUTE are specified, then TYPE _ATTRIBUTE takes precedence over TYPE_META_ ATTRIBUTE. If neither TYPE_ATTRIBUTE nor TYPE_META_ ATTRIBUTE is specified, then an internal API is called to obtain the type identifier. If the element is a Windchill type instance, this API returns the Windchill type identifier of the object.

If either TYPE_ATTRIBUTE or TYPE_META_ATTRIBUTE is specified and fail to resolve an element’s type identifier an internal API is used to obtain the type identifier.

EXAMPLE

```
<%@page language="java" session="true" errorPage="IEError.jsp" import="java.util.
Vector,java.util.Enumeration,com.infoengine.SAK.*" %>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>
<html>
<head>
<title>Dispatch Tasks</title>
</head>
<body bgcolor="#FFFFFF">
<ie:getService varName="ie"/>
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="group_out" data="tasks"/>
  <ie:param name="delimiter" data=":"/>
  <ie:param name="element"
                data="action=query:type=part:domain=s1.ptc.com:task=infoengine/
                                                      com/company/CreateGroup.xml"/>
  <ie:param name="element"
                data="action=query:type=part:domain=s2.ptc.com:task=infoengine/
                                                    com/company/CreateGroupA.xml"/>
  <ie:param name="element"
                data="action=query:type=part:domain=s3.ptc.com:task=infoengine/
                                                     com/company/JdbcQueryEmp.xml"/>
</ie:webject>
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="group_out" data="parts"/>
  <ie:param name="delimiter" data=":"/>
  <ie:param name="element" data="ufid=part1@s1.ptc.com:class=part:number=1"/>
  <ie:param name="element" data="ufid=part2@s2.ptc.com:class=part:number=2"/>
  <ie:param name="element" data="ufid=part3@s3.ptc.com:class=part:number=3"/>
  <ie:param name="element" data="ufid=part4@s1.ptc.com:class=part:number=4"/>
  <ie:param name="element" data="ufid=part5@s2.ptc.com:class=part:number=5"/>
  <ie:param name="element" data="ufid=part6@s3.ptc.com:class=part:number=6"/>
  <ie:param name="element" data="ufid=part7@s1.ptc.com:class=part:number=7"/>
  <ie:param name="element" data="ufid=part8@s2.ptc.com:class=part:number=8"/>
  <ie:param name="element" data="ufid=part9@s3.ptc.com:class=part:number=9"/>
</ie:webject>
<% long start = System.currentTimeMillis (); %>
<ie:webject name="Dispatch-Tasks" type="MGT">
  <ie:param name="tasks" data="tasks"/>
  <ie:param name="group_in" data="parts"/>
  <ie:param name="max_concurrent" data="$(@FORM[]threads[])" default="10"/>
  <ie:param name="action" data="$(@FORM[]action[])" default="query"/>
  <ie:param name="type_attribute" data="class"/>
  <ie:param name="id_attribute" data="ufid"/>
  <ie:param name="param" data="group_out=results"/>
  <ie:param name="param" data="jdbcAdapter=$(@FORM[]jdbcAdapter[])"
     default="jdbcAdapter"/>
</ie:webject>
<% long duration = System.currentTimeMillis () - start; %>
<h2>Execution time: <%= duration %> msec</h2><br>
<ie:webject name="Display-Object" type="DSP">
  <ie:param name="group_in" data="results"/>
  <ie:param name="border" data="1"/>
  <ie:param name="display_attribute_name" data="true"/>
</ie:webject>
</body>
</html>
```

Generate-WSDL

DESCRIPTION

Generates Web Service Definition Language (WSDL) from specified Info*Engine tasks for use by SOAP clients. WSDL is an XML-based language used to define client/server interfaces.

For the Info*Engine SOAP service to perform in a predictable manner, you must bind to the service using the HTTP URL specified in the SOAP port exposed in the generated WSDL, since it contains additional query parameters to pass to the underlying SOAP service.

The generated WSDL contains SOAP information specific to the SOAP RPC servlet that allows it to properly route requests. The generated WSDL contains a single service named **IESoapServlet** bound to a port named **IESoapPort**, and describes all methods and method signatures supported for the class specified in the CLASS parameter. For more information on SOAP, see Simple Object Access Protocol (SOAP) Web Service on page 17.

The tasks from which WSDL is generated must be prefaced with special comment lines that define their parameters and results (similar to Javadoc comment syntax). The webject parses all of the tasks found at the specified location and uses the special comment lines within them to generate WSDL. Depending on which webject parameters are specified, the WSDL can be written back as a BLOB or added to the VDB as a group.

SYNTAX

```
<ie:webject name="Generate-WSDL" type="MGT">
  <ie:param name="CLASS" data="class_name"/>
  <ie:param name="GROUP_OUT" data="output_group_name"/>
  <ie:param name="REPOSITORY" data="repository"/>
  <ie:param name="REPOSITORY_TYPE" data="repository_type"/>
  <ie:param name="SOAP_URI" data="uri"/>
  <ie:param name="STYLE" data="rpc"/>
  <ie:param name="TASKS" data="task_location"/>
  <ie:param name="VERSION" data="1.0"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
CLASS	GROUP_OUT	REPOSITORY
	SOAP_URI	REPOSITORY_TYPE
		STYLE
		TASKS
		VERSION

CLASS

Specifies the Java class with which the WSDL methods are associated. The value specified must correspond to an existing type identifier definition found in the supporting LDAP directory. The REPOSITORY or REPOSITORY_ TYPE parameters can be used to direct the webject to the correct location within LDAP to discover the type identifier. This parameter is required.

GROUP_OUT

Specifies the name of the output group to create. If GROUP_OUT is specified, then the output is added to the VDB as a group with the specified name. The group contains one element for each command delegate exposed by the type identifier corresponding to the CLASS parameter. Each group element contains one attribute per task parameter. The parameter name becomes the attribute name. The parameter type become the attribute value. Method name and task are stored as metadata on each element with the identifiers com.infoengine.soap.wsdl.methodName and com.infoengine.soap.wsdl.task respectively. If comments are specified on either the entire task or individual parameters the comments are stored as metadata with the identifier com.infoengine.soap.wsdl.comment. Task comments are stored as metadata on the corresponding element. Parameter comments are stored as metadata on the corresponding attribute. If GROUP_OUT is not specified, then SOAP_URI must be specified.

REPOSITORY

Specifies the name of the repository whose repository type contains the required type identifier and task delegate definitions. For example, host.myCompany.com, which represents an installed instance of Info*Engine with a repository type of com.ptc.windchill. If this parameter is not specified the repository type is generated based on the virtual machine name of the running JVM. For example, a virtual machine name of com.myCompany.host.server results in a repository of host.myCompany.com. This parameter is optional.

REPOSITORY_TYPE

Specifies the name of the repository type that contains the required type identifier and task delegate definitions. For example, com.ptc.windchill. If this parameter is not specified, then the behavior described with the REPOSITORY parameter is carried out to attempt to dynamically discover the appropriate repository type. This parameter is optional.

SOAP_URI

Identifies the URI of the SOAP service used to route requests. Only fully-qualified URIs should be specified for this parameter. For example:

```
http://localhost/Windchill/servlet/RPC
```

where RPC is an instance of com.infoengine.soap.SoapRPCRouter. If this parameter is specified, the WSDL is generated and returned to the client as a BLOB. If GROUP_OUT is not specified, SOAP_URI must be specified.

STYLE

Acceptable values are rpc or document. This parameter governs how the generated WSDL represents Info*Engine SOAP services, and how those services behave when it comes to processing requests and generating responses. If the value is rpc, the generated WSDL is Remote Procedure Call (rpc)-style SOAP. If the value is document, the generated WSDL is document-style SOAP. The default value is rpc. This parameter is only processed if the VERSION parameter is set to 1.1. This parameter is optional.

Since the defaults for VERSION and STYLE are 1.0 and rpc respectively, this means that by default Info*Engine SOAP services are rpc-encoded SOAP services. If you want Info*Engine SOAP to behave rpc-literal, you must specify `rpc` and `1.1` for STYLE and VERSION respectively. If you want Info*Engine SOAP to behave document-literal, specify `document` and `1.1` respectively.

TASKS

Specifies the location, relative to the configured task root, of the tasks to be externalized to the SOAP client using WSDL. This parameter is optional.

VERSION

Acceptable values are `1.0` or `1.1`. This parameter governs whether the generated WSDL is SOAP encoded or literal. The default value is `1.0`, which results in encoded SOAP. This parameter is select with the STYLE parameter. The STYLE parameter is only processed if VERSION is `1.1`. This parameter is optional.

Since the defaults for VERSION and STYLE are `1.0` and `rpc` respectively, this means that by default Info*Engine SOAP services are rpc-encoded SOAP services. If you want Info*Engine SOAP to behave rpc-literal, specify `rpc` and `1.1` for STYLE and VERSION respectively. If you want Info*Engine SOAP to behave document-literal, specify `document` and `1.1` respectively.

EXAMPLE

The following Generate-WSDL example generates WSDL from Info*Engine tasks. The REPOSITORY parameter is generated based on the running virtual machine name (as described in the REPOSITORY parameter description). The WSDL methods are associated with the Java class named `com.infoengine.soap`. The request is routed through the SOAP service specified in SOAP_URI, and the generated WSDL is returned to the client as a BLOB.

```
<ie:webject name="Generate-WSDL" type="MGT">
  <ie:param name="CLASS" data="com.infoengine.soap"/>
  <ie:param name="SOAP_URI"
    data="http://host/Windchill/servlet/RPC"/>
</ie:webject>
```

Get-Properties

DESCRIPTION

Creates a group from a Java property resource.

SYNTAX

```
<ie:webject name="Get-Properties" type="MGT">
  <ie:param name="ATTRIBUTE" data="name"/>
  <ie:param name="GROUP_OUT" data="group_name"/>
  <ie:param name="SOURCE" data="uri_or_file_pathname"/>
```

```
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		ATTRIBUTE
		GROUP_OUT
		SOURCE

ATTRIBUTE
Specifies the names of the properties to fetch and place in the output group. If ATTRIBUTE is not specified, all of the properties from the specified SOURCE are placed in the output group. This parameter is optional.

GROUP_OUT
Specifies the name of the group to create that contains the queue identifiers of the objects in the message queue.

The GROUP_OUT contains exactly one element. Each attribute of the element represents one property name/value pair obtained from the property source. The name of the attribute is the same as the name of the corresponding property, and the value of the attribute is the same as the value of the property.

This parameter is optional.

SOURCE
Specifies the location of the properties that are returned by the Get-Properties webject. It may be specified as a relative or absolute URI:

- Relative URIs reference files that reside under the root file system directory that is defined for the local Info*Engine task processor
- Absolute URIs reference files that reside in the local file system, reside on a remote HTTP server, or are referenced through an accessible LDAP directory.

It may also be specified as a system filepath referencing a properties file such as `wt.properties`. The default for this optional parameter is to return the properties defined by the **java.lang.System** class in the GROUP_OUT.

EXAMPLE

The following Get-Properties example JSP page gets properties from a predetermined source and returns those properties in a group that you have named:

```
<%@page language="java" session="false"
                                errorPage="../IEError.jsp"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>


<html>
<head><title>Get Properties</title></head>
```

```
<body>

<ie:webject name="Get-Properties" type="MGT">
  <ie:param name="SOURCE" data="$(@FORM[]source[0])"
                                                 default=""/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[0])"
                                                 default=""/>
</ie:webject>

<ie:webject name="DISPLAY-TABLE" type="DSP"/>
</body>
</html>
```

To actually run this example, you would need to provide a form where the source and **group_out** variables are identified.

Get-Resource

DESCRIPTION

Creates an Info*Engine group from the localized strings in a Java resource bundle. The language of the strings in the created group is determined by the language setting of a client's web browser.

The Get-Resource webject is used in conjunction with the Display-Resource webject to support insertion of localized text into web pages.

