Licensed Third-Party Technology Information

Certain PTC software products contain licensed third-party technology: Rational Rose 2000E is copyrighted software of Rational Software Corporation. RetrievalWare is copyrighted software of Convera Corporation. VisualCafé is copyrighted software of WebGain, Inc. VisTools library is copyrighted software of Visual Kinematics, Inc. (VKI) containing confidential trade secret information belonging to VKI. HOOPS graphics system is a proprietary software product of, and is copyrighted by, Tech Soft America, Inc. G-POST is copyrighted software and a registered trademark of Intercim. VERICUT is copyrighted software and a registered trademark of CGTech. Pro/PLASTIC ADVISOR is powered by Moldflow technology. Moldflow is a registered trademark of Moldflow Corporation. The JPEG image output in the Pro/Web.Publish module is based in part on the work of the independent JPEG Group. DFORMD.DLL is copyrighted software from Compaq Computer Corporation and may not be distributed. METIS, developed by George Karypis and Vipin Kumar at the University of Minnesota, can be researched at http://www.cs.umn.edu/~karypis/metis. METIS is © 1997 Regents of the University of Minnesota. LightWork Libraries are copyrighted by LightWork Design 1990-2001. Visual Basic for Applications and Internet Explorer is copyrighted software of Microsoft Corporation. Adobe Acrobat Reader is copyrighted software of Adobe Systems. Parasolid © Electronic Data Systems (EDS). Windchill Info*Engine Server contains IBM XML Parser for Java Edition and the IBM Lotus XSL Edition. Pop-up calendar components Copyright © 1998 Netscape Communications Corporation. All Rights Reserved. TECHNOMATIX is copyrighted software and contains proprietary information of Technomatix Technologies Ltd. Apache Server, Tomcat, Xalan, and Xerces are technologies developed by, and are copyrighted software of, the Apache Software Foundation (http://www.apache.org/) - their use is subject to the terms and limitations at: http://www.apache.org/LICENSE.txt. UnZip (© 1990-2001 Info-ZIP, All Rights Reserved) is provided "AS IS" and WITHOUT WARRANTY OF ANY KIND. For the complete Info-ZIP license see
ftp://ftp.info-zip.org/pub/infozip/license.html. Gecko and Mozilla components are subject to the Mozilla Public License Version 1.1 at http://www.mozilla.org/MPL/. Software distributed under the MPL is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the MPL for the specific language governing rights and limitations. Technology "Powered by Groove" is provided by Groove Networks, Inc. Technology "Powered by WebEx" is provided by WebEx Communications, Inc. Acrobat Reader is Copyright © 1998 Adobe Systems Inc. Oracle 8i run-time, Copyright © 2000 Oracle Corporation. The Java™ Telnet Applet (StatusPeer.java, TelnetIO.java, TelnetWrapper.java, TimedOutException.java), Copyright © 1996, 97 Mattias L. Jugel, Marcus Meißner, is redistributed under the GNU General Public License. This license is from the original copyright holder and the Applet is provided WITHOUT WARRANTY OF ANY KIND. You may obtain a copy of the source code for the Applet at http://www.mud.de/se/jta (for a charge of no more than the cost of physically performing the source distribution), by sending e-mail to leo@mud.de or marcus@mud.de-you are allowed to choose either distribution method. The source code is likewise provided under the GNU General Public License. GTK+The GIMP Toolkit are licensed under the GNU LPGL. You may obtain a copy of the source code at http://www.gtk.org/, which is likewise provided under the GNU LPGL. zlib software Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND
This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) (OCT'95) or DFARS 227.7202-1(a) and 227.7202-3(a) (JUN'95), is provided to the US Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 (OCT'88) or Commercial Computer Software-Restricted Rights at FAR 52.227-19(c)(1)-(2) (JUN'87), as applicable.

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA
Contents

Change Record ........................................................................................................... xiii
About This Guide ....................................................................................................... xv
    Related Documentation ............................................................................................ xvi
    Technical Support .................................................................................................. xvi
    Documentation for PTC Products ............................................................................. xvii
    Comments ............................................................................................................... xvii
    Documentation Conventions .................................................................................... xvii
Architectural Overview ......................................................................................... 1-1
    Identifying the Info*Engine Components .............................................................. 1-2
    Identifying Basic Configurations .......................................................................... 1-2
    Interacting with Info*Engine ............................................................................... 1-3
        Using a Custom Java Application ................................................................. 1-4
        Using a Web Server to Process Info*Engine Requests .................................... 1-5
    Making E-Mail Requests to Info*Engine ............................................................. 1-8
    Managing the Execution of Info*Engine Tasks .................................................... 1-8
        Requesting Task Execution Through a TCP/IP Connection ............................ 1-10
        Implementing Events that Execute Tasks ...................................................... 1-11
        Queuing Tasks for Execution ........................................................................ 1-12
    Starting and Locating Info*Engine Components .............................................. 1-13
    Setting Up Connections Through Adapters ....................................................... 1-14
        Using In-Process Adapters and Gateways ...................................................... 1-15
        Using Out-of-Process Adapters and Gateways ............................................ 1-16
How Info*Engine Manages Data .......................................................................... 2-1
    An Introduction to Info*Engine Data Management ............................................ 2-2
        JSP Pages and Tasks ..................................................................................... 2-2
        Webjects .......................................................................................................... 2-3
        Groups of Objects and the Virtual Database .................................................. 2-5
    Managing and Manipulating Groups with Info*Engine .................................... 2-8
        Manipulating More than One Group and Displaying the Resulting Group .... 2-8
        Manipulating and Displaying Multiple Groups in the VDB ............................ 2-11
        Sending Data Groups to Adapters .................................................................. 2-13
        Retrieving Data Groups from Adapters .......................................................... 2-13
Context Groups ........................................................................................................................................ 2-14
Managing Large Amounts of Data ........................................................................................................ 2-15
Managing Large Groups ......................................................................................................................... 2-15
Managing BLOBs ..................................................................................................................................... 2-16

Info*Engine JSP Pages ............................................................................................................................ 3-1
Authoring Info*Engine JavaServer Pages ............................................................................................... 3-2
Storing Info*Engine JSP Pages ................................................................................................................ 3-2
Accessing Info*Engine JSP Pages ............................................................................................................ 3-3
Creating Info*Engine JSP Pages ............................................................................................................. 3-4
Using Info*Engine Custom Tags ............................................................................................................ 3-6
Using Adapter Webjects ........................................................................................................................... 3-7
Using Task Webjects .................................................................................................................................. 3-12
Using Display Webjects ........................................................................................................................... 3-13
Specifying URLs and URLs .................................................................................................................. 3-15
Executing Multiple Webjects at the Same Time ...................................................................................... 3-18
Accounting for Webject Success or Failure .......................................................................................... 3-19
Catching Exceptions ................................................................................................................................ 3-21
Using an Error Page ................................................................................................................................ 3-21
Using unit Tags ........................................................................................................................................ 3-21
Executing Tasks From JSP Pages ............................................................................................................. 3-23

Info*Engine Tasks .................................................................................................................................. 4-1
Info*Engine Tasks .................................................................................................................................. 4-2
Authoring Info*Engine Tasks .................................................................................................................. 4-3
The Parts of a Task .................................................................................................................................... 4-5
Creating Tasks .......................................................................................................................................... 4-6
Executing Tasks ......................................................................................................................................... 4-7
Basic Rules for Constructing Info*Engine Tasks ..................................................................................... 4-10
XML Output for Info*Engine Groups ...................................................................................................... 4-12
Document Declaration .............................................................................................................................. 4-12
Info*Engine Groups ................................................................................................................................. 4-13
Info*Engine Objects ................................................................................................................................. 4-13
Metadata In XML Output .......................................................................................................................... 4-15
Nesting Tasks ........................................................................................................................................... 4-16
Include-Task Webject Syntax and Rules ................................................................................................. 4-16
task Tag Rules for Nesting Tasks ............................................................................................................ 4-17
A Simple Nested Task Example ............................................................................................................... 4-17

Alternatives to Basic Task and JSP Page Creation .................................................................................... 5-1
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Webobjects</td>
<td>6-31</td>
</tr>
<tr>
<td>Required Attributes for Custom Webobjects</td>
<td>6-31</td>
</tr>
<tr>
<td>Creating an External Custom Webject</td>
<td>6-32</td>
</tr>
<tr>
<td>Using Double-Byte Characters</td>
<td>6-39</td>
</tr>
<tr>
<td>Packages</td>
<td>6-40</td>
</tr>
<tr>
<td>Providing LDAP Directory Information</td>
<td>6-40</td>
</tr>
<tr>
<td>Managing Packages</td>
<td>6-42</td>
</tr>
<tr>
<td>Creating Packages</td>
<td>6-45</td>
</tr>
<tr>
<td>Installing Packages</td>
<td>6-48</td>
</tr>
<tr>
<td>Uninstalling Packages</td>
<td>6-49</td>
</tr>
<tr>
<td>About the Info*Engine J2EE Connector</td>
<td>6-51</td>
</tr>
<tr>
<td>JCA Contracts and the Common Client Interface</td>
<td>6-52</td>
</tr>
<tr>
<td>Transaction Contract</td>
<td>6-53</td>
</tr>
<tr>
<td>Security Contract</td>
<td>6-54</td>
</tr>
<tr>
<td>Common Client Interface (CCI)</td>
<td>6-54</td>
</tr>
<tr>
<td>Standalone Java SOAP Clients</td>
<td>6-56</td>
</tr>
<tr>
<td>Assumed Knowledge</td>
<td>6-56</td>
</tr>
<tr>
<td>Before You Begin</td>
<td>6-57</td>
</tr>
<tr>
<td>Implementing Tasks and Java Classes Example</td>
<td>6-57</td>
</tr>
<tr>
<td>Registering Delegates</td>
<td>6-57</td>
</tr>
<tr>
<td>Generating DAOs</td>
<td>6-58</td>
</tr>
<tr>
<td>Making a Connection</td>
<td>6-60</td>
</tr>
<tr>
<td>Managing Connection Factories in the LDAP Directory</td>
<td>6-61</td>
</tr>
<tr>
<td>Putting It All Together</td>
<td>6-65</td>
</tr>
<tr>
<td>J2EE SOAP Clients</td>
<td>6-67</td>
</tr>
<tr>
<td>Writing a J2EE SOAP Client</td>
<td>6-67</td>
</tr>
<tr>
<td>Automatic EJB Generation</td>
<td>6-68</td>
</tr>
<tr>
<td>Useful Client Side EJB Design Patterns</td>
<td>6-70</td>
</tr>
<tr>
<td>Custom Applications</td>
<td>7-1</td>
</tr>
<tr>
<td>SAK Example Application</td>
<td>7-2</td>
</tr>
<tr>
<td>Compiling the Application</td>
<td>7-2</td>
</tr>
<tr>
<td>Running the Application</td>
<td>7-2</td>
</tr>
<tr>
<td>Application Source</td>
<td>7-3</td>
</tr>
<tr>
<td>Bookstore Example</td>
<td>7-6</td>
</tr>
<tr>
<td>Introductory Topics</td>
<td>7-6</td>
</tr>
<tr>
<td>Bookstore Example Setup</td>
<td>7-7</td>
</tr>
<tr>
<td>Survey of JSP Pages</td>
<td>7-9</td>
</tr>
<tr>
<td>Survey of XML Tasks</td>
<td>7-10</td>
</tr>
<tr>
<td>Database Tables</td>
<td>7-11</td>
</tr>
</tbody>
</table>
Display Webject Reference ................................................................. 9-1