SYNTAX

```
<ie:webject name="Get-Resource" type="MGT">
  <ie:param name="BUNDLE" data="resource_bundle"/>
  <ie:param name="GROUP_OUT" data="group_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
BUNDLE		GROUP_OUT

BUNDLE
Identifies the Java class base resource bundle from which localized strings are to be retrieved. This parameter is required.

GROUP_OUT
Specifies the name of an Info*Engine group in which to store retrieved localized strings. The default for this parameter is `bundle`. This parameter is optional.

EXAMPLE

The following Get-Resource example JSP page creates a group using localized strings from the specified resource bundle:

```
<%@page language="java"
  session="false"
  errorPage="../IEError.jsp"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie" %>

<html>
<head><title>Get Properties</title></head>
<body>

<ie:webject name="Get-Resource" type="MGT">
  <ie:param name="BUNDLE"
                   data="com.infoengine.util.IEResource"/>
</ie:webject>

<ie:webject name="Display-Table" type="DSP">
  <ie:param name="ATTRIBUTE" data="$(@FORM[]key[0])"
                                  default="20" delim=","/>
</ie:webject>

</body>
</html>
```

The key variable can be specified in a form, or the specified default value is used.

List-Repositories

DESCRIPTION

Lists Info*Engine repository entries. The output group contains one element per repository whose contents are the LDAP attributes for that repository. In addition, each element contains a **dn** attribute containing the distinguished name of the repository entry, and a **domain** attribute containing the domain name associated with the repository that can be used for task delegation.

SYNTAX

```
<ie:webject name="List-Repositories" type="MGT">
 <ie:param name="FILTER" data="ptcRepositoryType=com.ptc.windchill" />
 <ie:param name="FILTER_MODE" data="[MATCH | NOMATCH]" />
 <ie:param name="GROUP_OUT" data="repositories" />
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		FILTER
		FILTER_MODE
		GROUP_OUT

FILTER

Specifies the filter used to return a subset of the existing repositories. The format of the filter is `<attribute>=<regular expression>` where *<attribute>* is an LDAP attribute (such as `ptcRepositoryType` or `ptcSupportingAdapter`), and *<regular expression>* is a regular expression to match attribute values against. This parameter is optional. The default value for this parameter is `ptcRepositoryType=.*` which matches all repositories with a **ptcRepository** attribute set.

FILTER_MODE

Specifies whether the filter should match or not match. Specifying a value of MATCH means that if the repository entry matches the expression that had been specified by the filter, that expression are added to the output group. This parameter is optional. The default value for this parameter is MATCH.

GROUP_OUT

Specifies the name of the output group to create. This parameter is optional. The default value for this parameter is `repositories`.

Lookup-Services

DESCRIPTION

This webject performs directory searches to find Info*Engine services. The resulting group contains one element per service found. Each element may contain the following attributes:

- **dn**
- **description**
- **ptcRuntimeServiceName**
- **ptcServiceName**
- **ptcServiceClassName**
- **ptcMetaType**
- **ptcServiceAddress**
- **ptcObjectSerializationType**
- **ptcCoresidentService**

(The contents of the **ptcApplicationService objectClass** plus distinguished name)

Since the **ptcApplicationService objectClass** only requires **ptcServiceName**, **dn** and **ptcServiceName** are found in all elements, and the presence of the others is dependent on configuration of each specific service.

The defaults values are:

CLASS - none

FILTER - `(objectclass=ptcApplicationService)` (all available service entries)

GROUP_OUT -SERVICES

SYNTAX

```
<ie:webject name="Lookup-Services" type="MGT">
  <ie:param name="GROUP_OUT" data="group name"/>
  <ie:param name="FILTER" data="LDAP search filter"/>
  <ie:param name="CLASS" data="class name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		CLASS
		FILTER
		GROUP_OUT

CLASS

Specifies the type of the objects contained in the output group named by the GROUP_OUT parameter. For example if a webject specifies `CLASS= MyClassName` and `GROUP_OUT=data_1`, the XML representation of the output group contains the following tags:

```
<MyClassName NAME="data_1" TYPE="Object" STATUS="0">
</MyClassName>
```

The default for this parameter is “Unknown-Class-Name.” This parameter is optional.

FILTER

This parameter allows the user to specify an LDAP search filter such as `(ptcServiceName=com.company.host.server)`. The default value is `(objectclass=ptcApplicationService)`, which returns all services within the Naming Service’s search base (the search is always performed within the VMs Naming Service search base).

GROUP_OUT

This parameter allows the user to specify the name of the output group. The default value is SERVICES. The output group only contains services (LDAP directory entries who specify `ptcApplicationService` as an **objectClass**). The output group contains one element per service found. Each element contains at a minimum the **dn** and **ptcServiceName** attributes. Each element may also contain the following attributes if they have values set in the corresponding service: **ptcRuntimeServiceName**, **ptcServiceClassName**, **ptcMetaType**, **ptcServiceAddress**, **description**, **ptcCoresidentService**, **ptcObjectSerializationType**. The following attributes may be multivalued if specified: **ptcServiceName**, **ptcServiceAddress**, **ptcCoresidentService**.

Map-Credentials

DESCRIPTION

Reads a file or executes a task to establish a group as the credentials map for the task in which the webject executes (or does both). A credentials map is a **Auth-Map** context group that provides authentication information used by adapters in establishing connections to back-end information systems. Each element of a credentials map provides a username and associated credentials that are used in connecting to a specific back-end system.

The webject can be specified explicitly at any point in a task to create or change the credentials map that affects the remainder of the webjects in the task. If the `.credentialsMapper` configuration property is set, Map-Credentials is also called implicitly at the beginning of every task to create the initial credentials map for the task. If a default credentials mapping task has not been configured, Map-Credentials webject is not called implicitly.

For additional information about credentials mapping, see Credentials Mapping on page 125.

SYNTAX

```
<ie:webject name="Map-Credentials" type="MGT">
  <ie:param name="FILES" data="dir_path"/>
  <ie:param name="TASK" data="uri"/>
  <ie:param name="USERNAME" data="user_name"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	FILES	USERNAME
	TASK	

FILES

Specifies the system filepath directory in which each file is assumed to contain credentials mapping information for a specific user. The USERNAME parameter is concatenated to the FILES parameter to produce the path name of one such file. The resulting file name is formatted as follows:

`FILES_parameter_value/USERNAME_parameter_value`

The file is then read and parsed to produce a credentials map. Each line of the file is assumed to be of the form:

```
instance:dbuser:passwd
```

where:

- *instance* specifies the name of an Info*Engine adapter or the instance names used with the WES or MSG webjects.
- *dbuser* specifies the username.
- *passwd* specifies the credentials that are provided to the adapter for creating connections to the associated back-end information system.

If both FILES and TASK are specified, file-based mapping is performed first, then the credentials mapping task is executed. This allows some base or default mapping information to be specified using files, then augmented or overridden by the task.

If the FILES parameter is omitted, file-based mapping is not performed. In this case, the credentials map is derived only from the output of the task specified by the TASK parameter. The FILES parameter must be specified if the TASK parameter is omitted.

TASK

Specifies the URI of an Info*Engine task that produces an **Auth-Map** group that is used as a credentials map. For example, the credential mapping task could do one of the following:

- Allow a user to explicitly authenticate to Info*Engine once, and then Info*Engine automatically authenticates the user to other enterprise information systems.
- Enable role-based access to network resources by identifying the role played by a particular user, and then creating the output group containing the authentication information shared by users who play the same role.

If both FILES and TASK are specified, file-based mapping is performed first, then the credentials mapping task is executed. This allows some base or default mapping information to be specified using files, then augmented or overridden by the task.

The last group produced by a webject in this task becomes the credentials map. If the TASK parameter is not specified, a credentials mapping task is not executed. In this case the credentials map is derived only from the FILES parameter. The TASK parameter must be specified if the FILES parameter is omitted.

USERNAME

Identifies the name of the user for which a credentials map is being created. If no username is supplied, it is assumed that the credentials mapping task takes appropriate action to reject anonymous access (for example, by throwing an exception), or provides a default credentials map for anonymous access.

The default for this parameter is for the webject to obtain a username from the attribute named **auth-user** of the SERVER context group. This parameter is optional.

Query-Schema

DESCRIPTION

Retrieves schema information about object types either from the LDAP directory or another repository.

SYNTAX

```
<ie:webject name="Query-Schema" type="MGT">
  <ie:param name="TYPE" data=" com.myCompany.Address"/>
  <ie:param name="SCHEMA_GENERATOR" data="<generatorClass>"/>
  <ie:param name="GROUP_OUT" data="Attributes"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
TYPE		GROUP_OUT
		SCHEMA_GENERATOR

TYPE

Defines the type for which schema should be generated.

GROUP_OUT

Defines the name of the output group to be generated. There is no default value. If not supplied no GROUP_OUT is generated and an instance of the generated **SchemaObject** bean is stored on the collection as the SOAP response.

SHEMA_GENERATOR

Defines the fully qualified name of a Java class that implements the `com.infoengine.schema.SchemaGenerator` interface. The implementing class is required to know how to navigate a type hierarchy to generate the requested **SchemaObject**. This parameter defaults to an implementation that knows how to generate **SchemaObject** instances based on information found in the LDAP directory.

Additional parameters may be specified/required based on the value of SCHEMA_GENERATOR. For example the LDAP SCHEMA_GENERATOR may require (or allow) parameters like REPOSITORY or REPOSITORY_ TYPE to tell it where in the directory to look for schema information.

Query-TypeHierarchy

Description

Allows retrieval of information about existing type hierarchies. The Info*Engine group created contains information for each type found within a type hierarchy and a list of its ancestor or descendant types (depending on the direction of the traversal). The output group can be formatted as either a flat list of types, or as a nested group that reflects the structure of the type hierarchy.

SYNTAX

```
<ie:webject name="Query-TypeHierarchy" type="MGT">
  <ie:param name="RESOLVER" data="typeGraph"/>
  <ie:param name="TYPE" data="typeName"/>
  <ie:param name="DIRECTION"
            data="ANCESTORS | DESCENDANTS | BOTH"/>
  <ie:param name="MODE" data="NESTED | FLAT"/>
  <ie:param name="GROUP_OUT" data="groupName"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
TYPE		DIRECTION
		GROUP_OUT
		MODE
		RESOLVER

DIRECTION

Specifies the direction of traversal. Acceptable values are ANCESTORS, DESCENDANTS, and BOTH. The default value is BOTH. This parameter is optional.

GROUP_OUT

Specifies the name of the Info*Engine group in which to store the type hierarchy information.

The contents and format of the group depend on the values of TYPE, DIRECTION, and MODE.

- If MODE is specified with a value of FLAT, the output group contains one element per type; each element contains two attributes **type** and **ancestors**.
- If MODE is specified with a value of NESTED, the output group contains a single top-level element containing two attributes: **type**, and either **ancestors** or **descendants**.

Whether **ancestors** or **descendants** is used depends on the value specified for DIRECTION. If DIRECTION is specified with a value of ANCESTORS, the **ancestors** attribute is included, otherwise **descendants** is used. The value of the **ancestors** or **descendants** attribute is an Info*Engine element representing the next type in the hierarchy, and in turn its ancestors or descendants.

If a type hierarchy supports additional information per type in the hierarchy, additional attributes may be included in each element of the output group. For example, the Windchill implementation tells whether a particular type in the hierarchy is modeled or soft by including an attribute named **definition** whose value can be one of the following: `interface`, `class`, or `soft`.

The default value for this parameter is `hierarchy`. This parameter is optional.

MODE

Specifies how the output group is formatted. Acceptable values are: FLAT or NESTED. The default value is FLAT.

RESOLVER

Specifies the name of the type hierarchy implementation in which TYPE can be found. Currently there are two acceptable values for this parameter:

- `com.infoengine.schema.type.LDAPTypeHierarchyGraph` — Allows traversal of type hierarchies stored in LDAP.
- `com.ptc.core.adapter.server.impl.WindchillTypeHierarchyGraph` — Allows traversal of Windchill **type** hierarchies (requires Windchill be installed).

The default value is `com.infoengine.schema.type.LDAPTypeHierarchyGraph`. This parameter is optional.

TYPE

Specifies the name of the type at which to begin traversing the type hierarchy. This is a required parameter.

Throw-Exception

DESCRIPTION

Causes an exception to be thrown. If the webject is called from within the body of a unit within a task, control is transferred to a `failure` block, if the block exists. If the webject is called from outside of a unit, the task exits.

Note

Info*Engine arranges for the names and messages of exceptions thrown during task execution to be registered automatically in the SERVER context group attributes named **exception-class** and **exception-message**. This makes it possible for failure blocks within units to process exceptions and then re-throw them by calling the Throw-Exception webject without passing on any parameters. The SERVER context attribute named **exception-object** unwraps nested exceptions and creates the registers the raw, unwrapped exception object. The Throw-Exception webject looks for this SERVER context attribute and, if present, throw the associated exception object directly.

SYNTAX

```
<ie:webject name="Throw-Exception" type="MGT">
  <ie:param name="BUNDLE" data="bundle_name"/>
  <ie:param name="CLASS" data="exception_class_name"/>
  <ie:param name="GROUP_IN" data="input_group"/>
  <ie:param name="KEY" data="resource_key"/>
  <ie:param name="MESSAGE" data="text_message"/>
  <ie:param name="PARAM" data="resource_param"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	BUNDLE	CLASS
	GROUP_IN	
	KEY	
	MESSAGE	
	PARAM	

BUNDLE

Identifies the class name of a Java resource bundle containing localized message templates to be associated with the exception.

To use the KEY and PARAM parameters, you must also specify either BUNDLE or GROUP_IN.

CLASS

Specifies the Java class name of the exception to be thrown. If CLASS is omitted, the webject obtains the class name from the SERVER context group attribute named **exception-class**. If the attribute does not exist, the class name defaults to `com.infoengine.util.IEException`. This parameter is optional.

GROUP_IN

Identifies a group containing localized resource definitions, such as the group produced by the Get-Resource webject. If both BUNDLE and GROUP_IN are specified, BUNDLE takes precedence, and GROUP_IN is ignored.

To use the KEY and PARAM parameters, you must also specify either BUNDLE or GROUP_IN.

KEY

Selects a localized message template from the bundle or group. The KEY value may be specified as a number or a Java variable reference name.

This parameter is used only when you also specify either the BUNDLE or GROUP_IN parameter.

MESSAGE

The MESSAGE parameter specifies the textual message to be associated with the exception. Use this parameter only when it is not necessary to generate localizable messages in exceptions. If this parameter is omitted and the BUNDLE and GROUP_IN parameters are also omitted, the webject obtains message text from the SERVER context group attribute named **exception-message**. If BUNDLE or GROUP_IN are specified in addition to MESSAGE, the MESSAGE parameter takes precedence while BUNDLE and GROUP_IN are ignored.

PARAM

Supplies one or more values that are substituted into the localized message template to produce the final message text. If PARAM is not specified, it is assumed that the message template identified by the BUNDLE or GROUP_IN and the KEY parameters does not contain any substitution keywords, thus no substitution is performed.

This parameter is used only when you also specify either the BUNDLE or GROUP_IN parameter.

EXAMPLE

The following Throw-Exception example JSP page throws an exception using the localized message identified by KEY number 19 in the specified resource bundle:

```
<%@page language="java"
  session="false"
  errorPage="../IEError.jsp"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
```

```
                                               prefix="ie" %>

html>
<head><title>Throw Exception</title></head>
<body>

ie:webject name="Throw-Exception" type="MGT">
  <ie:param name="BUNDLE"
                     data="com.infoengine.util.IEResource"/>
  <ie:param name="KEY" data="19"/>
</ie:webject>

/body>
</html>
```

Write-Log

DESCRIPTION

Writes a message to the log associated with the context in which it is executed. For example, if the webject is called from a task executed within the Info*Engine task processor, the message is written to the task processor log. If the webject is called from a JSP page, it is written to the JSP processor log.

Note

When writing Info*Engine tasks, it is preferable to make use of the Logging tag library to issue log statements to log4j.

SYNTAX

```
<ie:webject name="Write-Log" type="MGT">
  <ie:param name="GROUP_IN" data="input_group"/>
  <ie:param name="LOG" data="[DEBUG | INFO | ERROR | AUDIT |
                                       TRANSACTION | STAT]"/>
  <ie:param name="MESSAGE" data="message_text"/>
  <ie:param name="TAG" data="author_supplied_tag"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		GROUP_IN
		LOG

Required	Select	Optional
		MESSAGE
		TAG

GROUP_IN

Specifies the name of the input group to log. The input group is rendered as XML and written to the log file specified by the LOG parameter. Multiple values can be specified. Any input groups are logged after the values supplied on the MESSAGE parameter.

LOG

Specifies the log to which the message should be written. Valid values for this parameter are DEBUG, INFO, ERROR, AUDIT, TRANSACTION, and STAT. The default for this parameter is INFO. This parameter is optional.

MESSAGE

Provides the main text of the message to be written to the log.

The format of each log message written is:

```
date-time#:#host#:#service#:#tag#:#user#:#messagevalue
```

where:

date-time—The date and time at which the message is written.

host—The Internet domain name of the host from which the message is written.

service—The name of the Info*Engine service that wrote the message.

tag—The value specified by the TAG parameter.

user—The authenticated username of the user for which the service was working when the message is written.

messagevalue—The value specified by the MESSAGE parameter.

The default is for a log message to be written that has an empty message value. This parameter is optional.

TAG

Specifies a tag that is written as part of the log entry for the purpose of facilitating the searching and sorting of logs. The TAG parameter allows a task author to provide a simple identifier that can be used to distinguish one kind of log message from another. For example, you can use special tags for troubleshooting purposes, or to classify messages in site-defined ways.

The default for this parameter is to use the name of the task from which the webject is being called. This parameter is optional.

EXAMPLE

The following Write-Log example JSP page creates a group, writes messages to two different logs, and displays the output:

```
<%@page language="java"
  session="false"
  errorPage="../IEError.jsp"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>

<html>
<head>
<title>Write-Log Webject</title>
</head>
<body>

<!-- create a group -->
<ie:task uri="com/company/CreateGroup.xml"/>

<!-- Write a log message to the default log, "info" -->

<ie:webject name="Write-Log" type="MGT">
  <ie:param name="TAG" data="-----Write-Log-----"/>
  <ie:param name="MESSAGE" data="----Info log message----"/>
</ie:webject>

<!-- Write a log message to the "debug" log -->
<ie:webject name="Write-Log" type="MGT">
  <ie:param name="TAG" data="-----Write-Log-----"/>
  <ie:param name="MESSAGE"
                         data="----Debug log message----"/>
  <ie:param name="LOG" data="DEBUG"/>
</ie:webject>

<!-- generate the output -->
<ie:webject name="Display-Table" type="DSP"/>

</body>
</html>
```

Message Webjects

The following webjects can be used in conjunction with a third-party MOM for generic messaging and task queuing functions:

- Browse-Queue on page 441
- Call-RemoteProcedure on page 444

- Call-SOAPService on page 447
- Create-Object on page 454
- Delete-Object on page 460
- Delete-Results on page 463
- Query-Object on page 465
- Query-Results on page 469
- Queue-Task on page 472
- Send-Mail on page 476
- Subscribe-Queue on page 481
- Unsubscribe-Queue on page 484

All message webjects use the `MSG` **type** attribute value in the **webject** tag.

Browse-Queue

Selects a message or set of messages in a queue, allowing the properties on messages to be viewed while not actually returning message content.

Note

This webject can be used only after your environment has been set up for messaging or for queuing tasks.

SYNTAX

```
<ie:webject name="Browse-Queue" type="MSG">
  <ie:param name="ATTRIBUTE" data="attribute_name"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="GROUP_OUT" data="group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WHERE" data="message_selector"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
QUEUE		ATTRIBUTE
		DBUSER
		SERVICE
		GROUP_OUT

Required	Select	Optional
		PASSWD
		WHERE

ATTRIBUTE

Specifies which JMS properties should be added to the output group as attributes. In addition to properties manually set on a message at the time of queuing the following JMS-specific properties can be returned as well:

- `JMSDestination`
- `JMSDeliveryMode`
- `JMSMessageID`
- `JMSTimestamp`
- `JMSCorrelationID`
- `JMSReplyTo`
- `JMSRedelivered`
- `JMSExpiration`
- `JMSPriority`
- `JMSType`

The default for this parameter is “*”, in which case all properties set as attributes are returned. This parameter is optional.

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

GROUP_OUT

The name of the group to create that contains the queue identifiers of the objects in the message queue. The default for this parameter is `browse-results`. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

The LDAP relative distinguished name of the managed queue to be browsed. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. This parameter is required.

WHERE

The JMS `where` clause used to select the messages from the queue. If no `where` clause is specified, all messages in the queue are returned. This parameter is optional.

EXAMPLE

The following example browses a specified queue for all attributes, then returns the default "browse-results" group using the Return-Groups webject:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                    prefix="ie"%>


<ie:webject name="Browse-Queue" type="MSG">
  <ie:param name="QUEUE" data="$(@FORM[]QUEUE[0])"/>
</ie:webject>

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="*"/>
</ie:webject>
```

To actually run this example, you must provide a form where the QUEUE variable is identified.

Call-RemoteProcedure

Makes a remote procedure call (RPC) to a third-party Simple Object Access Protocol (SOAP) service. Results from an RPC invocation are returned to the Info*Engine task or JSP author in the form of a group. The group mimics the XML structure of the SOAP response. In simple cases, where an RPC invocation results in a single value or array of simple values, a basic Info*Engine group containing one attribute per element is returned. It is possible for an RPC service to return a complex data structure in response. In this case a group, possibly containing multiple elements with multiple attributes, is returned. If the return XML is deeply nested, then the resulting Info*Engine group is also deeply nested, containing attributes whose values are elements. Dealing with such responses might require the use of scriptlet code within the handling task or JSP.

SYNTAX

```
<ie:webject name="Call-RemoteProcedure" type="MSG">
  <ie:param name="WSDL" data="url"/>
  <ie:param name="OPERATION" data="operationName"/>
  <ie:param name="GROUP_OUT" data="groupName"/>
  <ie:param name="GROUP_IN" data="groupName"/>
  <ie:param name="SERVICE" data="soapServiceName"/>
  <ie:param name="PORT" data="soapPortName"/>
  <ie:param name="PARAM" data="name=data"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="PASSWORD" data="password"/>
  <ie:param name="AUTHORIZATION" data="Base64"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
OPERATION		AUTHORIZATION
WSDL		DBUSER
		GROUP_IN
		GROUP_OUT
		PARAM
		PASSWD

Required	Select	Optional
		PORT
		SERVICE

AUTHORIZATION

A base64-encoded username and password passed from a web browser to a web server through the `Authorization` HTTP header. This parameter is used if the SOAP endpoint is `http` or `https`. If supplied, the credentials are used in establishing a connection to the SOAP endpoint. If DBUSER and PASSWORD are specified, they are used instead of the AUTHORIZATION value. This parameter is optional.

DBUSER

The user name entered when making a connection to the SOAP endpoint, if the endpoint's supporting protocol is `http` or `https`. If DBUSER is specified then PASSWORD must also be specified or it is ignored. This is an optional parameter.

GROUP_IN

An Info*Engine group that represents the SOAP request to be sent. All attributes found in the input group are added as XML elements. Attribute values are added as values of those XML elements. XML elements can be nested by setting attribute values to elements. This parameter can be used to generate arbitrarily complex SOAP requests. All elements generated from the input group are added to the standard RPC top-level element whose name is the operation name. This is an optional parameter. If specified, no validation of input is performed by the webject.

GROUP_OUT

The name of the Info*Engine group that the RPC invocation results should be stored in. Data found in the output group is strongly typed, based on type information found in the SOAP response. This is an optional parameter. The default value is `rpcOutput`.

OPERATION

The name of the operation to be executed. This is a required parameter. The operation name must be exposed by the WSDL document referenced by the WSDL parameter.

PARAM

A parameter name and value that should be sent with the RPC. The value must be in the form of `name=data`. Call-RemoteProcedure verifies the supplied parameters against the supplied WSDL to make sure that all required parameters are present, that the specified parameters are known to the service, that maximum and minimum occurrences of parameters are adhered to, and that parameter values are typed appropriately. It is possible that an operation requires no input parameters, and so this parameter is optional. If a web service requires specific parameters for the operation in question, then this parameter is required. If not specified, an exception is issued.

PASSWORD

Password to use when making a connection to the SOAP endpoint if the endpoint's supporting protocol is `http` or `https`. If PASSWORD is specified then DBUSER must also be specified or it is ignored. This is an optional parameter.

PORT

The name of the SOAP port that should be used. A SOAP service might be tied to multiple ports in a WSDL document. If the selected SERVICE is tied to more than one port then this parameter is required. If the selected SERVICE is only tied to a single port then that port is used.

SERVICE

The name of the SOAP service that should be used. A WSDL document might expose multiple SOAP services. If the WSDL document referenced by the WSDL parameter exposes multiple services, then this parameter is required. If the WSDL document referenced by the WSDL parameter exposes only a single service, then that service is used.

WSDL

References a Web Services Definition Language (WSDL) document. The value must be a URL whose contents can be read by Call-RemoteProcedure. A WSDL document describes a web service; what service names it supports, what operations each service supports, what parameters each operation requires or supports, how the SOAP request should be generated (parameter order, name spaces to use). This is a required parameter. If a specific SOAP service does not have a WSDL document that describes it, then one must be made. If the URL specified is `http` or `https` and requires authorization, it can be specified as follows:

`http://`*`<user>`*`:`*`<password>`*`@host/path/to/wsdl/.`

EXAMPLE