- Display Webjects for HTML .......................................................... 9-2
  - Apply-XSL .................................................................................. 9-3
  - Display-Object ........................................................................ 9-11
  - Display-Resource .................................................................... 9-23
  - Display-Selection .................................................................... 9-25
  - Display-Table .......................................................................... 9-37
  - Display-Value .......................................................................... 9-44
  - Display-XML ........................................................................... 9-54
  - Echo-Request .......................................................................... 9-56

- Image Webjects for JPEG ................................................................. 9-58
  - Bar-Graph .............................................................................. 9-59
  - Line-Graph .............................................................................. 9-64
  - Pie-Chart ................................................................................. 9-69

Task Webject Reference ..................................................................... 10-1

- Administrative Webjects ................................................................. 10-3
  - Get-Statistics .......................................................................... 10-4
  - Reload-Properties .................................................................. 10-5

- Group Webjects ............................................................................ 10-6
  - Call-Task ................................................................................ 10-7
  - Change-Group ........................................................................ 10-9
  - Concat-Groups ....................................................................... 10-11
  - Copy-Group ........................................................................... 10-13
  - Create-Group ......................................................................... 10-16
  - Diff-Groups ............................................................................ 10-23
  - Extract-Group ........................................................................ 10-27
  - Format-Group ......................................................................... 10-29
  - Intersect-Groups ..................................................................... 10-32
  - Join-Groups ............................................................................. 10-35
  - Merge-Groups ......................................................................... 10-41
  - Return-Groups ........................................................................ 10-44
  - Set-Identity ............................................................................. 10-47
  - Set-Metadata ........................................................................... 10-50
  - Sort-Group .............................................................................. 10-55
  - Subset-Group .......................................................................... 10-61
  - Summarize-Groups .................................................................. 10-64
  - Translate-Group ....................................................................... 10-66
  - Union-Groups .......................................................................... 10-72
## Change Record

Table 1 Changes for Release 7.0

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 5, <a href="#">Alternatives to Basic Task and JSP Page Creation</a></td>
<td>Using SOAP Requests - This section has been updated and expanded to include information on the SOAP RPC Servlet, SOAP task requirements, and writing tasks for use with SOAP.</td>
</tr>
<tr>
<td>Chapter 6, <a href="#">Advanced User Topics</a></td>
<td>Packages - This new section introduces packages, and the Package Manager. About the Info<em>Engine J2EE Connector - This new section introduces the Info</em>Engine J2EE Connector, how it works, and how it is used. Standalone SOAP Clients - This new section discusses writing standalone SOAP clients to be used with the Info<em>Engine J2EE Connector. J2EE SOAP Clients - This new section discusses writing J2EE SOAP clients to be used with the Info</em>Engine J2EE Connector.</td>
</tr>
<tr>
<td>Chapter 7, <a href="#">Custom Applications</a></td>
<td>Info<em>Engine J2EE Book Viewer Example - This new section introduces the Info</em>Engine J2EE book viewer example, an example of a J2EE SOAP client.</td>
</tr>
<tr>
<td>Chapter 9, <a href="#">Display Webobject Reference</a></td>
<td>Added new webobjects, updated existing webobjects.</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Chapter 10, Task Webject Reference</td>
<td>Added new webjects, updated existing webjects.</td>
</tr>
</tbody>
</table>
The *Info* Engine User’s Guide documents the use of the Info*Engine software. The chapters are organized so that successive chapters build on earlier chapters. The guide is made up of the following chapters and appendices:

- **Architectural Overview** introduces the Info*Engine architecture, describing how the components of the Info*Engine software work in concert to provide users with information.

- **How Info*Engine Manages Data** describes how Info*Engine manages data, specifically introducing authors of Info*Engine solutions to the concepts of JSP pages and tasks, the mainstay of Info*Engine requests.

- **Info*Engine JSP Pages** and **Info*Engine Tasks** discuss JavaServer Pages (JSP) and tasks respectively and describe how to use both to create data requests and manipulate data. Specifically, these chapters discuss the syntax and format of JSP pages and tasks.

- **Alternatives to Basic Task and JSP Page Creation** introduces alternative methods of accessing and manipulating data through Info*Engine. These alternatives allow advanced and expert solution authors to bypass traditional task and page creation when complex solutions are required.

- **Advanced User Topics** describes advanced functionality of Info*Engine.

- **Custom Applications** documents the example applications that are provided.

- **Info*Engine Custom Tag Reference** contains reference topics for the tags in the Info*Engine core and directory tag libraries. It also contains an overview of common JSP element types with tips for using the 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 webjects associated with Info*Engine tasks.

This guide assumes you are familiar with the basics of HTML and XML as defined by the World Wide Web Consortium (http://www.w3c.org). To take
advantage of the advanced functionality of Info*Engine you must have a basic knowledge of not only HTML and XML, but also JSP.

Related Documentation

The following documentation may be helpful:

- The *Info*Engine Installation and Configuration Guide* details the procedure for installing and configuring the Info*Engine* components.
- The *Info*Engine Java Adapter Development Kit Programming Reference* describes the how to develop your own native Info*Engine* adapters using the Java programming language if the PTC adapter library does not contain an adapter suited to your needs.
- The *Info*Engine C Adapter Development Kit Programming Reference* describes the how to develop your own non-native Info*Engine* adapters using the C programming language if the PTC adapter library does not contain an adapter suited to your needs.

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

Technical Support

Contact PTC Technical Support via the PTC Web site, phone, fax, or e-mail if you encounter problems using Windchill.

For complete details, refer to Contacting Technical Support in the *PTC Customer Service Guide* enclosed with your shipment. This guide can also be found under the Support Bulletins section of the PTC Web site at:

http://www.ptc.com/support/index.htm

The PTC Web site also provides a search facility that allows you to locate Technical Support technical documentation of particular interest. To access this page, use the following link:

http://www.ptc.com/support/support.htm

You must have a Configuration ID before you can receive technical support. If you do not have an ID, contact PTC License Management using the instructions found in your *PTC Customer Service Guide* under Contacting License Management.
Documentation for PTC Products

PTC provides documentation in the following forms:

- Help topics
- PDF books

All books are available in PDF format on product CDs. To view and print PDF books, you must have the Adobe Acrobat Reader installed.

All Windchill documentation is included on the CD for the application. In addition, books updated after release (for example, to support a hardware platform certification) are available from the Reference Documents section of the PTC Web site at the following URL:

http://www.ptc.com/cs/doc/reference/

Comments

PTC welcomes your suggestions and comments on its documentation--send comments to the following address:

documentation@ptc.com

Please include the name of the application and its release number with your comments. For online books, provide the book title.

Documentation Conventions

Windchill documentation uses the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Names of elements in the user interface such as buttons, menu paths,</td>
<td>Click OK.</td>
</tr>
<tr>
<td></td>
<td>and dialog box titles.</td>
<td>Select File &gt; Save.</td>
</tr>
<tr>
<td></td>
<td>Required elements and keywords or characters in syntax formats.</td>
<td>License File dialog box</td>
</tr>
<tr>
<td></td>
<td></td>
<td>create_&lt;tablename&gt;_.sql</td>
</tr>
<tr>
<td><strong>Italic</strong></td>
<td>Variable and user-defined elements in syntax formats. Angle brackets</td>
<td>create_&lt;tablename&gt;_.sql</td>
</tr>
<tr>
<td></td>
<td>(&lt;&gt; ) enclose individual elements.</td>
<td></td>
</tr>
<tr>
<td><strong>Monospace</strong></td>
<td>Examples</td>
<td>JavaGen &quot;wt.doc.*&quot; F true</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>Processing completed.</td>
</tr>
<tr>
<td><strong>&quot;Quotation</strong></td>
<td>Strings</td>
<td>The string &quot;UsrSCM&quot;...</td>
</tr>
<tr>
<td><strong>marks&quot;</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convention</td>
<td>Item</td>
<td>Example</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>!</td>
<td>The CAUTION symbol indicates potentially unsafe situations which</td>
<td>When you add a value to an enumerated type (for example, by adding a role in the RolesRB.java resource file), removing that value can result in a serious runtime error. Do not remove a role unless you are certain there is no reference to it within the system.</td>
</tr>
<tr>
<td></td>
<td>may result in minor injury, machine damage or downtime, or corruption or loss of software or data.</td>
<td></td>
</tr>
</tbody>
</table>
This chapter provides an overview of the Info*Engine architecture.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying the Info*Engine Components</td>
<td>1-2</td>
</tr>
<tr>
<td>Identifying Basic Configurations</td>
<td>1-2</td>
</tr>
<tr>
<td>Interacting with Info*Engine</td>
<td>1-3</td>
</tr>
<tr>
<td>Managing the Execution of Info*Engine Tasks</td>
<td>1-8</td>
</tr>
<tr>
<td>Starting and Locating Info*Engine Components</td>
<td>1-13</td>
</tr>
<tr>
<td>Setting Up Connections Through Adapters</td>
<td>1-14</td>
</tr>
</tbody>
</table>
Identifying the Info*Engine Components

The following components make up the Info*Engine architecture:

- The *Info*Engine Servlet provides an interface between the Web server and Info*Engine.

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

- 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.

- 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 Info*Engine Servlet, the Info*Engine server, and the E-Mail Broker use the SAK to invoke tasks and individual webjects.

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

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

- The *Info*Engine SOAP RPC Servlet catches and processes 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 E-Mail Broker provides a process by which users can e-mail Info*Engine requests to a mailbox. Using the SAK, the messages in the mailbox are then passed on to the Info*Engine server for processing.

The remainder of the chapter describes 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 many 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:

• Custom Java applications, including JavaServer Pages (JSP).
• Web servers that process Info*Engine requests. The requests can come from applications or Web browsers.
• E-mail requests that contain formatted messages sent to a predefined Info*Engine mailbox.
• Java Message Service (JMS) events and messages that queue Info*Engine tasks for execution.
• Custom third-party applications that make requests to execute Info*Engine tasks. These applications use the Info*Engine SOAP RPC servlet.

The following diagram shows how Info*Engine components and other customer software components can interact to execute Info*Engine code.
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. The webjects and tasks can be easily instantiated and invoked as Java objects from a Java application or in a text file. Info*Engine text files can be accessed using requests or code within an application.

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

Using a Custom Java Application

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.

Using a Web Server to Process Info*Engine Requests

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. After the installation is complete, your Info*Engine environment is set up so that Info*Engine requests to execute JSP and HTML pages coming from Web browsers are processed correctly.

By doing some additional Info*Engine configuration steps, you can set up your Info*Engine environment to process requests from the following additional sources:

- If you configure the Info*Engine SOAP RPC servlet, Info*Engine SOAP requests can come from third-party applications.

The steps required to configure Info*Engine so that it processes requests from these additional sources are described in the Configuration chapter of this guide.
Processing Web Browser Requests

The following diagram shows the relationships among the components that process Web browser requests for JSP pages.

![Diagram showing the relationships among components processing Web browser requests for JSP pages.]

This diagram shows the components that are used when the 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 servlet engine installed on your Web 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.

Web Browser Request Example

Assume that a user needs to compare the manufacturing bill of materials (MBOM) for an assembly with the engineering bill of materials (EBOM) for the same assembly. However, the as-designed EBOM exists on a Product Data Management (PDM) system and the as-built MBOM exists on an Enterprise Resource Planning (ERP) system.

Through Info*Engine, the information from these two disparate systems can be combined and displayed to the user at the same time. The first step is to author a JSP page that uses Info*Engine to request this information and then display the requested items in side-by-side frames within a browser window. Usually, this JSP page resides in the directory assigned to Info*Engine JSP pages. To display
the page, the user includes the assigned Info*Engine application URL prefix in the URL. For example, assume the following:

- The Web page is named bill_of_materials.jsp.
- The Web server is named Wsvr1.
- The Info*Engine URL prefix that identifies where JSP pages reside is Windchill.

To initiate the request for the bill_of_materials JSP page, a user sends the following URL using HTTP:

```
http://Wsvr1/Windchill/bill_of_materials.jsp
```

Because the URL contains the Windchill URL prefix, the Web server passes the URL on to the JSP engine. The JSP engine then finds and processes the JSP page. In this example, the JSP page contains standard JSP and HTML tags that format the page to display the bill of material data in side-by-side frames.

In addition to the standard tags, the JSP page contains custom Info*Engine tags that identify requests for the data that will fill out the bills of material. The custom tags also provide the information that formats the data that is returned from the requests. When the JSP engine encounters the custom tags, it passes these tags off to the SAK for processing.

When the SAK processes the custom tags on the JSP page, it performs the following tasks:

- Locates the information needed to access any requested applications (for example, the PDM or ERP system).
- Obtains any additional information that is required to retrieve, manipulate, and display the requested data.
- Identifies any authentication requirements (who can or cannot access the information requested) if you set up custom tags to do so.

To obtain the data, the SAK sends requests through the Info*Engine protocol to the appropriate Info*Engine adapters. After the appropriate PDM and ERP applications are located, the requested EBOM and MBOM data is retrieved and returned to the SAK. The SAK maintains the data as two separate groups in an internal, virtual database (VDB).

Using Info*Engine custom tags that the author has included in the JSP page, the data returned in these two groups is manipulated and formatted into standard HTML-encoded text. Then, the SAK returns the encoded text to the JSP engine. The JSP engine passes the text back to the Web server, which passes it back to the user’s browser to be displayed. As described earlier in this example, the display shows the data in the side-by-side frames that were set up for the EBOM and MBOM comparison originally requested by the user.
Making E-Mail Requests to Info*Engine

The E-Mail Broker allows users to make Info*Engine requests by e-mail.

The E-Mail Broker provides a process that monitors a mailbox for requests to execute Info*Engine tasks. When a request arrives in the mailbox, the E-Mail Broker connects to the Server and passes the request to the Info*Engine server for processing. It also captures output from the processed template or task, and returns the output in an e-mail message to the address specified in the From or Reply-To heading of the original request.

Managing the Execution of Info*Engine Tasks

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

- Info*Engine webjects that retrieve and manipulate data.
- Surrounding Info*Engine custom tags that manage the execution of the webjects.

There are two basic ways to execute tasks:

- Incorporate tasks directly into any Java application, including JSP pages, using Info*Engine custom tags.
- Put the tasks in individual text-based documents, specify which tasks to execute in the Info*Engine custom tags within a Java application (or JSP page).

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 will be completed, even if it is interrupted due to a system problem.
By performing the basic Info*Engine 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 Info*Engine 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 mail messages funneled through the Info*Engine E-Mail Broker, or from SOAP requests that come from third-party applications through the Info*Engine SOAP RPC servlet.
Implementing Events that Execute Tasks

The following diagram show 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 different 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 Info*Engine 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 RPC 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. For example, the Metaphase and Oracle adapters are non-native adapters.

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. For example, to retrieve information from a Metaphase database, you must install and configure the Metaphase adapter and, because this adapter is a non-native adapter, you must also configure a gateway for the adapter.
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 the in-process adapter).

- 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 upon the installation options.

**Using In-Process Adapters and Gateways**

In-process adapters and gateways are installed and run in the same Java Virtual Machine 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:

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. 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.
How Info*Engine Manages Data

This chapter introduces the concepts of JSP pages and tasks. The chapter briefly explains how these Info*Engine document types make use of webjects to influence how information appears and behaves. The chapter also explains how information is grouped and placed in a special holding area called the virtual database (VDB) when it is created and manipulated by JSP pages and tasks. Finally, information about context groups is explained.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>An Introduction to Info*Engine Data Management</td>
<td>2-2</td>
</tr>
<tr>
<td>Managing and Manipulating Groups with Info*Engine</td>
<td>2-8</td>
</tr>
<tr>
<td>Context Groups</td>
<td>2-14</td>
</tr>
<tr>
<td>Managing Large Amounts of Data</td>
<td>2-15</td>
</tr>
</tbody>
</table>
An Introduction to Info*Engine Data Management

All basic Info*Engine solutions take advantage of five fundamental concepts: JSP pages, tasks, webjects, groups, and the virtual database. 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 (VDB) 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.

If you used templates in a previous release, you can continue to use them as described in the Templates and Their Use appendix. However, consider using JSP pages or custom applications for all new development.

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 the VDB work with them will 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, you 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 will help you understand how Info*Engine manages data.

The following types of webjects exist:

- *Display* webjects transform Info*Engine groups into HTML or into another display-type markup language (such as WML) 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.

- **Image** webjects display JPEG images based on group data. The data is displayed in a graph or a chart.

- **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.

- **Administrative** webjects perform specific administrative functions, such as gathering simple statistics, or causing service properties to reload at runtime.

- **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.

- **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.

- **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.

- **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.

- **Web Event Service** webjects provide a set of webjects that can be used in conjunction with a third-party MOM for handling Info*Engine events.

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.

An 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 that follows:

<table>
<thead>
<tr>
<th>ename</th>
<th>phone</th>
<th>department</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td>Sales</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td>Sales</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td>Sales</td>
<td>Administrative Assistant</td>
</tr>
</tbody>
</table>

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. You can find more information about what is in a context group in the section entitled Context Groups that is later in this chapter.
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:

```xml
<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:

<table>
<thead>
<tr>
<th>ename</th>
<th>phone</th>
<th>department</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td>Sales</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td>Sales</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td>Sales</td>
<td>Administrative Assistant</td>
</tr>
</tbody>
</table>

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:

```html
<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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Telephone</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td>Administrative Assistant</td>
</tr>
</tbody>
</table>
Managing and Manipulating Groups with Info*Engine

Understanding how Info*Engine operates with more than just basic queries and displays of those queries will give you a greater understanding of how you can use Info*Engine to develop solutions at your site. Additional scenarios that build on the information earlier in this chapter can give you a clearer understanding of how you can manipulate data using Info*Engine.

Manipulating More than One Group and Displaying the Resulting Group

Expanding on the employee example described earlier in this chapter, let's examine how to obtain a single list of employees from two distinct departments. For example, the following scenario combines the employees from the sales department and the development department of your company. Also assume that each department has their employee lists in different locations of a database at the company. Then, 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 the Basic Group Creation and Display scenario to this page. 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:

```xml
<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:

<table>
<thead>
<tr>
<th>ename</th>
<th>phone</th>
<th>department</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td>Sales</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td>Sales</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td>Sales</td>
<td>Administrative Assistant</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>ename</th>
<th>phone</th>
<th>department</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anderson, Pat</td>
<td>873-2428</td>
<td>Development</td>
<td>Engineer</td>
</tr>
<tr>
<td>Stein, Chris</td>
<td>873-2608</td>
<td>Development</td>
<td>Manager</td>
</tr>
<tr>
<td>Wong, May</td>
<td>873-2741</td>
<td>Development</td>
<td>Engineer</td>
</tr>
</tbody>
</table>

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:

```xml
<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 one used in the Basic Group Creation and Display scenario:

```xml
<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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Telephone</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anderson, Pat</td>
<td>Development</td>
<td>873-2428</td>
<td>Engineer</td>
</tr>
<tr>
<td>Burton, Jack</td>
<td>Sales</td>
<td>873-2302</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>Sales</td>
<td>873-2200</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>Sales</td>
<td>873-3313</td>
<td>Administrative Assistant</td>
</tr>
<tr>
<td>Stein, Chris</td>
<td>Development</td>
<td>873-2608</td>
<td>Manager</td>
</tr>
<tr>
<td>Wong, May</td>
<td>Development</td>
<td>873-2741</td>
<td>Engineer</td>
</tr>
</tbody>
</table>
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 webobjects on the page and possibly to other webobjects called on other pages in the session (depending on the scope that you 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 a special webobject known as the Return-Groups webobject:

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

Using an example similar to the employee example described earlier in this chapter, let’s examine 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, let’s display 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 webobjects necessary to retrieve these two lists are the same webobjects that were used in the previous example:

```xml
<ie:webobject 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:webobject>

<ie:webobject 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:webobject>
```

As before, the first Query-Objects webobject asks for information about employees from the sales employee table and the second Query-Objects webobject 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 webobjects 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:

```xml
<ie:task url="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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Telephone</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burton, Jack</td>
<td>Sales</td>
<td>873-2302</td>
<td>Sales Representative</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>Sales</td>
<td>873-2200</td>
<td>Sales Manager</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>Sales</td>
<td>873-3313</td>
<td>Administrative Assistant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Telephone</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anderson, Pat</td>
<td>Development</td>
<td>873-2428</td>
<td>Engineer</td>
</tr>
<tr>
<td>Stein, Chris</td>
<td>Development</td>
<td>873-2608</td>
<td>Manager</td>
</tr>
<tr>
<td>Worg, May</td>
<td>Development</td>
<td>873-2741</td>
<td>Engineer</td>
</tr>
</tbody>
</table>

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 were to choose to 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 our example "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 "*" will cause all groups in the task's VDB to be returned.

To execute the "listseparate" task outside of the JVM where the JSP engine resides, you 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 Data Groups to 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.

### Retrieving Data Groups from Adapters

When you are working with data that is coming from an information system, you 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 will substitute values from the current context groups and modify 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.

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.

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.

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. Executing authentication tasks is discussed later in the Advanced User Topics chapter of this guide.
Managing Large Amounts of Data

Large amounts of data can come in two different forms:

- Groups that have many elements
- Attributes within a given element that contain BLOBs.

The following sections provide some general information about how to manage large groups and BLOBs.