```
<%@page language="java"
%><%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%><html>
<head><title>Weather by Zip</title></head>
<body>

<%
String zip = request.getParameter ( "zipCode" );
if ( zip != null ) {
%><ie:webject name="Call-RemoteProcedure" type="MSG">
<ie:param name="WSDL"
data="http://www.vbws.com/services/weatherretriever.asmx?WSDL"/>
<ie:param name="OPERATION" data="GetWeather"/>
    <ie:param name="PORT" data="WeatherRetrieverSoap"/>
    <ie:param name="PARAM" data="zipCode=$(@FORM[]zipCode[])"/>
```

```
  </ie:webject><%
} %>

<form method="GET">
Zip:&nbsp;<input type=text size=10 name=zipCode value="<%=
(zip==null?"":zip)%>">&nbsp;
<input type=submit value="Get Weather" name=temp>
</form>
<%
if ( zip != null ) {
%><br><br>
<table>
<tr><td colspan=2 align=middle><img src="<ie:getValue name=
"IconUrl"/>"></td></tr>
<tr><td align=right><strong>Last Updated:<strong><td><td><ie:getValue
name="LastUpdated"/></td><tr>
<tr><td align=right><strong>Conditions:
</strong></td><td><ie:getValue name="Conditions"/></td></tr>
<tr><td align=right><strong>CurrentTemp:
<strong><td><td><ie:getValue name="CurrentTemp"/></td></tr>
<tr><td align=right><strong>Humidity:
</strong><td><td><ie:getValue name="Humidity"/></td></tr>
<tr><td align=right><strong>Barometer:</strong><td><td><ie:getValue
name="Barometer"/></td></tr>
<tr><td align=right><strong>Barometer Direction:
<strong><td><td><ie:getValue
name="BarometerDirection"/></td></tr>
</table><%}%></body></html>
```

Call-SOAPService

Makes a request to a third-party Simple Object Access Protocol (SOAP) service. This webject uses an Info*Engine group and an XSL template to format a SOAP request. The webject output can be directly generated as an Info*Engine group that mimics the SOAP response body, or an XSL template can be specified to translate the SOAP response into an Info*Engine group.

If a SOAP fault occurs, then an instance of `com.infoengine.exception.fatal.IERemoteException` is generated.

SYNTAX

```
<ie:webject name="Call-SOAPService" type="MSG">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="ENDPOINT" data="url"/>
  <ie:param name="GROUP_IN" data="groupName"/>
  <ie:param name="GROUP_OUT" data="groupName"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="SOAP_ACTION" data="operationName"/>
  <ie:param name="XSL_URL" data="url"/>
  <ie:param name="XSL_RESPONSE_URL" data="url"/>
  <ie:param name="XSL_DBUSER" data="username"/>
  <ie:param name="XSL_PASSWD" data="passwd"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
ENDPOINT		DBUSER
GROUP_IN		EXCEPTION_ON_FAULT
XSL_URL		GROUP_OUT
		PASSWD
		SOAP_ACTION
		XSL_DBUSER
		XSL_PASSWD
		XSL_RESPONSE_URL

DBUSER

The username to use to authenticate to ENDPOINT. This parameter is used in conjunction with PASSWD if the SOAP endpoint specified on the ENDPOINT parameter requires authentication. This parameter is optional.

ENDPOINT

The URL of the SOAP endpoint to which the SOAP request should be sent. This parameter is required.

EXCEPTION_ON_FAULT

Specifies whether an exception is thrown if the service responds with a SOAP fault. The default value is `true`. If the parameter is set to `false`, then no exception is thrown if a fault occurs. If XSL is being used to transform the results, then an XSL template can be used to translate the fault into Info*Engine data (including the contents of the detail element), and other actions taken. If XSL is not being used, then the webject translates the fault into an Info*Engine group. This parameter is optional.

GROUP_IN

The Info*Engine group containing data to combine with the XSL template specified on XSL_URL. This parameter is required.

GROUP_OUT

The name of the Info*Engine group representing the SOAP response. If XSL_RESPONSE_URL is specified, then it is used to translate the SOAP response into an Info*Engine group named with the GROUP_OUT value. If XSL_

RESPONSE_URL is not specified, then this group contains a single element and one attribute per XML element in the response. The value of an attribute can be an object, representing element data; or an Info*Engine element, representing additional nested XML elements. The name of an attribute matches the corresponding XML element name. The default value for this parameter is `SOAPResponse`. This parameter is optional.

PASSWD
The password corresponding to DBUSER. This parameter is optional.

SOAP_ACTION
The value (if any) of the `SOAPAction` header that should be sent. This parameter is optional.

XSL_DBUSER
The username to used to authenticate to XSL_URL and XSL_RESPONSE_URL. This parameter is used in conjunction with XSL_PASSWD if the XSL templates to be used by the webject reside on a remote HTTP server that requires authentication. This parameter is optional.

XSL_PASSWD
The password corresponding to XSL_DBUSER. This parameter is optional.

XSL_RESPONSE_URL
The location of an XSL template used to process the SOAP response. The XSL template is combined with the SOAP response to create the GROUP_OUT. This parameter is optional.

Fully qualified URLs are dereferenced using **Auth-Map** context group data. The **Auth-Map** is searched for a username and password based on the domain name found in the fully qualified URL. For example, if the fully qualified URL is:

```
http://machine.com/infoengine/servlet/IE/tasks/createGroupData.xml
```

the **Auth-Map** context group is searched for a username and password that has an INSTANCE name of `http://machine.com`. If a username and password are found, BASIC authentication information is used when accessing the URL. If no username and password are found, no authentication information is sent to the remote web server.

If the data value contains the `://` string, it is assumed to be a fully qualified internet URL. If the data value does not contain the string, it is assumed to be a local file relative to the current task root directory.

XSL_URL
The location of the XSL template used to generate the SOAP request. The XSL template is combined with the GROUP_IN to generate the SOAP request.

When authoring the XSL documents used to translate the SOAP request and response, if the example request and response pair for the XML documents are available they can be used as templates for the XSL. If there are no such example request and response documents available, then the XSL author needs to pay close attention to the schema provided by the web service's WSDL document. Complying with the WSDL document is necessary to provide valid and acceptable XML to the web service. Depending on how the service is implemented, it might or might not be flexible in terms of what it accepts. However, the schema provided in the WSDL should be taken literally and be used to craft XML that validates against that schema by complying with the order, case, optionality, cardinality, and data type for all XML elements and attributes.

Fully qualified URLs are de-referenced using **Auth-Map** context group data. The **Auth-Map** is searched for a username and password based on the domain name found in the fully qualified URL. For example, if the fully qualified URL is:

```
http://machine.com/infoengine/servlet/IE/tasks/createGroupData.xml
```

the **Auth-Map** context group is searched for a username and password that has an INSTANCE name of `http://machine.com`. If a username and password are found, BASIC authentication information is used when accessing the URL. If no username and password are found, no authentication information is sent to the remote web server.

If the data value contains the `://` string, it is assumed to be a fully qualified internet URL. If the data value does not contain the string, it is assumed to be a local file relative to the current task root directory.

This parameter is required.

EXAMPLE: TRANSLATE-GROUP AND XSL

Note

Info*Engine SOAP comments are used in the example code for clarity. For more information on SOAP and SOAP comments, see SOAP Services on page 195.

This example uses the Translate-Group webject and XSL to simplify a SOAP response that was returned from the Call-SOAPService webject as a complex Info*Engine group.

hostlookup.xml

```
<%@page language="java" import="java.util.StringTokenizer,com.infoengine.
object.
```

```
            IeMultipartInputStream,java.io.ByteArrayInputStream"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                  prefix="ie"%>

<!--com.infoengine.soap.rpc.def
Resolves information about a host name.

@param string host The hostname
@return INFOENGINE_GROUP $(output)-->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="host=$(@FORM[]host[])"
                           default="www.google.com"/>
  <ie:param name="CLASS" data="parameters"/>
  <ie:param name="GROUP_OUT" data="parameters"/>
</ie:webject>

<ie:webject name="Call-SOAPService" type="MSG">
  <ie:param name="ENDPOINT"
     data="http://www.esynaps.com/webservices/YourHostInfo.asmx"/>
  <ie:param name="SOAP_ACTION"
            data="http://tempuri.org/GetHostInfoByName"/>
  <ie:param name="XSL_URL"
          data="/com/company/hostlookup/ hostlookupRequest.xsl"/>
  <ie:param name="GROUP_IN" data="parameters"/>
  <ie:param name="GROUP_OUT" data="response"/>
  <ie:param name="CLASS" data="response"/>
</ie:webject>

<ie:webject name="Translate-Group" type="GRP">
  <ie:param name="GROUP_IN" data="response"/>
  <ie:param name="XSL_URL"
         data="/com/company/hostlookup/
                                    hostlookupResponse.xsl"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"
                             default="output"/>
  <ie:param name="CLASS" data="lookupResponse"/>
</ie:webject>
```

hostlookupRequest.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:wc="http://www.ptc.com/infoengine/1.0">

<xsl:template match="/wc:COLLECTION/parameters/wc:INSTANCE">
```

```
<soap:Envelope xmlns:s0="http://tempuri.org/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
      <s0:GetHostInfoByName>
        <s0:Name><xsl:value-of select="host"/></s0:Name>
      </s0:GetHostInfoByName>
    </soap:Body>
</soap:Envelope>
</xsl:template>
</xsl:stylesheet>
```

hostlookupResponse.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"

  xmlns:wc="http://www.ptc.com/infoengine/1.0">
<xsl:template match="/wc:COLLECTION/response/wc:INSTANCE">
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
<lookupResponse TYPE="Object" STATUS="0">
  <wc:INSTANCE>
    <xsl:for-each select="Aliases/wc:INSTANCE/Alias">
    <Alias><xsl:value-of select="current()"/></Alias>
    </xsl:for-each>
    <Name><xsl:value-of select="Name"/></Name>
    <xsl:for-each select="IPList/wc:INSTANCE/IPAddress">
    <IPAddress><xsl:value-of select="current()"/></IPAddress>
    </xsl:for-each>
  </wc:INSTANCE>
</lookupResponse>
</wc:COLLECTION>
</xsl:template>
</xsl:stylesheet>
```

EXAMPLE: XSL_RESPONSE_URL

This example uses the XSL_RESPONSE_URL parameter to translate the SOAP response into a group identical to that produced by the previous example:

hostlookup2.xml

```
<%@page language="java"           import="java.util.StringTokenizer,com.
infoengine.object.
            IeMultipartInputStream,java.io.ByteArrayInputStream"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                    prefix="ie"%>
```

```
<!--com.infoengine.soap.rpc.def
Resolves information about a host name, goes straight from SOAP
response to group.

@param string host The hostname
@return INFOENGINE_GROUP $(output)
-->

<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT" data="host=$(@FORM[]host[])"
                           default="www.google.com"/>
  <ie:param name="CLASS" data="parameters"/>
  <ie:param name="GROUP_OUT" data="parameters"/>
</ie:webject>

<ie:webject name="Call-SOAPService" type="MSG">
  <ie:param name="ENDPOINT"
     data="http://www.esynaps.com/webservices/YourHostInfo.asmx"/>
  <ie:param name="SOAP_ACTION"
            data="http://tempuri.org/GetHostInfoByName"/>
  <ie:param name="XSL_URL"
          data="/soap/document/hostlookup/hostlookupRequest.xsl"/>
  <ie:param name="XSL_RESPONSE_URL"
     data="/soap/document/hostlookup/hostlookupResponseSOAP.xsl"/>
  <ie:param name="GROUP_IN" data="parameters"/>
  <ie:param name="GROUP_OUT" data="output"/>
  <ie:param name="CLASS" data="output"/>
</ie:webject>
```

hostlookupResponseSOAP.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
  xmlns:wc="http://www.ptc.com/infoengine/1.0"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:dflt="http://tempuri.org/">

<!--Due to XSL/XPath limitation and not being able to deal with
the "default" namespace weneed to assign a bogus id (here dflt)
to the default namespace and use it in our XSL so that via XPath
we can select unqualified elements within the SOAP response.
-->

<xsl:template match="/soap:Envelope/soap:Body/
     dflt:GetHostInfoByNameResponse/dflt:GetHostInfoByNameResult">
<wc:COLLECTION xmlns:wc="http://www.ptc.com/infoengine/1.0">
```

```
      <output TYPE="Object" STATUS="0">
        <wc:INSTANCE>
          <Name><xsl:value-of select="dflt:Name"/></Name>
          <xsl:for-each select="dflt:Aliases/dflt:Alias">
          <Alias><xsl:value-of select="current()"/></Alias>
          </xsl:for-each>
          <xsl:for-each select="dflt:IPList/dflt:IPAddress">
          <IPAddress><xsl:value-of select="current()"/></IPAddress>
          </xsl:for-each>
        </wc:INSTANCE>
      </output>
    </wc:COLLECTION>
    </xsl:template>
    </xsl:stylesheet>
```

Create-Object

Creates a JMS message and places that message in a JMS message queue. The type and contents of the message are governed by its parameters.

Note

This webject can be used only after your environment has been set up for messaging.

SYNTAX

```
<ie:webject name="Create-Object" type="MSG">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="GROUP_IN" data ="group_name"/>
  <ie:param name="GROUP_OUT" data ="status_group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="PRIORITY" data ="numeric_value"/>
  <ie:param name="PROPERTY" data ="name_value_pair"/>
  <ie:param name="PUT_BOOLEAN" data ="[TRUE | FALSE]"/>
  <ie:param name="PUT_BYTE" data ="byte"/>
  <ie:param name="PUT_BYTES" data ="bytes"/>
  <ie:param name="PUT_CHAR" data ="char"/>
  <ie:param name="PUT_DOUBLE" data ="double"/>
  <ie:param name="PUT_FLOAT" data ="float"/>
  <ie:param name="PUT_INT" data ="int"/>
  <ie:param name="PUT_LONG" data ="long"/>
  <ie:param name="PUT_SHORT" data ="short"/>
  <ie:param name="PUT_STRING" data ="string"/>
  <ie:param name="QUEUE" data ="managed_queue_name"/>
  <ie:param name="TIME_TO_LIVE" data="minutes"/>
  <ie:param name="TYPE" data ="message_type"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
QUEUE	PUT_BOOLEAN	DBUSER
	PUT_BYTE	GROUP_IN
	PUT_BYTES	GROUP_OUT
	PUT_INT	PASSWD
	PUT_CHAR	PRIORITY
	PUT_DOUBLE	PROPERTY
	PUT_FLOAT	TIME_TO_LIVE
	PUT_LONG	SERVICE
	PUT_SHORT	
	PUT_STRING	
	TYPE	

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

GROUP_IN

The name of an Info*Engine group to serialize with the request. A value of "*" means serialize all groups. This parameter only makes sense if the TYPE parameter is Info*Engine. This parameter is optional.

GROUP_OUT

The name of the status group to create upon success. The default for this parameter is to created a group named “create-results”. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM. This parameter is optional.

PRIORITY

Specifies an integer priority to be set on the outgoing messages. The valid range for JMS Queue priorities is from 0 – 9, with 4 the typical default value. Priorities are ignored by Info*Engine, but can be used by third-party software receiving messages queued by Info*Engine. This parameter is optional.

PROPERTY

Specifies a name and value pair to set on the outgoing JMS message. These properties are ignored by Info*Engine, but can be used by third-party software. The format for the PROPERTY value is:

```
name=value
```

where:

- *name*—The property name.
- *value*—The corresponding value of the property.

This default behavior for this parameter is not to set any additional properties on the outgoing message. This parameter is optional.

PUT_BOOLEAN

Places a Java Boolean value into the JMS message. The value can be specified as TRUE or FALSE. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_BYTE

Places a single byte into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_BYTES

Places multiple bytes into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_CHAR

Places a single Java character into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_DOUBLE

Places a Java double value into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_FLOAT

Places a floating point number into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_INT

Places an integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_LONG

Places a long integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_SHORT

Places a short integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **MapMessage**, or **StreamMessage**.

PUT_STRING

Places an instance of `java.lang.String` into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **TextMessage**, **MapMessage**, or **StreamMessage**.

QUEUE

The LDAP relative distinguished name of the JMS Queue to which to submit the new message. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. This parameter is required.

TIME_TO_LIVE

Specifies how long, in minutes, the queued message should be valid. The default for this parameter is 0, in which case the message remains valid indefinitely. This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

TYPE

Indicates the type of JMS message to create. Message types are case-sensitive. Possible message types are:

- **InfoEngine**—Used for Info*Engine to Info*Engine communication. VDB groups serialized are governed by the GROUP_IN parameter.
- **BytesMessage**—For outgoing messages, this type supports several PUT_type parameters for building a message that might be specific to a third-party recipient. Incoming messages are processed in the same manner as **Unknown**. The contents of a **BytesMessage** is supplied as BLOB data on an input stream to the calling task.
- **TextMessage**—Supports only a PUT_STRING parameter. Multiple PUT_STRING parameter values are concatenated together to create the outgoing **TextMessage**.
- **MapMessage**—For outgoing messages, this type supports several PUT_type parameters with values in the form of `name=data`. The *name* portion of the value is the key used to place *data* in the **MapMessage**.
- **StreamMessage**—For outgoing messages, this type supports several PUT_type parameters for building a message that might be specific to a third-party recipient.
- **Unknown**—Turns BLOB data into a **BytesMessage**. For incoming messages, a new empty `IeRequest` is created and the contents of the message are tacked on as BLOB data.

The following table indicates which PUT_type parameters can be specified with which message types:

	BytesMessage	TextMessage	MapMessage	StreamMessage
PUT_BOOLEAN	X		X	X
PUT_BYTE	X		X	X
PUT_BYTES	X		X	X
PUT_CHAR	X		X	X
PUT_DOUBLE	X		X	X
PUT_FLOAT	X		X	X
PUT_INT	X		X	X
PUT_LONG	X		X	X
PUT_SHORT	X		X	X
PUT_STRING	X	X	X	X

Note

Support for the type STRING in **BytesMessage** actually causes the value of the string to be written as a byte array; it is not actually written as a string.

The default value for this parameter is `InfoEngine`, meaning that an Info*Engine request is serialized and placed in the queue. “Serialized” in this case can mean actual Java serialization or serialization of an Info*Engine request to XML (this is the case when BLOB data needs to be queued as well).

EXAMPLES

The following example creates a JMS message and submits it to the specified queue:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<ie:webject name="Create-Object" type="MSG">
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="QUEUE" data="$(@FORM[]queue[])"/>
  <ie:param name="TIME_TO_LIVE" data="$(@FORM[]time[])"/>
  <ie:param name="PRIORITY" data="$(@FORM[]priority[])"/>
  <ie:param name="TYPE" data="TextMessage"/>
  <ie:param name="PUT_STRING" data="$(@FORM[]string[])"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

To actually run this example you must provide a form where the **dbuser**, **passwd**, **queue**, **time**, **priority**, and **string** variables are identified.

The following example shows how the Create-Object webject could be used to queue a **TextMessage** containing XML data to be processed by a third-party application:

```
<%@page language="java"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<!-- get an "employee object from JDBC/JNDI or where ever,
     for the purposes of example just using Create-Group -->
<ie:webject name="Create-Group" type="GRP">
  <ie:param name="ELEMENT"
            data="name=joe:email=joe@somewhere.nice"/>
  <ie:param name="ELEMENT" data="name=bob:email=bob@nowhere.net"/>
  <ie:param name="GROUP_OUT" data="employees"/>
</ie:webject>

<!-- xml-ize the parts -->
<ie:webject name="Format-Group" type="GRP">
  <ie:param name="GROUP_IN" data="employees" />
  <ie:param name="GROUP_OUT" data="formatted"/>
  <ie:param name="ATTRIBUTE" data="name"/>
  <ie:param name="ATTRIBUTE" data="email"/>
  <ie:param name="FORMAT"
  data="<employee><name>{0}</name><email>{1}</email></employee>"/>
</ie:webject>

<!-- turn them into an XML document and put in in a TextMessage on
my queue -->
<ie:webject name="Create-Object" type="MSG">
  <ie:param name="PUT_STRING" data="<?xml version='1.0'>&#10;"/>
  <ie:param name="PUT_STRING" data="<employees>&#10;"/>
  <ie:param name="PUT_STRING"
            data="$(formatted[*]nameemail[0])&#10;"
                                valueSeparator=";" delim=";" />
  <ie:param name="PUT_STRING" data="</employees>&#10;"/>
  <ie:param name="TYPE" data="TextMessage"/>
  <ie:param name="QUEUE" data="cn=The.Q"/>
</ie:webject>
```

Delete-Object

Removes a message from a JMS queue.

Note

This webject can be used only after your environment has been set up for messaging.

SYNTAX

```
<ie:webject name="Delete-Object" type="MSG">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="GROUP_OUT" data="group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WHERE" data="message_selector"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
QUEUE		DBUSER
		SERVICE
		GROUP_OUT
		PASSWD
		WHERE

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

GROUP_OUT

The name of the status group to create and return upon success. The default for this parameter is `delete-results`. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

Identifies the LDAP relative distinguished name of the JMS queue to remove the message from. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. This parameter is required.

WHERE

Specifies a properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select a subset of messages from the message queue for deletion. The first message in the subset on the queue is deleted. The default behavior for this parameter is for the first message on the queue to be deleted. This parameter is optional.

EXAMPLE

The following example deletes a JMS message from the specified queue:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<ie:webject name="Delete-Object" type="MSG">
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="QUEUE" data="$(@FORM[]queue[])"/>
  <ie:param name="WHERE" data="$(@FORM[]where[])"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the **dbuser**, **passwd**, **queue**, and **where** variables are identified.

Delete-Results

Retrieves and deletes a message from an Info*Engine task queue. It is assumed that the message contains a known Info*Engine object, normally a group to return to a user.

Note

This webject can be used only after your environment has been set up for queuing tasks.

SYNTAX

```
<ie:webject name="Delete-Results" type="MSG">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="<managed_queue_name>"/>
  <ie:param name="WHERE" data="<message_selector>"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		DBUSER
		SERVICE
		PASSWD
		QUEUE
		WHERE

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

The LDAP distinguished name of a managed queue. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultResponseQueue` property.

This parameter is optional.

WHERE

Specifies a properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select a subset of messages from the message queue for deletion. The first message in the subset on the queue is deleted.

The value of the CORRELATION_ID parameter from the Queue-Task webject can be used as the WHERE value to select the corresponding result.

The default value of this parameter is derived from the user identifier found in the SERVER context group and results in a subset that contains this user's messages. This parameter is optional.