Managing Large Groups

When the resulting group that you want to display through a JSP page or through a custom application is larger than the display area, you can manage how much information is displayed in two of the display webjects: Display-Object and Display-Table. Using these display webjects, you can specify the starting element and the maximum number of elements to display. For example, if the task webject similar to the one shown previously in the Basic Group Creation and Display scenario created a group of 60 elements and your display area can only hold only 32 elements, you could modify the Display-Table webject in the scenario so that it displays only the first 32 elements by including the MAX parameter as follows:

```xml
<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:param name="MAX" data="32"/>
</ie:webject>
```

To display the final 28 elements, you would need to call the Display-Table webject again. This time you would include the START parameter as follows:

```xml
<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:param name="MAX" data="32"/>
  <ie:param name="START" data="33"/>
</ie:webject>
```

To generalize the use of the MAX and START parameters, you could for example, include some code that checks to see how many elements are in a group and then increment the value of the START parameter each time a user wants to see more data until all data had been displayed. Info*Engine provides some additional methods that can be helpful in this area. After you are familiar with the basic JSP and task concepts covered in the next couple of chapters of this guide,
you can go to the Using the Server Access Kit and Bookstore Example sections for information that could be useful.

Managing BLOBs

BLOBs are Binary Large Objects, such as picture files, sound files, or Word documents. Most Info*Engine adapters provide webjects that store BLOBs in information systems and retrieve BLOBs from the systems.

To setup the transfer of BLOBs to your information system, you can generate HTML INPUT elements that have their type set to FILE. This causes the Web browser to transfer the specified file to the Web browser and then on to Info*Engine. Info*Engine then streams the file to the specified adapter, which in turn, stores the BLOB in the information system.

When retrieving BLOBs from an information system, the adapter gets each BLOB from the system and then streams the BLOB data back to Info*Engine. Info*Engine then streams the data to the Web browser for display based on the MIME type specified.

After you are familiar with the basic JSP and task concepts covered in the next couple of chapters of this guide, you can go to Uploading and Downloading BLOBs for the details about managing BLOBs.
This chapter introduces the concept of an Info*Engine JSP page and describes how to use Info*Engine custom tags.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoring Info*Engine JavaServer Pages</td>
<td>3-2</td>
</tr>
<tr>
<td>Executing Multiple Webobjects at the Same Time</td>
<td>3-18</td>
</tr>
<tr>
<td>Accounting for Webobject Success or Failure</td>
<td>3-19</td>
</tr>
<tr>
<td>Catching Exceptions</td>
<td>3-21</td>
</tr>
<tr>
<td>Executing Tasks From JSP Pages</td>
<td>3-23</td>
</tr>
</tbody>
</table>
Authoring Info*Engine JavaServer Pages

JavaServer Pages (JSP) is a core technology of J2EE (the Java 2 Platform, Enterprise Edition) 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 webobjects 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:

```jsp
<%@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>
```

Storing Info*Engine JSP Pages

When Info*Engine is installed, the installer specifies an Info*Engine installation directory which determines where JSP pages must be stored. All Info*Engine JSP pages must reside under the codebase directory where Info*Engine is installed. For example, if the installation directory is "C:\ptc\Windchill", and the installation creates the "infoengine\jsp\examples" additional subdirectories for categorizing JSP pages, then the DisplayTable.jsp page can be found in the "\codebase\infoengine\jsp\examples" subdirectory. The complete DisplayTable.jsp file path is:

C:\ptc\Windchill\codebase\infoengine\jsp\examples\DisplayTable.jsp"
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:

- Including the host name and port (if the port isn't the default), and the application URL prefix specified when Info*Engine was installed. The default application URL is "Windchill".
- Naming the path for the JSP page that is relative to the codebase directory.
- Specifying any optional values to pass to the page.

Therefore to execute the "infoengine\jsp\examples\DisplayTable.jsp" page using the "myServer" host name and the "Windchill" application URL, specify the following URL:

```
http://myServer/Windchill/infoengine/jsp/examples/DisplayTable.jsp
```

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

### I*E Display-Table JSP Using Taglibs

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>myGroup2</td>
<td><a href="mailto:xxx@xxx.com">xxx@xxx.com</a></td>
<td>PTC</td>
</tr>
</tbody>
</table>

Usually, a Web server supporting JSP is configured such 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 the 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, 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 re-uses the previously generated servlet.

Info*Engine can also be configured so that you can direct JSP pages through the Info*Engine servlet. This involves specifying a different URL and is discussed later in this guide. For more information, see the section titled Executing Tasks.
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 will not be 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 Scriptlets, Expressions, Declarations, Directives, and Info*Engine Tags sections of the Info*Engine Custom Tag Reference chapter.
The previous example DisplayTable.jsp page includes the following standard JSP directives:

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

Is the standard page directive. It contains a reference to the example error page "IEError.jsp".

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

Identifies the "core" Info*Engine tag library, which provides the general custom tags that can be used in a JSP page, and identifies the "ie" prefix as the required prefix. The prefix identifies the tag as a tag belonging to the core tag library.

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. It assumes that you have acquired the knowledge you need from the books and Web sites that are available. To review the supported tags, you can access the following sites:

- [http://www.javasoft.com](http://www.javasoft.com)
- [http://www.w3c.org](http://www.w3c.org)
Using Info*Engine Custom Tags

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 the actions.

Available Tag Libraries

Info*Engine provides you with two 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.
- The directory library contains Info*Engine tags that can be used when you are manipulating information that is in a directory.

As described earlier, before using these tags on a JSP page, you must identify the library and it's 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:

```html
<%@ 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:

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

You can use any prefix that uniquely identifies the tags on the page.

Common Uses for Custom Tags

The following table gives some of the common uses for Info*Engine custom tags:

<table>
<thead>
<tr>
<th>Action</th>
<th>Custom Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define webjects and their parameters</td>
<td>webject</td>
</tr>
<tr>
<td></td>
<td>param</td>
</tr>
<tr>
<td>Execute Info*Engine tasks and define their parameters</td>
<td>task</td>
</tr>
<tr>
<td></td>
<td>param</td>
</tr>
<tr>
<td>Authenticate users who access the JSP page</td>
<td>authenticate</td>
</tr>
<tr>
<td>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</td>
<td>unit</td>
</tr>
<tr>
<td>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.</td>
<td>init</td>
</tr>
<tr>
<td></td>
<td>success</td>
</tr>
<tr>
<td></td>
<td>failure</td>
</tr>
</tbody>
</table>
The sections that follow show 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 chapter later in this guide.

For the general rules that apply to both JSP pages and Info*Engine tasks, also refer to the Info*Engine Custom Tag Reference chapter.

**Using Adapter Webjects**

The adapter webjects you can use are those action and query webjects that 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 choose to use simpler service names. For example, if your site knows that there will only be two JDBC adapters installed at your site, they could choose to name them "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 Info*Engine administrator.

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

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

You should use Info*Engine syntax:

```html
<ie:param name="INSTANCE" data="<%=instance%>"/>
```
Specify the INSTANCE parameter on each adapter webject. For example, it is specified on the following Query-Objects webject, which is part of the infoengine/jsp/examples/TableList.jsp page. This webject queries a database table using the JDBC adapter:

```jsp
<%@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>
<!-- Example url
<!-- http://localhost/Windchill/infoengine/jsp/examples/TableList.jsp?instance=myadapter&table=DEPT&where=DEPTNO=10 -->
<%
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"/>
</p>
</body>
</html>
```
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"

Using this information, an example URL is as follows:

```
http://localhost/Windchill/infoengine/jsp/examples/
TableList.jsp?instance=jdbcAdapter1&table=EMP&
where=DEPTNO=10
```

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.

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
<td>7839</td>
<td>1981-06-09 00:00:00.0</td>
<td>2450</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
<td></td>
<td>1981-11-17 00:00:00.0</td>
<td>5000</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>CLERK</td>
<td>7782</td>
<td>1982-01-23 00:00:00.0</td>
<td>1300</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

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:

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

  String table = request.getParameter ("table");
  if ( table == null )
    table = "EMP";
```
String instancel = request.getParameter("instance1");
if ( instancel == null )
    instancel = "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/jsp/examples/TableList.jsp?
    instance1=jdbcAdapter1&instance2=jdbcAdapter2&
    table=EMP&where=DEPTNO=10

If the 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 webobject 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 webobject example as follows:

```xml
<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 webobjects, see the adapter guides for the adapters you are using.
Using Task Webjects

The Info*Engine task webjects give you a quick way to generate and manipulate information, and to manage your JSP pages. Info*Engine maintains the information that is generated in groups within its VDB, as discussed in the How Info*Engine Manages Data chapter. 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:

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

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 provided at the beginning of this chapter contains the following group webject:

```html
<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):

<table>
<thead>
<tr>
<th>Attribute Name:</th>
<th>name</th>
<th>email</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>myGroup2</td>
<td><a href="mailto:xxx@xxx.com">xxx@xxx.com</a></td>
<td>PTC</td>
</tr>
</tbody>
</table>

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.

You can find additional example JSP pages that use task webjects under the "codebase/infoengine/jsp/examples" directory where Info*Engine is installed. Also, the Task Webject Reference chapter of this guide describes all group, management, message and Web Event Service webjects and describes many of the examples. You will also see some task webjects used in the examples shown throughout the remaining sections of this chapter.

The action and query webjects available to you are determined by which adapters have been installed at your site. For more information about adapter webjects, see the following Using Adapter Webject section.

**Using Display Webjects**

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. You declare that a webject is a display webject by specifying its type as "DSP" in the type attribute on the webject tag. For example, the DisplayTable.jsp page described earlier in this chapter contains the following webject tag:

```html
<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 webobjects provide the ability to display generated data in the following formats:

<table>
<thead>
<tr>
<th>Display Format</th>
<th>Webobject</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML tables</td>
<td>Display-Table</td>
</tr>
<tr>
<td>HTML form elements</td>
<td>Display-Selection</td>
</tr>
<tr>
<td>HTML elements such as a check box, radio button, or text box</td>
<td>Display-Value</td>
</tr>
<tr>
<td>Author-specified HTML coding</td>
<td>Display-Object</td>
</tr>
</tbody>
</table>

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

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

In addition to the general display webobjects listed in the previous table, there are display webobjects 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 at the beginning of this chapter shows one use of the Display-Table webobject. You can find additional example JSP pages that use display webobjects under the "codebase/infoengine/jsp/examples/dspwebobjects" directory where Info*Engine is installed. Also, the Display Webject Reference chapter of this guide describes all display webobjects and describes many of the examples. You will also see display webobjects used in the examples shown throughout the remaining sections in this chapter.
Specifying URIs and URLs

Info*Engine parameters can require that you specify URIs or URLs as their values. The following sections identify the formats you can use to specify URIs and URLs.

File Entries

Some custom tags require that you enter specific parameter values as valid URIs to files and some webobjects 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 webobject. 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, reside 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. Although Windows systems recognize either uppercase or lowercase letters as the same drive, Info*Engine does not treat two file references in which only the case of the drive letter is different as the same file reference.
- 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:

```
```

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 coma-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, such as some of the tags in the directory tag library, you may 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:

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

**Example File Locations**

Example URI locations of a "task1.xml" task file follow:

- 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:
  ```plaintext
  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:
  ```plaintext
  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:
  ```plaintext
  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 "ptcCommandSourceCode". 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:
  ```plaintext
  uri="ldap://srv3/cn=IEtasks,dc=myHost,dc=com?
  ptc.CommandDelegateSource"
  ```
Rule for Entering URIs and URLs

**Note:** The URIs and URLs 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 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.
Executing Multiple Webobjects at the Same Time

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

For example, the infoengine/jsp/examples/taglibs/tagParallel.jsp example page executes two queries at the same time. When both queries complete, the resulting groups are joined and then displayed.

```html
<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 chapter of this guide.
Accounting for Webject Success or Failure

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 taglibs must be processed. Ideally these tags should only be used from Info*Engine tasks.

For example, the following infoengine/jsp/examples/taglibs/tagUnit.jsp example 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.