EXAMPLE

The following example retrieves a message from the specified queue, then deletes the message:

```
<%@page language="java" session="false" %>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>
<ie:webject name="delete-results" type="MSG">
  <ie:param name="QUEUE" data="$(FORM[]QUEUE[0])"/>
  <ie:param name="CORRELATION_ID"
                              data="$(FORM[]CORRID[0])"/>
  <ie:param name="WHERE" data="$(@FORM[]where[0])"/>
  <ie:param nam

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="*"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the QUEUE, CORRID, and **where** variables are identified.

Query-Object

Retrieves a message from a JMS message queue. If there is no message to return, an instance of `com.infoengine.exception.nonfatal.IERequestTimeOutException` is thrown.

Note

This webject can be used only after your environment has been set up for messaging.

SYNTAX

```
<ie:webject name="Query-Object" type="MSG">
  <ie:param name="BLOB_COUNT" data="number"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="DELETE" data="[TRUE | FALSE]"/>
  <ie:param name="GET_BOOLEAN" data ="[TRUE | FALSE]"/>
  <ie:param name="GET_BYTE" data ="byte"/>
  <ie:param name="GET_BYTES" data ="bytes"/>
  <ie:param name="GET_CHAR" data ="char"/>
  <ie:param name="GET_DOUBLE" data ="double"/>
  <ie:param name="GET_FLOAT" data ="float"/>
  <ie:param name="GET_INT" data ="int"/>
  <ie:param name="GET_LONG" data ="long"/>
  <ie:param name="GET_SHORT" data ="short"/>
  <ie:param name="GET_STRING" data ="string"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WHERE" data="message_selector"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
QUEUE		BLOB_COUNT
		DBUSER
		SERVICE
		DELETE
		GET_BOOLEAN
		GET_BYTE
		GET_BYTES
		GET_CHAR
		GET_DOUBLE
		GET_FLOAT
		GET_INT
		GET_LONG
		GET_SHORT
		GET_STRING
		PASSWD
		WHERE

BLOB_COUNT

The number of BLOBs to return on the caller's output stream. Any BLOBs not returned can be used as input to other tasks or webjects. The default for this parameter is 0. This parameter is optional.

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted. This parameter is optional.

SERVICE
The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

DELETE
Specifies whether or not to remove the message from the queue once it is retrieved from the queue. The default for this parameter is TRUE, which deletes the object once it is retrieved from the queue. Use FALSE to disable. This parameter is optional.

GET_BOOLEAN
Retrieves a Java Boolean value from the JMS message. The value can be specified as TRUE or FALSE. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_BYTE
Retrieves a single byte from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_BYTES
Retrieves multiple bytes from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_CHAR
Retrieves a single Java character from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_DOUBLE
Retrieves a Java double value from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_FLOAT
Retrieves a floating point number from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_INT
Retrieves an integer from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_LONG

Retrieves a long integer from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_SHORT

Retrieves a short integer from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

GET_STRING

Retrieves an instance of `java.lang.String` from the JMS message. This parameter can only be used if the incoming message type is **StreamMessage**. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

The LDAP distinguished name of the JMS Queue from which to retrieve the message. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. This parameter is required.

WHERE

Specifies a properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select a subset of messages from the message queue for retrieval. The first message in the subset on the queue is returned. The default for this parameter is to select all messages in the queue. This parameter is optional.

EXAMPLE

The following example retrieves a message from the specified queue:

```
<%@page language="java" session="false"  %>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                                prefix="ie"%>

<ie:webject name="Query-Object" type="MSG">
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="DELETE" data="$(@FORM[]delete[])"/>
  <ie:param name="QUEUE" data="$(@FORM[]queue[])"/>
  <ie:param name="WHERE" data="$(@FORM[]where[])"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the **dbuser**, **passwd**, **delete**, and **where** variables are identified.

Query-Results

Retrieves a message from an Info*Engine task queue, returns the results, and by default, deletes the message from the queue. It is assumed that the message contains a known Info*Engine object, normally a group to return to a user.

Note

This webject can be used only after your environment has been set up for queuing tasks.

SYNTAX

```
<ie:webject name="Query-Results" type="MSG">
  <ie:param name="CORRELATION_ID" data="message_selector"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="DELETE" data="[TRUE | FALSE]"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WAIT_TIME" data="wait_time"/>
  <ie:param name="WHERE" data="message_selector"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		CORRELATION_ID
		DBUSER
		SERVICE
		DELETE
		PASSWD
		QUEUE
		WAIT_TIME
		WHERE

CORRELATION_ID

The JMS header correlation identifier. This value is used to select results from a response queue and corresponds to the CORRELATION_ID parameter used when queueing a task for execution with the Queue-Task webject. This parameter is optional.

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

DELETE

Indicates whether or not the message is to be deleted from the queue when successfully returned to the user. If the value specified is TRUE, then the message is deleted. If the value specified is FALSE, then the message is not deleted. The default for this parameter is TRUE. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

An LDAP distinguished name of a managed queue. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultResultsQueue` property.

This parameter is optional.

WAIT_TIME

The time in seconds to wait for the result to arrive in the specified queue. The default for this parameter is 0. This parameter is optional.

WHERE

Specifies a properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select a subset of messages from the message queue for retrieval. The first message in the subset on the queue is returned.

The value of the CORRELATION_ID parameter from the Queue-Task webject can be used as the WHERE value to select the corresponding result.

The default value of this parameter is derived from the user identifier found in the SERVER context group and results in a subset that contains this user's messages. This parameter is optional.

EXAMPLE

The following example retrieves a message from the specified queue and returns the results:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>
```

```
<ie:webject name="Query-Results" type="MSG">
  <ie:param name="QUEUE" data="$(@FORM[]queue[0])"/>
  <ie:param name="CORRELATION_ID"
                                data="$(@FORM[]corrid[0])"/>
  <ie:param name="WHERE" data="$(@FORM[]where[0])"/>
  <ie:param name="DELETE" data="$(@FORM[]delete[])"/>
  <ie:param name="WAIT_TIME" data="10"/>
</ie:webject>

<ie:webject name="Return-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="*"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the **queue**, **corrid**, **where**, and **delete** variables are identified.

Queue-Task

Allows a subscriber of an Info*Engine task queue to add a task to the queue for execution. The task is built from a task execution name and if necessary one or more existing Info*Engine groups.

Note

This webject can be used only after your environment has been set up for queuing tasks.

SYNTAX

```
<ie:webject name=Queue-Task type="MSG">
  <ie:param name="CORRELATION_ID" data="message_selector"/>
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="DESTINATION" data="queue_name"/>
  <ie:param name="GROUP_IN" data="group_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="PRIORITY" data="numeric_value"/>
  <ie:param name="QUEUE" data="queue_name"/>
  <ie:param name="RESULT_TIME_TO_LIVE" data="minutes"/>
  <ie:param name="TASK" data="task_uri"/>
  <ie:param name="TASK_TIME_TO_LIVE" data="minutes"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
TASK		CORRELATION_ID
		DBUSER
		SERVICE
		DESTINATION
		GROUP_IN
		PASSWD
		PRIORITY
		QUEUE
		RESULT_TIME_TO_LIVE
		TASK_TIME_TO_LIVE

CORRELATION_ID

Specifies a JMS message selector to set as the JMS header correlation identifier of the result. The value of this parameter can then be used as the value of the WHERE parameter of the Query-Results or Delete-Results webjects to select the corresponding result.

The default value of this parameter is derived from the `auth-user` HTTP header value found in the SERVER context group. If `auth-user` is not defined and no explicit correlation identifier is specified, no correlation identifier is associated with the task. In this case, the Info*Engine group returned from Query-Task returns the value of the correlation identifier that was associated with the task. This parameter is optional.

DESTINATION

The LDAP distinguished name of the queue where the response should be queued after the task is executed. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultResponseQueue` property. This parameter is optional.

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted. This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

GROUP_IN

The name of a local VDB Info*Engine group object to attach to the task to be queued. The default behavior is that no group is attached. Multiple groups can be specified on this parameter. This parameter is optional.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM. This parameter is optional.

PRIORITY

Specifies an integer priority to be set on the outgoing messages. The valid range for JMS Queue priorities is from 0 – 9, with 4 the typical default value. Priorities are ignored by Info*Engine, but can be used by third-party software receiving events sent by Info*Engine. The default for this parameter is not to set a priority which results in the message being queued with the MOM’s default priority. This parameter is optional.

QUEUE

The LDAP distinguished name of the queue where the task is to be placed. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultExecutionQueue` property. This parameter is optional.

RESULT_TIME_TO_LIVE

Indicates a numeric value specifying how long the results of a task execution should remain in the response queue before being discarded by the MOM. This parameter is specified as a numeric string in minutes. The default for this parameter is 0, in which case the MOM does not discard the response. This parameter is optional.

TASK

Specifies a URI that is the location of the XML task file for the listener on the EXECUTION_QUEUE to execute. The URI can be a relative or absolute URI:

- Relative URIs reference files that reside under the root file system directory that is defined for the local Info*Engine task processor.
- Absolute URIs reference files that reside in the local file system, reside on a remote HTTP server, or are referenced through an accessible LDAP directory.

For example URIs, see Specifying URIs and URLs on page 68.

This parameter is required.

Note

The URIs shown in this guide use the forward slash as the separator (/) in file paths even though the back slash (\) is the directory separator used on NT systems. Info*Engine correctly identifies all system URIs when you specify the forward slash. If you prefer to use the back slash for NT URIs, you must escape the back slash in the URI. This means that you enter two \ \ for each \ in the URI.

TASK_TIME_TO_LIVE

Indicates a numeric value specifying how long the message should remain in the execution queue before it can be discarded by the MOM. This parameter is specified as a numeric string in minutes. The default for this parameter is 0, in which case the MOM does not discard the message. This parameter is optional.

EXAMPLE

The following example adds the specified task to the specified queue:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>

<ie:webject name="queue-task" type="MSG">
  <ie:param name="TASK" data="$(@FORM[]TASK[0])"
             default="/com/company/CreateGroup.xml"/>
```

```
     <ie:param name="QUEUE" data="$(@FORM[]QUEUE[0])"/>
     <ie:param name="TASK_TIME_TO_LIVE"
                                data="$(@FORM[]TTTL[0])"/>
     <ie:param name="RESULT_TIME_TO_LIVE"
                                data="$(@FORM[]RTTL[0])"/>
     <ie:param name="DESTINATION" data="$(@FORM[]DEST[0])"/>
     <ie:param name="CORRELATION_ID"
                              data="$(@FORM[]CORRID[0])"/>
     <ie:param name="PRIORITY" data="$(@FORM[]PRIORITY[])"/>
     <ie:param name="GROUP_OUT" data="myQueueResults"/>
   </ie:webject>
```

To actually run this example you would need to provide a form where the TASK, QUEUE, TTTL, RTTL, DEST, CORRID, and PRIORITY variables are identified.

Send-Mail

Creates and sends an email message. Send-Mail supports designation of primary, copy, and blind copy recipients, as well as message subject and message bodies. It can create and send multi-part MIME messages containing a mixture of message body part types. It allows message body parts to be specified as simple text using webject parameters or as BLOBs using the task input stream.

SYNTAX

```
<ie:webject name="Send-Mail" type="MSG">
  <ie:param name="BCC" data="e-mail_address"/>
  <ie:param name="BLOB_COUNT" data="number"/>
  <ie:param name="CC" data="e-mail_address"/>
 <ie:param name="CONNECTION_TIMEOUT" data="milliseconds"/>
  <ie:param name="CONTENT" data="message_body"/>
  <ie:param name="CONTENT_TYPE" data="MIME_type"/>
  <ie:param name="CONTENT_URL_LANGUAGE" data="
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="FROM" data="e-mail_address"/>
  <ie:param name="GROUP_OUT" data="results"/>
  <ie:param name="HEADERS_CHARSET" data="
  <ie:param name="MAIL_SERVER" data="host_name"/>
  <ie:param name="MAX_DELIVERY_TIME" data="
  <ie:param name="MULTIPART_TYPE" data="
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="PASSWORD" data="password"/>
  <ie:param name="REPLY_TO" data="e-mail_address"/>
  <ie:param name="SUBJECT" data="message_subject"/>
 <ie:param name="TIMEOUT" data="milliseconds"/>
  <ie:param name="TO" data="e-mail_address"/>
  <ie:param name="USERNAME" data="user_name"/>
```

```
</webject>
```

PARAMETERS

Required	Select	Optional
TO	BLOB_COUNT	CC
		CONNECTION_TIMEOUT
	CONTENT	BCC
	CONTENT_LINE_BREAK	DBUSER
	CONTENT_TYPE	FROM
	CONTENT_URL	GROUP_OUT
		HEADER
		HEADERS_CHARSET
		MAIL_SERVER
		MAX_DELIVERY_TIME
		MULTIPART_TYPE
		PASSWD
		PASSWORD
		REPLY_TO
		SUBJECT
		TIMEOUT
		USERNAME

BCC

The email addresses of the blind copy recipients of the message. Each of these recipients receives a copy of the message, but none of these addresses are disclosed to any other recipients, including any other blind copy recipients. This parameter can be specified multiple times to specify multiple blind copy recipients. Each instance of the parameter specifies exactly one address. This parameter is optional.