```jsp
<%@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:unit>
<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:parallel>
```
<ie:unit>
  <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="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 chapter of this guide.
Catching Exceptions

There are two 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 the 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, the IEEError.jsp page that is located in the infoengine/jsp/examples directory where Info*Engine is installed contains the following page directive:

```jsp
<%@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":

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

On all of the Info*Engine example JSP pages provided with Info*Engine, you will see the IEEError.jsp specified. The IEEError.jsp page checks for many of the common Info*Engine exceptions and, based on what type of exception it is, it attempts to provide a short description or label for the error. For example, if the error "IEFatalServiceUnavailableException" occurs, then the page generates the message "Service Unavailable". Then the page prints out any message in the top level of the exception, and then prints the stack trace. If the exception happens to be "IEPartialResultsException", it will also display the partial results using the Display-Table webject.

The error page provided is just an example of how you could write a general error handling page. You can customize it or create your own.
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 last section the failure block provided was for all failures. In addition to having a general failure block like that one, 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.
Executing Tasks From JSP Pages

Although you can include any webject in a JSP page, you may 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.

The next chapter describes how to create and use tasks.
This chapter introduces the concept of an Info*Engine task and describes how to create and use tasks. The chapter also describes the XML output that Info*Engine produces when displaying groups.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info*Engine Tasks</td>
<td>4-2</td>
</tr>
<tr>
<td>Authoring Info*Engine Tasks</td>
<td>4-3</td>
</tr>
<tr>
<td>Basic Rules for Constructing Info*Engine Tasks</td>
<td>4-10</td>
</tr>
<tr>
<td>XML Output for Info*Engine Groups</td>
<td>4-12</td>
</tr>
<tr>
<td>Nesting Tasks</td>
<td>4-16</td>
</tr>
</tbody>
</table>
Info*Engine Tasks

Info*Engine text-based documents called standalone tasks 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 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 above whenever it can determine that a cached class is up to date. A cached class will be discarded and regenerated whenever the compiler discovers that the task has been updated since the last time that the task was compiled.
Authoring Info*Engine Tasks

All webjects that do not directly deal with the display of information can be placed in Info*Engine tasks. That is, any webject with a type equal to "DSP" exists only in a JSP page (or custom application). The remaining standard webject types can be included in Info*Engine tasks and are known as task webjects. These types are as follows:

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

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

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 people’s names, street addresses, and e-mail 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 can be shown as the following table:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O’Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>

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

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

Is the standard page directive. No additional classes are required for this task.

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

Identifies the "core" Info*Engine tag library, which provides the general custom tags that can be used in the task and identifies the "ie" prefix as the required prefix that identifies the tag as a tag belonging to the core tag library.
The previous CreateGroup.xml example task also includes the webject and param custom 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 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 the Info*Engine Tags section.

In addition to the directives and custom tags discussed previously, 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 the Scriptlets, Expressions, and Declarations sections.

Creating Tasks

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

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.

When Info*Engine is installed, the installer specifies an Info*Engine installation directory which determines where you should store your tasks. All Info*Engine tasks should reside under the tasks directory where Info*Engine is installed. For example, if the installation directory is "C:\ptc\Windchill", and the installation creates the "infoengine\examples" additional subdirectories for categorizing tasks, then the CreateGroup.xml file can be found in the "tasks\infoengine\examples" subdirectory. The complete CreateGroup.xml file path is "C:\ptc\Windchill\tasks\infoengine\examples\CreateGroup.xml".

Using the Info*Engine structure that is set up during installation, you should save your tasks under the "tasks/infoengine" subdirectory where Info*Engine is installed. Or, if you want to organize your tasks into categories, you can create other subdirectories similar to the "examples" subdirectory under the "tasks/infoengine" subdirectory and put your tasks in those subdirectories.
Executing Tasks

You can execute a standalone task in the following ways:

- Name the task on the Info*Engine custom task tag.
  
  You can include the task tag in JSP pages and in other tasks.

- Enter the URL to the task in a Web browser.

- Submit the task so that it is queued for execution.

- Emit an event that results in a task being executed.

Note: The URLs and URIs shown in this guide use the forward slash as the separator (/) in file 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:

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

```jsp
<%@page language="java" session="false"
       errorPage="IEError.jsp"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core"
       prefix="ie" %>
<ie:task uri="infoengine/examples/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 in this chapter. 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>

### 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. You produce the URL that contains the request to execute Info*Engine JSP pages by:

- Including the host name and application URL prefix specified when Info*Engine 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 values to pass to the task.

Therefore to execute the "infoengine/examples/CreateGroup.xml" task using the "myServer" host name and the "Windchill" application URL, specify the following URL:

```
http://myServer/Windchill/servlet/IE/tasks/infoengine/examples/CreateGroup.xml
```
The XML output displayed in the browser is similar to the following:

```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’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 options.

Your site administrator should know if your environment includes any of these optional features. For implementation information, see the Info*Engine Installation and Configuration Guide. For webject information, see the Queue-Task and Subscribe-Event webject reference sections.
**Basic Rules for Constructing Info*Engine Tasks**

The following sections show an example task and describe some important rules.

**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:

  ```jsp
  <%@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 which custom tag libraries are used in the file. Info*Engine has two custom libraries; the core and directory libraries. To identify the libraries, include `taglib` directives similar to the following:

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

  **Note:** You must specify the uri attribute values exactly as they are 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 compiler to skip a section of the task. If a webject is surrounded by a comment, it will not be 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 Scriptlets, Expressions, Declarations, Directives, and Info*Engine Tags sections of the Info*Engine Custom Tag Reference chapter.
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 which are coded as subelements 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 in the Group Webject Reference sections later in this document.

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 which 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 the Info*Engine, is 1.0. In addition, the Info*Engine support UTF8 for the character encoding. All XML generated by the Info*Engine therefore begins with the following XML declaration:

```xml
<?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. The 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.

```xml
<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
<?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 clearly harder to understand what data the unnamed group represents.

```xml
<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 subelement 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:

```xml
<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>
```

By viewing the example, you see the subelement 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 Info*Engine (such as Windchill) can generate metadata. For example, the 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 webobjects can be included within a task, there can be times when repetitive tasks are performed. Candidates for task nesting include any general task that can be used under multiple conditions. By using dynamic substitution (which is described in the Advanced User Topics chapter of this guide), you can easily construct tasks that can be used multiple times.

There are two ways to nest tasks. You can nest tasks by:

- Using the Include-Task webobject.
- Specifying the Info*Engine custom task tag within a task.

Include-Task Webobject Syntax and Rules

The syntax for the Include-Task webobject is:

```xml
<ie:webject name="Include-Task" src="path/someXMLFile"/>
```

where `path/someXMLFile` is the path of the task that is relative to the root directory of the task processor. This root directory was defined when Info*Engine was installed and configured.

Keep the following things in mind when working with nested tasks within XML tasks:

- This webobject can only be used in tasks. You cannot use it in JSP pages.
- Only one task can be named in the SRC attribute. However, the task named can nest another task by including the Include-Task webobject in its XML.
- The task file being referenced can contain other webobjects as necessary.

When searching for nested tasks that are specified with no file extension, Info*Engine first searches for the file without an extension and, if the file does not exist, Info*Engine appends the .xml extension to the file name and searches for the XML file. For example, assume that the task to be nested within another task is specified using `SRC="foo/bar"` and the value of the task root directory is `/usr2/tasks`. Then, the following order will be used to determine which file is used for the nested task:


The task processor stops at the first file it finds and uses that file for the nested task.
task Tag Rules for Nesting Tasks

Keep the following things in mind when working with Info*Engine task tag:

- 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 the task Tag section.

A Simple Nested Task Example

The following task includes a task tag that executes another task named "QueryTask.xml" that resides in the tasks "infoengine/examples" subdirectory:

```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>
```
If you assume that the QueryTask.xml file contains the following:

```xml
<%@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 the section entitled Dynamic Parameter Value Substitution. This section is in the Advanced User Topics chapter.
This chapter describes how Info*Engine provides advanced users with a different method of accessing the Info*Engine application programming interface (API) directly. To understand how direct access to the Info*Engine works, you must understand the basics of Java classes and applications.

The chapter also describes how to set up e-mail requests to process tasks, how to use SOAP requests, and how to use the Info*Engine Task Editor.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Server Access Kit</td>
<td>5-2</td>
</tr>
<tr>
<td>Using the Info*Engine E-Mail Broker</td>
<td>5-9</td>
</tr>
<tr>
<td>Using SOAP Requests</td>
<td>5-11</td>
</tr>
<tr>
<td>Info*Engine Task Editor</td>
<td>5-23</td>
</tr>
</tbody>
</table>
Using 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:

```java
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 should probably point to the ie.properties file. You will need to replace this Naming Service name, the distinguished name, and the LDAP host name with ones that apply to your network and LDAP directory structure. Normally, the LDAP directory structure and the Naming Service definition are 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:

```java
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 the section titled Specifying URIs and URLs.

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:

```java
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 multi-valued 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 user name under which the task should execute. This user name can then be used by Info*Engine as a credentials mapping key to select the credentials (for example, user name and password) that will be provided to each adapter with which the task will communicate. The task’s user name can be set as follows:

```java
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:

```java
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:

```java
    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:

```java
    task.invoke ();
```

### Catching Errors

If any error conditions are detected while executing the task, Java exceptions will be 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 will be 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 `Throwables` 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

Control is returned to the application 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
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:

```java
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 will add the group to the task's VDB before calling the `invoke` method. For example:

```java
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 are free to add their own metadata items, but they should avoid using names that begin with com.infoengine because this namespace is reserved by Info*Engine itself. 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.

Some of the common methods on these classes are listed in the following sections.

Common Group Methods

getAttributeValue
Retrieves the value of a specific attribute (Att) of a specific Element of the Group.

addElement
Adds a new Element to the Group.

ggetElementCount
Returns the number of Element objects contained in the Group.

getElements
Returns all of the Element objects of the Group as an enumeration.

ggetElementAt
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.
The following example method traverses a Group object to display all of its Elements and their attributes:

```java
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 ("\n");
            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 ("  ");
                    }
                    System.out.println (value);
                }
            }
            System.out.println ("\n");
        }
    }
}
```
Using the Info*Engine E-Mail Broker

The E-Mail Broker is a process that monitors a mailbox for requests to execute Info*Engine tasks. When a request arrives in the mailbox, the E-Mail Broker connects to the Info*Engine server and passes the request for execution. It captures the output of the executed template or task and returns it in an e-mail message to the address specified in the From or Reply-To heading of the original request. Thus, the E-Mail Broker allows users to make Info*Engine requests by e-mail.

The E-Mail Broker expects messages to have a specific format. When a message arrives in its mailbox, it opens the message and looks for a plain text body or plain text attachment containing a line with the following format:

```
URL=http://somehost/some-ie-uri
```

The E-Mail Broker processes the first such text body or attachment found, and it ignores all other bodies and attachments. In addition to specifying the URL of the Info*Engine template or task to be executed, the message can also specify authentication information and parameters to be passed to the template or task. This information is specified as `keyword=value` pairs, each of which must be specified on a line by itself. The keywords and values recognized by the E-Mail Broker are:

- **URL=absolute-url**
  - Specifies the absolute URL of the template or task to be executed.

- **USER=username**
  - Specifies the user name to be used in authenticating to the Web server through which the Info*Engine server is accessed. Normally, this user name also identifies the name to which e-mail messages for the E-Mail Broker should be addressed.

- **AUTH=credentials**
  - Specifies the credentials (usually a password) associated with the user name specified by the USER= line.

- **REPLYTO=email_address**
  - Specifies the e-mail address to use to reply to this request. When this line is included, the E-Mail Broker ignores the original cc and From mail headers.

- **PARAM='name=value'**
  - Specifies the name and value of a parameter to be passed to the template or task. Note that the name/value pair must be enclosed in apostrophes. If a
template or task accepts multiple parameters, specify a separate PARAM= line for each one.

The following example shows a request that would be sent as the body of an e-mail message. The request specifies the URL of an Info*Engine task. It also specifies a user name and password to use in authenticating to the Web server through which the Info*Engine server is accessed, and it provides to parameters to the task.

```
url= http://iehost.acme.com/Windchill/servlet/IE/tasks/
executeQuery.xml
user= ieguest
auth= ieguestpw
param= 'arg1=value1'
param= 'arg2=value2'
```

Assuming that the e-mail box specified is the default inbox, users can send Info*Engine requests to the E-Mail Broker by addressing e-mail messages to the name specified by the `emailBroker.username` property at the host specified by the `emailBroker.emailHost` property. For example, if `emailBroker.username` is defined as `ie` and `emailBroker.emailHost` is specified as `infoengine.acme.com`, then users would send Info*Engine requests to the following e-mail address:

```
ie@infoengine.acme.com
```
Using SOAP Requests

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, access the Web information at the following address:

http://www.w3.org/TR/SOAP

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. The servlet allows SOAP clients to invoke these specially written Info*Engine business tasks. For additional information about the required SOAP task format, see Writing Tasks for Use with SOAP.

How to implement the Info*Engine SOAP RPC servlet is described in the Info*Engine Installation and Configuration Guide. After the SOAP RPC servlet is implemented, you can construct a SOAP request that the servlet, in conjunction with the Web server, can process.

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 J2EE connector, the Microsoft SOAP SDK, or some other third-party SOAP client. The SOAP RPC servlet automatically generates the Web Service Definition Language (WSDL) that is used by the client to determine the methods that are supported and to identify the parameters each method requires. WSDL, defined as a standard by W3C, is an XML-based language for formally defining Web service interfaces.

The Info*Engine J2EE connector uses WSDL to generate client side source code to simplify interactions with Info*Engine. For additional information, see About the Info*Engine J2EE Connector.

The Microsoft SOAP SDK 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 additional information, and to download the Microsoft SOAP SDK, use the following URL:

http://www.microsoft.com/downloads

Other third-party clients may use WSDL to generate source code, or to drive interactions with Info*Engine.
The following diagram shows 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 Aphelion Directory 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:

- a primitive value, for example int, float, date, etc.
- a Java bean
- an array of primitive values
- an array of Java beans
- an Info*Engine group.
If an Info*Engine group is returned, the group is returned to the SOAP client as a single string that contains the Info*Engine XML representation of that group.

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 J2EE connector clients receive SOAP faults as Java exceptions; Visual Basic clients can make use of instance variables such as faultcode, faultstring, faultactor, and detail.

Sample SOAP Client

Info*Engine provides the ping.vbs sample Visual Basic script. This script verifies that you can execute a SOAP RPC servlet request, thus providing you with a basic SOAP client. When the script runs, it produces a SOAP request that executes an Info*Engine task file. This sample is located in the prog_examples/SOAP_apps directory where Info*Engine is installed and can be used for demo and testing purposes. You can run this SOAP client to determine if your environment is set up correctly. You can also study the code in this script to learn more about how to generate Info*Engine SOAP requests in your environment. The output from the sample SOAP client is XML output from the Info*Engine task. For full implementation, you can provide your own SOAP client, which can be a Java or non-Java application.

For the details on testing the sample SOAP client, see the Info*Engine Installation and Configuration Guide.

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 application URL 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:

  urn: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 will attempt 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 will attempt 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 Aphelion Directory that is used by Info*Engine. For additional information on repositories, repository types, type identifiers and task delegates, see the Delegate Administrator section of the *Info*Engine Installation and Configuration Guide.

For details on how the request is processed by your Web server and Info*Engine, see the *Info*Engine Installation and Configuration Guide.

**Writing Tasks for Use with SOAP**

For an Info*Engine task to be invoked through a SOAP RPC 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 Administrator, or using supplied tools.

- The task must include the RPC markup comment lines.

- The task must include special SOAP comments lines within the RPC markup comment if the task requires input parameters or returns a response other than an Info*Engine group. See the SOAP Comments section below for further information.
**Supported Data Types**

The Info*Engine SOAP RPC servlet supports the following data types, which include both primitive types and java 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

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. Arrays of any type are also accepted as both input and output.

**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 SDK, 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 J2EE connector, the java bean class must also be available in the client CLASSPATH. For additional information on the Info*Engine J2EE connector, see About the Info*Engine J2EE Connector.
**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 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.

**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.

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.

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 discussed below.

**@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 is no white space 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.
@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. When returning a basic data type, or array of a basic data type, 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 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 must not be 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.

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 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 would occur.

Simple Data Type Examples

The following example specifies two integers as input parameters, and returns their sum.

```java
<%@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.

```java
<%@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].doubleValue();

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

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]}

-->
The following example searches for a list of users using an LDAP search filter.

```java
<%@page language="java"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<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[]}" default="uid=*"/>
    <ie:param name="GROUP_OUT" data="output"/>
</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.

```java
<%@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="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>
```
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:weobject 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:weobject>
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.

Installing Info*Engine Task Editor

Task Editor is installed with Info*Engine, but the following steps are necessary to load the application before it can be used.

1. Locate the \ptc\windchill\taskeditor\ where Info*Engine is located on your machine.
2. Double-click Install.exe (for Windows), or Install.bin (for Unix), to begin the installation.
3. Quit any programs that are running on your machine before continuing.
4. Follow the onscreen instructions.
5. During installation, click Next to proceed to the next screen, or Previous to return to a previous screen.
6. Note the location of the Info*Engine Task Editor installation. You will need it to start the Task Editor.
7. Click Done when the installation is completed.

Starting Info*Engine Task Editor

1. Run each of the following in turn:
   - For Apache:
     cd <ie_dir>/Apache/bin/Apache.exe
   - For Tomcat:
     cd <ie_dir>/Tomcat/bin/wttomcat_start.bat
   - For Winchill:
     cd <ie_dir>/Windchill/bin/infoengine/starnamingservice.bat
     where <ie_dir> is the location of the Info*Engine installation.
2. Once all of the above are running, minimize their windows.
3. Start the Info*Engine Task Editor. If you accepted the defaults during installation, you can start Task Editor from the Start menu. If you are running
on Windows, this will initially open a DOS window, and then the Setup dialog box.

The Setup dialog box is used to initialize the Info*Engine Task Editor. The information in this dialog box defaults to what you entered during installation, and allows the Task Editor to determine your configuration information and list available services and adapters.

**Provider URI** - The provider that supplies your configuration information. For example, ldap://myHost.myCompany.com/.

**Note:** This is a mandatory field. If it is not filled, you will not be able to proceed.

**Principal** - The principal used to bind to the directory provider. For example, cn=Manager.

**Password** - The principal entry's password.

**Service Search Base** - The location in the directory where Info*Engine services and adapters can be found. For example, dc=myHost,dc=myCompany,dc=com.

4. Make any necessary changes and click **OK** to accept the entries.

Once this information is collected, the Info*Engine Task Editor will attempt to look up service information.

If an Info*Engine Server and Naming Service are available, they will be used to remotely invoke tasks on available task processors.

If available task processors have installed Info*Engine Task Editor IO tasks, it will be possible to use them to remotely administer tasks and other files found on the task processor's local file system, or in its LDAP directory.

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

**Stopping Info*Engine Task Editor**

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

Panels

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

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 to run the task or JSP.

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

The services panel contains 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.

Menus

File

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 - Use the Open dialog box to open an Info*Engine task or JSP from the local file system. The file appears in the editor panel. (Keyboard shortcut, press CTRL+O.)

Note: Your Open option may include additional sub menus, one for each Task Processor supporting Info*Engine Task Editor IO that is available to your Info*Engine configuration. These sub menus will contain one menu item for each type of IO the Task Processor supports. Use these sub menus to remotely edit Info*Engine task or JSP source code.

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.
Note: Your Save As option may include additional sub menus, one for each Task Processor supporting Info*Engine Task Editor IO that is available to your Info*Engine configuration. These sub menus will contain one menu item for each type of IO the Task Processor supports. Use these sub menus to remotely save Info*Engine task and JSP source.

Exit - Exit the Info*Engine Task Editor.

Edit

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.)

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... - Use the Find/Replace dialog box to locate text within the active Info*Engine task or JSP in the editor panel. (Keyboard shortcut, press CTRL+F.)

Goto... - Use the Go To Line dialog box 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 will move the cursor to the corresponding open or close brace character. (Keyboard shortcut, press CTRL+\.)

If no corresponding open or close brace is found a beep will sound. If the cursor is not adjacent to a brace character this action will do nothing. The following are considered brace characters; (), [], {}, and <>.

Tools

Edit Tag - Open the Edit dialog box for the tag in which the cursor is placed in the editor panel.

Preferences... - Open the Preferences dialog box.

View

Toolbars - Show/hide the file, edit, find, or edit toolbars.