BLOB_COUNT

The number of BLOBs to read from the input stream. If the CONTENT parameter is also specified, the BLOBs are appended to the message as attachments. If the CONTENT parameter is not specified, then the BLOBs become the message.

CC

The email addresses of non-primary recipients to which copies of the message are sent. This parameter can be specified multiple times to specify multiple copy recipients. Each instance of the parameter specifies exactly one address. This parameter is optional.

CONTENT

The text that appears in the message body. Each instance of the CONTENT parameter can correspond to an instance of the CONTENT_TYPE parameter, which specifies the MIME content type of the CONTENT parameter. If no CONTENT_TYPE is specified, the CONTENT parameter defaults to plain text unless the BLOB_COUNT parameter is specified. Multiple values can be specified for this parameter.

CONTENT_LINE_BREAK

This parameter is a multi-valued parameter that can specify a string (or strings) to be used as a line break. Wherever the value of this parameter exists in the content of the email it is replaced with a line break.

CONTENT_TYPE

The MIME content type of a corresponding CONTENT parameter. Each instance of the CONTENT_TYPE parameter corresponds to an instance of the CONTENT parameter and specifies the MIME content type of that parameter. If no CONTENT_TYPE parameters are specified, the MIME content types of the CONTENT parameters defaults to plain text. Similarly, if there are fewer CONTENT_TYPE parameters than CONTENT parameters, plain text values are implicitly added to the list of CONTENT_TYPE parameters.

CONTENT_URL

The CONTENT_URL parameter can be multi-valued. Each value specifies a URL that is opened and read. The content read from the URL becomes a body part (main body or attachment) for the message that is sent.

DBUSER

The username to use when connecting to the SMTP mail server. This parameter is optional.

FROM

The email address of the originator of the message. The default for this parameter is to use the value of the `com.infoengine.mail.originator` property. If the property does not exist, and the FROM parameter is not specified, an exception is thrown. This parameter is optional.

GROUP_OUT

The optional name of an output group. If this parameter is specified, then an output group is returned containing a single element with three attributes named **sent**, **unsent**, and **invalid**:

- The **sent** attribute specifies the addresses to which the message was sent.
- The **unsent** attribute specifies the addresses that are assumed to be valid but to which the message could not be sent (for example, because the mail server was unavailable).
- The **invalid** attribute specifies the addresses that were invalid.

HEADER

An optional header to add to the outgoing message. Multiple values can be specified for this parameter.

The format of the parameter is `name=value` (no quotation marks should be placed around `value`).

For example, the following value adds a `CC` header to the message even though the user is not actually included in the CC list, and does not receive the email:

```
cc=user@host.com
```

Tip

This parameter proves useful when sending an email to recipients in multiple languages. When recipient groups have different language locales set, Windchill sends a separate email for each locale. However, sending a separate email per language means that the recipient list only displays users who share the same locale. To prevent confusion as to whether disparate locale groups have been included in an email, you can set HEADER to `cc=user@host.com`.

HEADERS_CHARSET

The name of the character set that is used in email headers, such as the subject header.

MAIL_SERVER

The internet domain name of the SMTP mail server to which the message is sent for routing to the recipients. The default for this parameter is to obtain the mail server name from the `com.infoengine.mail.smtp.server` property. If the property does not exist, and MAIL_SERVER is not specified, the mail server defaults to `localhost`. This parameter is optional.

MAX_DELIVERY_TIME

The maximum length of time in seconds to take in attempting to deliver messages to valid addresses. This is for situations in which the mail transport indicates that the addresses are valid, but the mail server does not accept them (for example, because the mail server is temporarily unavailable or congested). The default value is 60 seconds.

MULTIPART_TYPE

The MIME subtype of the multipart message when multiple CONTENT and/or CONTENT_URL parameter values are specified. The default value for this parameter is `mixed`.

PASSWD

The password associated with the DBUSER parameter. This parameter is optional.

PASSWORD

The password to specify when authenticating to the SMTP mail server. While PASSWORD can be used in conjunction with USERNAME, it is not necessary for both parameters to be specified. The default for this parameter is to obtain the password from the `.mail.smtp.password` property. This parameter is optional.

REPLY_TO

The email address or addresses to which replies should be sent. Multiple values can be specified for this parameter. For each instance of the parameter, specify exactly one email address. The default for this parameter is for replies to be sent to the address specified by FROM. This parameter is optional.

SUBJECT

The subject header of the message. If it is not specified, the message does not contain a subject header. This parameter is optional.

TO

The email address or addresses of the primary recipients of the message. It can be specified multiple times to specify multiple primary recipients. Each instance of the parameter specifies exactly one address. This is a required parameter

USERNAME

The username to specify when authenticating to the SMTP mail server. While USERNAME can be used in conjunction with PASSWORD, it is not necessary for both parameters to be specified. The default for this parameter is for the username to be obtained from the `.mail.smtp.username` property. If the property does not exist, and USERNAME is not specified, an anonymous binding to the mail server is attempted. This parameter is optional.

EXAMPLE

The following example sends an email based on user input: .

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<ie:webject name="Send-Mail" type="MSG">
  <ie:param name="USERNAME" data="$(@FORM[]user[])"/>
  <ie:param name="PASSWORD" data="$(@FORM[]password[])"/>
  <ie:param name="TO" data="$(@FORM[]to[])"/>
  <ie:param name="FROM" data="$(@FORM[]from[])"/>
  <ie:param name="SUBJECT" data="$(@FORM[]subject[])"/>
  <ie:param name="CONTENT_TYPE"
```

```
                                   data="$(@FORM[]contentType[])"/>
     <ie:param name="CONTENT" data="$(@FORM[]content[])"/>
   </ie:webject>
```

To actually run this example you would need to provide a form where the **user**, **password**, **to**, **from**, **subject**, **contentType**, and **content** variables are identified.

Subscribe-Queue

Registers an event handler that is called by the MOM implementation when a message arrives in the specified queue.

Note

This webject can be used only after your environment has been set up for queuing tasks.

SYNTAX

```
<ie:webject name=Subscribe-Queue type="MSG">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="EXECUTE_TASK" data="task_name"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WHERE" data="jms_message_selector"/>
 <ie:param name="MAX_CONCURRENCY"="managed_receiver_generation"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		DBUSER
		SERVICE
		EXECUTE_TASK
		PASSWD
		QUEUE
		WHERE
		MAX_CONCURRENCY

DBUSER

The username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.msg.username` and `.jms.username` properties of the service that is used to connect to the MOM.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.msg.username` property is used. If the `.msg.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

EXECUTE_TASK

The task to be executed when a message is received in the subscribed queue. If the incoming message does not contain a task to execute, then the task specified on this parameter is run.

If EXECUTE_TASK is not specified, then the subscription can only handle Info*Engine requests on its queue. Other incoming messages are rejected. This parameter is optional.

MAX_CONCURRENCY

The number of receivers that should be generated upon queue subscription. This specifies the maximum number of messages that the Info*Engine system processes simultaneously. The default value is `1`, causing messages to be processed serially.

Specifying a MAX_CONCURRENCY value greater than one means that messages that are placed in a queue are not necessarily processed in the order in which they were queued. This parameter should not be used if the order of messages is important. This parameter is optional.

Note

Specifying this parameter does not necessarily mean that all receivers are kept busy when given a queue load. This functionality is partially dependent upon the configuration of the to which MOM you are connecting.

To illustrate this, Tibco EMS supports a queue property named “prefetch.” The default value for this property is `5`, meaning that any given receiver receives five messages at a time. Based on the amount of data in a queue, this can mean that if your MAX_CONCURRENCY is too high, then certain receivers might remain idle. For example, given the default queue configuration for Tibco EMS (five messages per receiver), if there were fifteen messages in a queue and upon subscription you specified a MAX_CONCURRENCY of `4`, then only three out of the four available receivers would actually be used (resulting in a real concurrency of three). Because the fifteen messages would be divided by the default prefetch value of five and distributed evenly to the first three receivers, the fourth would then be left idle. Setting a value for prefetch of `1` for the queue in question would result in all receivers staying busy, but if message sizes are typically large then this might mean that there is a lag between messages.

PASSWD

The password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.msg.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Info*Engine messaging software are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.msg.password` property is used. If the `.msg.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

QUEUE

The LDAP distinguished name of an administered queue. The value can be an LDAP distinguished name relative to a configured base URI or a fully qualified LDAP distinguished name. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultExecutionQueue` property. If there is no such value, then the QUEUE parameter must be specified. This parameter is optional.

SERVICE
The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

WHERE
A properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select a subset of messages to execute from the specified queue. The default for this parameter is to subscribe to all messages that arrive in the queue. This parameter is optional.

EXAMPLE

The following example registers a subscription to listen to the specified queue:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<ie:webject name="Subscribe-Queue" type="MSG">
  <ie:param name="QUEUE" data="$(@FORM[]queue[0])"/>
  <ie:param name="WHERE" data="$(@FORM[]where[0])"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the **queue** and **where** variables are identified.

Unsubscribe-Queue

Unregisters an event handler that is called by the MOM implementation when a message arrives in a specified queue.

Note

This webject can be used only after your environment has been set up for queuing tasks.

SYNTAX

```
<ie:webject name=Unsubscribe-Queue type="MSG">
  <ie:param name="QUEUE" data="managed_queue_name"/>
  <ie:param name="WHERE" data="jms_message_selector"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
		GROUP_OUT
		QUEUE
		WHERE
		SERVICE

GROUP_OUT

The name of the status group to create upon success. The default behavior for this parameter is to create a group named “unsubscribe-results”. This parameter is optional.

QUEUE

The LDAP distinguished name of an administered queue. The value is an LDAP distinguished name relative to a configured base URI. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified. The default for this parameter is to use the value specified in the `com.infoengine.msg.defaultExecutionQueue` property. If there is no such value, then the QUEUE parameter must be specified. This parameter is optional.

WHERE

A properly formatted JMS message selector, as defined in the Sun Java Message Service specification, used to select the queue listener to unsubscribe from the specified queue. The default for this parameter is to unsubscribe the queue listener associated with no message selector. This parameter is optional.

SERVICE

The name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM). This parameter is optional.

EXAMPLE

The following example cancels a subscription listening to the specified queue:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<ie:webject name="Unsubscribe-Queue" type="MSG">
  <ie:param name="QUEUE" data="$(@FORM[]queue[0])"/>
</ie:webject>
```

To actually run this example you would need to provide a form where the **queue** variable is identified.

Web Event Service Webjects

The following Web Event Service (WES) webjects can be used in conjunction with a third-party MOM for handling Info*Engine events:

- Emit-Event on page 486
- Subscribe-Event on page 492
- Unsubscribe-Event on page 494

All Web Event Service webjects use the `WES` **type** attribute value in the **webject** tag.

Note

These webjects can be used only after your environment has been set up to use the Web Event Service.

Emit-Event

DESCRIPTION

Causes Info*Engine to send a named event through the Java Messaging Service (JMS) to be handled by any subscribers. When the execution of the webject completes successfully, an empty status group with a status of 0 is returned with a message stating that the event was successfully sent. If the execution of the webject is unsuccessful, an appropriate exception is thrown.