Services/Webjects - Show/hide the services and webjects panel.
**Help**

**Contents...** - Open the Help dialog box. (Keyboard shortcut, press F1.)

**About...** - Open the About dialog box.

### Toolbar Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="forEach" /></td>
<td>Drop forEach Block</td>
<td>Click this icon and then in the editor panel to insert a forEach block into your Info*Engine task or JSP source. When dropping a forEach block into your source, you will be presented with a dialog box that will allow you to enter values for the groupIn and groupOut attributes.</td>
</tr>
<tr>
<td><img src="image2" alt="parallel" /></td>
<td>Drop parallel Block</td>
<td>Click this icon and then in the editor panel to insert a parallel block into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image3" alt="task" /></td>
<td>Drop task Tag</td>
<td>Click this icon and then in the editor panel to insert a task tag into your Info*Engine task or JSP source. When dropping a task into your source, you will be presented with a dialog box that will allow you to enter values for the URI and processor attributes. You will also be able to add parameters to be sent along with the request to execute the task.</td>
</tr>
<tr>
<td><img src="image4" alt="declaration" /></td>
<td>Drop declaration Block</td>
<td>Click this icon and then in the editor panel to insert a declaration block into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image5" alt="scriptlet" /></td>
<td>Drop scriptlet Block</td>
<td>Click this icon and then in the editor panel to insert a scriptlet block into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image6" alt="expression" /></td>
<td>Drop JSP Expression Block</td>
<td>Click this icon and then in the editor panel to insert an expression block into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image7" alt="taglib" /></td>
<td>Drop JSP Taglib Directive</td>
<td>Click this icon and then in the editor panel to insert a taglib directive into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image8" alt="page" /></td>
<td>Drop JSP Page Directive</td>
<td>Click this icon and then in the editor panel to insert a page directive into your Info*Engine task or JSP source.</td>
</tr>
<tr>
<td><img src="image9" alt="edit" /></td>
<td>Edit</td>
<td>Clicking the <strong>Edit</strong> icon with the cursor in a tag will display the associated wizard for you to make the desired changes. When finished, click <strong>OK</strong> to apply the changes. Wizards are available for several different types of tags, such as; webject, task, param, forEach, JSP page directive, and JSP taglib directive.</td>
</tr>
</tbody>
</table>
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 Task Editor.

1. Select **Open** and then **Open file** from the **File** menu. This will the Open dialog box where you can browse for task and JSP files.

2. Select the file you want to open and click **Open**. The selected file will appear in the editor.

Creating an Info*Engine Task or JSP

With Info*Engine Task Editor, you can create tasks and JSPs that meet the same criteria as explained in Chapter 3 with minimal code entry.
New Task

Select **New** and then **New task** from the **File** menu. This will open an editor for a new task. Scriptlets for page and taglib directives will already be added according to settings on the Preferences dialog box.

```xml
<%@page language="java"%>
<%@taglib url="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 then clicking the **Edit** icon.

For example placing the cursor in the taglib directive, as shown above, and clicking **Edit** will open the Taglib dialog box.
New JSP

Select New and then New JSP from the File menu. This will display the New JSP dialog box, with fields and dropdown menus for you to enter and select page and taglib directives.

![New JSP dialog box]

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

```xml
<%@ page language="java" %>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix='ie' %>
<html>
<body>
</body>
</html>
```
Entering and Editing an Info*Engine and JSP Tags

Webjets

When a service or adapter is selected in the top panel, the lower panel displays the webjets it supports (if any). The webjets 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.

A wizard specific to that webject will display the parameters supported by that webject. The parameters will be organized on two tabs: Required/Select and Optional.

You can edit parameter attributes other than name and data by double-clicking the parameter name. Doing so will display a parameter wizard where you can edit other parameter attributes like default, delim, elementSeparator and valueSeparator. (The same is true of double-clicking within the Name text field when using the task wizard).
Fill in the desired parameter values and click **OK** to insert the webject into your Info*Engine task or JSP source.

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

**Tag Insight**

The text editor may make suggestions relative to the text you are typing. If you are typing the start of a tag that it recognizes, it will display a dropdown menu of possible tags from which you can select.

```xml
<%@page language="java"
%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"
%>
<body>
<ie:
<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; Info*Engine core tags, Info*Engine directory tags, and JSP tags. You can create your own Tag Insight on the **Tag Insight** tab of the **Preferences** dialog box.
Edit Wizard

Click anywhere within a tag in the editor panel, and then right-click. If the selected tag has a wizard associated with it, an Edit dialog box will appear.

Clicking the Edit icon with the cursor in a tag will also display the associated wizard for you to make the desired changes. When finished, click the OK button to apply the changes.

Wizards are available for several different types of tags, such as: webjct, task, param, forEach, JSP page directive, and JSP taglib directive.
This chapter describes some of the more advanced uses of Info*Engine beyond simple data retrieval and manipulation. Because these are advanced topics, using them can require specialized knowledge beyond the basics of HTML or XML required for simple template and task creation typical of basic Info*Engine solutions.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Parameter Value Substitution</td>
<td>6-2</td>
</tr>
<tr>
<td>Uploading and Downloading BLOBs</td>
<td>6-13</td>
</tr>
<tr>
<td>Authenticating Users</td>
<td>6-22</td>
</tr>
<tr>
<td>Custom Webjects</td>
<td>6-31</td>
</tr>
<tr>
<td>Using Double-Byte Characters</td>
<td>6-39</td>
</tr>
<tr>
<td>Packages</td>
<td>6-40</td>
</tr>
<tr>
<td>About the Info*Engine J2EE Connector</td>
<td>6-51</td>
</tr>
<tr>
<td>Standalone Java SOAP Clients</td>
<td>6-56</td>
</tr>
<tr>
<td>J2EE SOAP Clients</td>
<td>6-67</td>
</tr>
</tbody>
</table>
Dynamic Parameter Value Substitution

Although many simple Info*Engine installations will not require them, often webject parameter values must 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 which 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.

General Value Substitution Syntax

Info*Engine substitution expressions are always delimited by the characters "\${" and "}". Within these delimiters, you 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 (*), 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:

$$\{\text{grp1}[\text{name=}Doyle]\text{phone[]}\}$$

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. 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 contains the information produced when an internal error has occurred.
- **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:

$$\{\text{Grp123}[\text{META}]\text{MESSAGE[]}\}$$

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 (*), 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[]}
```

Where Substitution Can Be Used

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

Including Default Values When Doing Substitutions

To cover the possibility that the evaluation of a substitution expression could return no values, you should include a default for the expression. You 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="**"/>
```
When Values Come from a Web-Based Form or as URL Parameters

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:

\[\text{data}="${@FORM[]\text{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:

\[\text{data}="${@FORM[]\text{variable}[*]}" \text{delim}=","\]

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.

For additional information about the substitution expressions you can use, see the previous section titled General Value Substitution Syntax.

\textbf{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 additional information, see the section titled Uploading and Downloading BLOBs, later in this chapter.
Example: Display-Object Substitution Using @FORM

Assume that the following variables have been set through a Web-based form:

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

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

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:

```xml
<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:

```xml
<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>
```
When Values Come from a VDB Data Group

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[]]}\]

You use \(grp\_name[]\) in these formats when there is only one element (row) in the group named \(grp\_name\). You use \(att\_name[]\) in these formats when there is only one value in the attribute named \(att\_name\). 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[*]]} \text{ 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 additional information about the substitution expressions you can use, see the previous section entitled General Value Substitution Syntax.
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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ename</td>
<td>phone</td>
<td></td>
</tr>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td></td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td></td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td></td>
</tr>
</tbody>
</table>

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:

```xml
<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:

```xml
<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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TELEPHONE</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burton, Jack</td>
<td>873-2302</td>
<td>Sales</td>
</tr>
<tr>
<td>Law, Gracie</td>
<td>873-2200</td>
<td>Sales</td>
</tr>
<tr>
<td>LoPan, David</td>
<td>873-3313</td>
<td>Sales</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, A1</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>B, B1</td>
<td>77</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

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":

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>433</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A1</td>
</tr>
<tr>
<td>2</td>
<td>221</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B1</td>
</tr>
<tr>
<td>3</td>
<td>758</td>
<td>C</td>
</tr>
</tbody>
</table>

**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 provides examples for uploading and downloading BLOBs.

For additional information on the adapter webjects that upload and download BLOBs, see the adapter reference guide for the adapters your site is using.

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.

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 `UploadBlob.jsp` page contains explanatory text and a form for selecting a file to store in a column of an Oracle database row.

This page prompts for a "name" and "file" to store in an Oracle database. The "name" is used to select a row in the Oracle table. The value of "name" and the "file" contents are stored in columns in the selected row. The Oracle table must be created before this example is run. The Info*Engine task in `../tasks/infoengine/examples/BuildBlobDb.xml` can be used to create the table.

### Upload File to Oracle BLOB Column

**Adapter Instance:**

**Name:**

**File:** [Browse...]

**Submit**

On this form, a user:

1. Enters the name of the adapter and name under which the BLOB data will be stored in the "instance" and "filename" form variables.

2. Selects the file to upload.

3. Clicks the **Submit** button to stream the data from the form to the Web browser.
This page prompts for a "name" and "file" to store in an Oracle database. The "name" is used to select a row in the Oracle table. The value of "name" and the "file" contents are stored in columns in the selected row.

The Oracle table must be created before this example is run. The Info*Engine task in .../tasks/infoengine/examples/BuildBlobDb.xml can be used to create the table.

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 things 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 UploadBlob.xml example task that follows 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:

```html
<%@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 DownloadBlob.jsp page contains an HTML form for downloading a BLOB file from a database and displaying it through an application called by the Web browser. The page contains the following text and form:

This page prompts for a "name" of the BLOB file to retrieve from an Oracle database and a "MIME type" to associate with the BLOB file. The "name" is used to select a row in the Oracle table where the BLOB is stored. The Oracle table must be created and BLOBs uploaded to rows in the table before this example is run. The Info*Engine task in .../tasks/infoengine/examples/BuildBlobDb.xml can be used to create the table.

Download File from Oracle BLOB Column

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:

- 'text/plain'
- 'application/msword'
- 'application/msexcel'
- 'application/vnd.ms-excel'
- 'image/gif'

Include the single quotes in your MIME Type entry.

**Adapter Instance:**

**Name:**

**Mime Type:**

[Retrieve]

On this form, a user:

1. Enters the name of the adapter and name under which the BLOB data is stored in the "instance" and "filename" form variables.
2. Enters the MIME type of the BLOB data in the "mimetype" form variable so that the correct application displays the BLOB.
3. Clicks the **Retrieve** button to start the task that streams the BLOB data from the database to the Web browser.
The code for the page is as follows:

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

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

<html>
<head><title>Send Blob</title>
<BASE HREF="http://localhost/infoengine/servlet/IE/tasks/">
</head>
<body>

This page prompts for a "name" of the BLOB file to retrieve from an Oracle database and a "MIME type" to associate with the BLOB file. The "name" is used to select a row in the Oracle table where the BLOB is stored. The Oracle table must be created and BLOBs uploaded to rows in the table before this example is run. The Info*Engine task in .../tasks/infoengine/examples/BuildBlobDb.xml can be used to create the table.

Download File from Oracle BLOB Column

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:

- 'text/plain'
- 'application/msword'
- 'application/msexcel'
- 'application/vnd.ms-excel'
- 'image/gif'

Include the single quotes in your MIME Type entry.