SYNTAX

```
<ie:webject name="Emit-Event" type="WES">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="EVENT" data="my_event"/>
  <ie:param name="GROUP_IN" data="group1"/>
  <ie:param name="GROUP_OUT" data="results"/>
  <ie:param name="PASSWD" data="password"/>
  <ie:param name="PRIORITY" data="numeric_value"/>
  <ie:param name="PROPERTY" data="name=value"/>
  <ie:param name="PUT_BOOLEAN" data ="[TRUE | FALSE]"/>
  <ie:param name="PUT_BYTE" data ="byte"/>
  <ie:param name="PUT_BYTES" data ="bytes"/>
  <ie:param name="PUT_CHAR" data ="char"/>
  <ie:param name="PUT_DOUBLE" data ="double"/>
  <ie:param name="PUT_FLOAT" data ="float "/>
  <ie:param name="PUT_INT" data ="int"/>
  <ie:param name="PUT_LONG" data ="long"/>
  <ie:param name="PUT_SHORT" data ="short"/>
  <ie:param name="PUT_STRING" data ="string"/>
  <ie:param name="TIME_TO_LIVE" data="minutes"/>
  <ie:param name="TYPE" data="[InfoEngine | BytesMessage |
         TextMessage | MapMessage | Unknown]"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
EVENT	PUT_BOOLEAN	DBUSER
	PUT_BYTE	GROUP_IN
	PUT_BYTES	GROUP_OUT
	PUT_CHAR	PASSWD
	PUT_DOUBLE	PRIORITY
	PUT_FLOAT	PROPERTY
	PUT_INT	TIME_TO_LIVE
	PUT_LONG	SERVICE
	PUT_SHORT	
	PUT_STRING	
	TYPE	

DBUSER

Specifies the username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.wes.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Web Event Service (WES) are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.wes.username` property is used. If the `.wes.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

EVENT

Specifies the LDAP distinguished name of the event to emit. The value is an LDAP distinguished name relative to a configured base URI. A syntax of `<event_name>@<domain>` is recommended to give additional meaning and avoid name conflicts. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified.

This parameter is required.

GROUP_IN

Specifies the name of a group to be sent along with the event. This parameter should only be specified if the TYPE is `InfoEngine`. If GROUP_IN is specified and the TYPE is set to a different value, then GROUP_IN is ignored.

Multiple GROUP_IN values may be specified. A value of "*" causes the entire VDB to be sent. The default for this parameter is to send an empty VDB collection.

This parameter is optional.

GROUP_OUT

Specifies the name of the status group to create upon success. The default for this parameter is to create a group named `emit-results`.

This parameter is optional.

PASSWD

Specifies the password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.wes.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information about credentials mapping, see Credentials Mapping on page 125.

If this parameter is omitted, the mapped credentials for the Web Event Service (WES) are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.wes.password` property is used. If the `.wes.password` property is not set, then the value set in the `.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

TIME_TO_LIVE

Specifies in minutes how long the event should be valid. The default value for this parameter is 0, which is a special case indicating that the message does not expire.

This parameter is optional.

SERVICE

Specifies the name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM).

This parameter is optional.

PRIORITY

Specifies an integer priority to be set on the outgoing messages. The valid range for JMS Queue priorities is from 0 – 9, with 4 the typical default value. Priorities are ignored by Info*Engine, but can be used by third-party software receiving events sent by Info*Engine. This parameter is optional.

PROPERTY

Specifies a name and value pair to set on the outgoing JMS message. These properties are ignored by Info*Engine, but can be used by third-party software. The format for the PROPERTY value is

```
name=value
```

where:

`name`—The property name.

`value`—The corresponding value of the property.

The default behavior for this parameter is to not set any additional properties on the outgoing message.

This parameter is optional.

PUT_BOOLEAN

Places a Java Boolean value into the JMS message. The value may be specified as `True` or `False`. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_BYTE

Places a single byte into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_BYTES

Places multiple bytes into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_CHAR

Places a single Java character into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_DOUBLE

Places a Java double value into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_FLOAT

Places a floating point number into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_INT

Places an integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_LONG

Places a long integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_SHORT

Places a short integer into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage** or **MapMessage**.

PUT_STRING

Places an instance of `java.lang.String` into the JMS message. This parameter can only be used if the TYPE is specified as **BytesMessage**, **TextMessage**, or **MapMessage**.

TYPE

Specifies the type of message to be sent by selecting the processor used to generate the outgoing message. Message types are case-sensitive.

The default value for this parameter is `InfoEngine`, which means that an Info*Engine request is serialized and placed in the queue. Serialized in this case can mean actual Java serialization or serialization of an Info*Engine request to XML (this is the case when BLOB data needs to be queued as well).

Possible message types are:

- **InfoEngine**—Used for Info*Engine to Info*Engine communication. VDB groups serialized are governed by the GROUP_IN parameter.
- **BytesMessage**—For outgoing messages, this type supports several PUT_ type parameters for building a message that may be specific to a third-party recipient. Incoming messages are processed in the same manner as **Unknown**. The contents of a **BytesMessage** is supplied as BLOB data on an input stream to the calling task.
- **TextMessage**—Supports only a PUT_STRING parameter. Multiple PUT_STRING parameter values are concatenated together to create the outgoing **TextMessage**.
- **MapMessage**—For outgoing messages, this type supports several PUT_ type parameters with values in the form of *`name=data`*, where *`name`* is the key used to place *`data`* in the **MapMessage**.
- **Unknown**—Turns BLOB data into a **BytesMessage**. For incoming messages, a new, empty Info*Engine request is created and the contents of the message are attached as BLOB data.

The following table indicates which PUT_type parameters can be specified with which message types:

	BytesMessage	TextMessage	MapMessage
PUT_BOOLEAN	X		X
PUT_BYTE	X		X
PUT_BYTES	X		X
PUT_CHAR	X		X
PUT_DOUBLE	X		X
PUT_FLOAT	X		X
PUT_INT	X		X
PUT_LONG	X		X
PUT_SHORT	X		X
PUT_STRING	X	X	X

Note

Support for the type STRING in **BytesMessage** actually causes the value of the string to be written as a byte array; it is not actually written as a string.

EXAMPLE

The following example emits a specified event to be handled by any subscribers:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                              prefix="ie"%>

<ie:webject name="Emit-Event" type="WES">
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="GROUP_IN" data="$(@FORM[]group_in[])"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"/>
  <ie:param name="EVENT" data="$(@FORM[]event[])"/>
  <ie:param name="TIME_TO_LIVE" data="0"/>
  <ie:param name="PRIORITY" data="$(@FORM[]priority[])"/>
</ie:webject>
```

To actually run this example, you need to provide a form where the DBUSER, PASSWD, GROUP_IN, GROUP_OUT, EVENT, and PRIORITY variables are identified.

Subscribe-Event

DESCRIPTION

Causes the Info*Engine process where this webject is executed to subscribe to listen for a particular event. When an event is caught, Info*Engine builds an execution environment from the request or available data and executes a configured task. When the execution of the webject completes successfully, an empty status group with the status of 0 is returned with a message relaying successful subscription to the event. If the execution of the webject completes unsuccessfully, an appropriate exception is thrown.

SYNTAX

```
<ie:webject name="Subscribe-Event" type="WES">
  <ie:param name="DBUSER" data="username"/>
  <ie:param name="EVENT" data="event_name"/>
  <ie:param name="EXECUTE_TASK" data="task_name"/>
  <ie:param name="FAILURE_TASK" data="task_name"/>
  <ie:param name="GROUP_IN" data="event_handlers"/>
  <ie:param name="GROUP_OUT" data="results"/>
  <ie:param name="PASSWD" data="password"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
	EVENT	DBUSER
	EXECUTE_TASK	GROUP_OUT
	FAILURE_TASK	PASSWD
	GROUP_IN	SERVICE

DBUSER

Specifies the username to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over any existing credentials mapping for the authenticated user, or any value specified in the `.wes.username` and `.jms.username` properties of the service that is used to connect to the MOM. For more information, see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Web Event Service (WES) are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither of these mapped credentials are found, then the value set in the `.wes.username` property is used. If the `.wes.username` property is not set, then the value set in the `.jms.username` property is used. If neither property is set and you do not specify a value for this DBUSER parameter, Info*Engine does not specify a username when connecting to the MOM, and an anonymous connection is attempted.

This parameter is optional.

EVENT

Indicates the LDAP distinguished name of the event for which to listen. The value is an LDAP distinguished name relative to a configured base URI. A syntax of `<event_name>@<domain>` is recommended to give additional meaning and avoid name conflicts. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified.

This parameter is required if GROUP_IN is not supplied.

EXECUTE_TASK

Indicates the task to be executed upon event notification. This parameter is required if GROUP_IN is not supplied.

FAILURE_TASK

Indicates the task to be executed if EXECUTE_TASK is not executed. This parameter can only be specified if GROUP_IN is not specified. The default for this parameter is to log the error.

GROUP_IN

Specifies the name of a group from which to extract EVENT, EXECUTE_TASK and FAILURE_TASK. The group must have EVENT and EXECUTE_TASK columns. It can also have a FAILURE_TASK column. A subscription occurs for each row in the input group. This parameter is required if EVENT and EXECUTE_TASK are not specified.

GROUP_OUT

Specifies the name of the status group to be created upon success. The default for this parameter is to create a group named `subscription-results`. This parameter is optional.

PASSWD

Specifies the password to use when connecting to the MOM. If this parameter is specified, the parameter value takes precedence over existing credentials mapping for the authenticated user, or any value specified in the `.wes.password` and `.jms.password` properties of the service that is used to connect to the MOM. For more information see Credentials Mapping for MOMs on page 131.

If this parameter is omitted, the mapped credentials for the Web Event Service (WES) are used. If no such mapping is found, then the mapped credentials for the Java Messaging Service (JMS) are used. If neither such mapped credentials are found, then the value set in the `.wes.password` property is used. If the `.wes.password` property is not set, then the value set in the

`.jms.password` property is used. If neither property is set and you do not specify a value for this PASSWD parameter, Info*Engine does not specify a password when connecting to the MOM.

This parameter is optional.

SERVICE

Specifies the name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM).

This parameter is optional.

EXAMPLE

The following example subscribes the current Info*Engine process to the specified event:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<ie:webject name="Subscribe-Event" type="WES">
  <ie:param name="DBUSER" data="$(@FORM[]dbuser[])"/>
  <ie:param name="PASSWD" data="$(@FORM[]passwd[])"/>
  <ie:param name="EVENT" data="$(@FORM[]event[])"/>
  <ie:param name="GROUP_IN" data="$(@FORM[]group_in[])"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"/>
</ie:webject>
```

To actually run this example, you need to provide a form where the DBUSER, PASSWD, EVENT, GROUP_IN, and GROUP_OUT variables are identified.

Unsubscribe-Event

DESCRIPTION

Causes Info*Engine to cease listening for a particular event that is currently being monitored. When the execution of the webject completes successfully, an empty status group with the status of 0 is returned with a message relaying successful unsubscription from the event. If the execution of the webject completes unsuccessfully, an appropriate exception is thrown.

SYNTAX

```
<ie:webject name="Unsubscribe-Event" type="WES">
  <ie:param name="EVENT" data="event_name"/>
  <ie:param name="GROUP_OUT" data="results"/>
</ie:webject>
```

PARAMETERS

Required	Select	Optional
EVENT		GROUP_OUT
		SERVICE

EVENT

Specifies the LDAP distinguished name of the event from which to unsubscribe. The value is an LDAP distinguished name relative to a configured base URI. A syntax of `<event_name>@<domain>` is recommended to give additional meaning and avoid name conflicts. If relative, the **cn**= (common name attribute) is implicit if not explicitly specified.

This parameter is required if GROUP_IN is not supplied.

GROUP_OUT

Specifies the name of the status group to create upon success. The default behavior for this parameter is to create a group named `unsubscribe-results`.

This parameter is optional.

SERVICE

Specifies the name of the Info*Engine property set that is configured for connectivity to a specific JMS service. This allows you to configure more than one JMS service per Info*Engine Virtual Machine (VM).

This parameter is optional.

EXAMPLE

The following example unsubscribes the current Info*Engine process to the specified event:

```
<%@page language="java" session="false"%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
                                               prefix="ie"%>

<ie:webject name="Unsubscribe-Event" type="WES">
  <ie:param name="EVENT" data="$(@FORM[]event[])"/>
  <ie:param name="GROUP_OUT" data="$(@FORM[]group_out[])"/>
</ie:webject>
```

To actually run this example, you need to provide a form where the EVENT and GROUP_OUT variables are identified.