<form method="POST" action="examples/DownloadBlob" enctype="multipart/form-data">
<table>
<tr><td align=right><B><FONT FACE=arial,helvetica>Adapter Instance:</FONT></B></td><td><input name = "instance" type="text" size=50></td></tr>
<tr><td align=right><B><FONT FACE=arial,helvetica>Name:</FONT></B></td><td><input name = "filename" type="text" size=50></td></tr>
<tr><td align=right><B><FONT FACE=arial,helvetica>Mime Type:</FONT></B></td><td><input name = "mimetype" type="text" size=50></td></tr>
```

6-20 Info*Engine User's Guide
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.

**BLOB Download Task**

The DownloadBlob.xml example task that follows 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.

The code for the example task is as follows:

```html
<%@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[0]}"
      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>
```
Authenticating Users

There are three basic types of user authentication:

- Web server authentication
- Servlet-based authentication
- Authentication through credentials mapping

The following sections describe the authentication methods.

Web Server Authentication

Your Info*Engine administrator can set up Web server authentication for Info*Engine URL requests when the Web server is configured. If this is done, users must enter a user name and password of an authenticated user to execute Info*Engine requests. How to setup Web server authentication is described in the Info*Engine Installation and Configuration Guide.

Servlet-based Authentication

There are two ways to use servlet-based authentication with JSP pages:

- Use the standard Info*Engine implementation of executing JSP pages directly through the JSP engine, but include an Info*Engine authenticate tag on each page.
- Implement a JSP Model II approach to executing JSP pages. In this approach, JSP pages are routed through the Info*Engine servlet. The servlet can then process the requests, which includes checking for authentication.

Using the authenticate Tag on JSP Pages

Basic authentication for JSP pages can be done by including the authenticate tag on each JSP page you want authenticated. For example, including the following tag prevents anonymous access:

```xml
<ie:authenticate/>
```

In this case, if the user has not logged on, Info*Engine returns an authentication challenge.

If your site has set defaults for an authentication task in the following servlet property:

```
.jsp.taglibs.authenticationTask
```

The previous authenticate tag will also execute the default authentication task. You can override the default task by specifying the task attribute. For example, the following tag executes the "authenticate.xml" task that is located in the "infoengine" directory where tasks are stored:

```xml
<ie:authenticate task="infoengine/authenticate.xml"/>
```
When this tag is used, the servlet checks the HTTP request for the presence of an Authorization header. If no such header is present, the servlet returns an authentication challenge requesting Basic authentication. If this header is present, and it specifies HTTP basic authentication, the servlet decodes the user name and password provided and adds them to the SERVER context group. The user name is added as an attribute named auth-user, and the password is added as an attribute named auth-password.

After successfully obtaining a user name and password from the HTTP request, the servlet connects to the Info*Engine server task processor and requests that it execute the authentication task.

A typical authentication task that makes use of an adapter Validate-User webject might look like:

```html
<%@page language="java" session="false"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie" %>
<!-- Authenticator --/
<ie:webject name="Validate-User" type="ACT">
  <ie:param name="INSTANCE" data="jdbcAdapter"/>
  <ie:param name="DBUSER" data="${@SERVER\[\]auth-user\[\]}"/>
  <ie:param name="PASSWD" data="${@SERVER\[\]auth-password\[\]}"/>
  <ie:param name="GROUP_OUT" data="VUstatus"/>
</ie:webject>

The servlet checks for either of two types of groups returned by the task, so the task can return either of:

- A status group. Normally, this is what a Validate-User webject returns. If the status group indicates failure, then the servlet returns an authentication challenge. Otherwise, it indicates success and allows the original request sent by the browser to proceed.

- A normal object group. If the object group specifies a failure status or it contains no elements, then the servlet interprets this as a failure indication and returns an authentication challenge. Otherwise, if the object group specifies success status and it contains at least one element, the servlet allows access.

In addition, the servlet checks for exceptions that the authentication task could throw. If an exception is thrown, the servlet returns another authentication challenge to the user.

**Note:** If you select OK from the Authentication Dialog box without providing anything for User Name and Password, Info*Engine picks up the default values for these from the Adapter configuration properties.

For additional authenticating options that can be used with this tag, see the authenticate Tag description.
Authenticating JSP Pages Through the Info*Engine Servlet Engine

If your site has implemented JSP Model II for executing JSP pages or you send requests to execute tasks through the Info*Engine servlet, your Info*Engine administrator can set up servlet-based authentication for the JSP pages and the tasks by doing the following things:

- Create a user validation task and store it in the specified directory. The task can be similar to the one described under the last section, Using the authenticate Tag on JSP Pages on JSP Pages.

- Configure the following Info*Engine servlet properties:
  - .authenticationTask
  - .authenticationRealm
  - .authenticationIdleTimeout

This feature is enabled by defining the Info*Engine servlet property named "authenticationTask". For example, the following property sets the authentication task to "authTask.xml", which is in the "infoengine" directory where tasks are stored:

```
/authenticationTask=infoengine/authTask.xml
```

If the .authenticationRealm property is defined, it specifies the realm name to be returned in authentication challenges. An example of usage is:

```
/authenticationRealm=ie.acme.com
```

The servlet also supports "session" idle timeouts. This is enabled through a configuration property named authenticationIdleTimeout. For example, the following property sets the timeout to "10":

```
/authenticationIdleTimeout=10
```

This property specifies the number of minutes that the servlet will wait for a request from a user before returning an authentication challenge. As long as a user continues to make requests at a steady pace (less than the timeout period apart) the servlet will not return a challenge. Thus, this property has the effect of logging the user off after a specified period of inactivity. Note that this feature operates independently of the servlet authentication feature, and it can be enabled independently of it, too. This allows the servlet to challenge users for authentication after periods of inactivity even when the servlet itself is not configured to perform authentication.

**Note:** If you select OK from the Authentication Dialog box without providing anything for User Name and Password, Info*Engine picks up the default values for these from the Adapter configuration properties.
**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 user name, Info*Engine obtains a map that defines the user names 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 a task or JSP page contains the Map-Credentials webject, the results from this webject override any credentials that come from the system-wide credentials mapping information that can be set.

If no credentials mapping information is available, or if the credentials mapping does not provide valid DBUSER and PASSWD values for the adapter to which a webject is being routed or for the JMS MOM, 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 as follows:

- 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 will check the webject to see if DBUSER and PASSWD parameters have been specified explicitly. 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 will add them to the webject as if they had been specified explicitly by the author of the task. Otherwise, the webject will be routed to the adapter unmodified.

For information on how to set up credentials mapping for adapter webjects, see the Info*Engine Installation and Configuration Guide.

The following sections describe how to set up a credentials mapping task and credentials mapping files.

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 will be available to it. For example, the SERVER context group will be available. If a request sent by a browser to Info*Engine has been authenticated by the Web server, then the SERVER group will contain an attribute named Auth-User. This attribute specifies the user name that was authenticated by the Web server. The credentials mapping task can use that user name 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 example credentials mapping task creates a group that contains all valid user name and INSTANCE name combinations with corresponding DBUSER and PASSWD values. From this group, it then selects only those elements that contain the authenticated user name. These elements then make up the Auth-Map group.

```java
<%@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[0]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:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>INSTANCE</th>
<th>DBUSER</th>
<th>PASSWD</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>com.myHost.jdbcScott</td>
<td>scott</td>
<td>tiger</td>
</tr>
<tr>
<td>abc</td>
<td>com.myHost.jdbcAdapter</td>
<td>abc</td>
<td>abc123</td>
</tr>
</tbody>
</table>

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, you 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 user name to be set in the DBUSER parameter of the webject.
- `passwd` specifies the password that corresponds with the user name. It is the value to be set in the PASSWD parameter of the webject.

The resulting file contains the user names and passwords that can be used to access information systems.

Your site administrator specifies the directory in which all of the mapping files must reside in the .credentialsFiles property. If this property is not set, credential 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 .credentialsFiles and the .credentialsMapper 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.
Credentials Mapping for MOMs

If your site has installed and configured a MOM, 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. Instead, Info*Engine provides the following as pseudo INSTANCE parameter values:

<table>
<thead>
<tr>
<th>Pseudo INSTANCE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.infoengine.msg</td>
<td>Value for message webjects (type=MSG)</td>
</tr>
<tr>
<td>com.infoengine.wes</td>
<td>Value for Web Event Service webjects (type=WES)</td>
</tr>
<tr>
<td>com.infoengine.jms</td>
<td>Value for either message or Web Event Service webjects</td>
</tr>
</tbody>
</table>

By setting up three pseudo INSTANCE parameter values, Info*Engine allows your site the flexibility of setting up three accesses to your MOM through credentials mapping:

- Use the com.infoengine.jms pseudo INSTANCE parameter value to set a user name 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 user names and passwords.

If a user name 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 JMS, 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 user name 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 user name. These elements then make up the Auth-Map group.

```jsp
<%@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:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>INSTANCE</th>
<th>DBUSER</th>
<th>PASSWD</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>com.myHost.jdbcScott</td>
<td>scott</td>
<td>tiger</td>
</tr>
<tr>
<td>abc</td>
<td>com.myHost.jdbcAdapter</td>
<td>abc</td>
<td>abc123</td>
</tr>
<tr>
<td>abc</td>
<td>com.infoengine.jms</td>
<td>mom1</td>
<td>gen456</td>
</tr>
</tbody>
</table>

Then, user "abc" will use the user name "mom1" and password "gen456" to gain access to the MOM whenever the user tries executing a message or Web Event Service webject.
Custom Webjects

Custom webjects 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 they 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE=EXT</td>
<td>Indicates that a user-defined class is to be used.</td>
</tr>
<tr>
<td>USE=ClassName</td>
<td>Identifies the user-defined class (including the package) in which the method implementing the webject resides.</td>
</tr>
</tbody>
</table>
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
- Throw IEException on error
- Return a com.infoengine.object.factory.Task object

For example, the signature for the method that implements the Average-Column custom webject is the following:

```java
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 ie.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 Example

The following sections include:

- Source code for the NumericColumnWebjects class.
- 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** which is a private method used by the public methods to get the elements in a column.
- **averageColumn** which averages the elements in a numeric column of data.
- **totalColumn** which computes the 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.GroupWebObjectException;
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 formatting of the results is performed.
 * 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.customwebjects.NumericColumnWebjects">
 *   <ie:param name="COLUMN" data="SAL"/>
 * </ie:webject>
 */
```
public class NumericColumnWebjects
{

    private static Vector getColumnElements ( Task task ) throws IEException {
        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
IEException {
    try {
        String grp_out = null;
        String column = null;

        Wobject w = task.getWobject();

        // 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 ( int j = 0; j < elements.size(); ++j )
            dval = elements.elementAt ( j ).doubleValue();

            average += dval;
        }

        return average / elements.size();
    } catch ( Exception e ) { throw new IEException ( e ); }
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 IEException {
    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 GroupWebobjectException
        ( "NumericColumnWebobjects: 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 webobjects.
Example Average-Column and Total-Column External Webjects

The following NumericColumnWebjects.jsp executes a Query-Objects webject that creates the "employees" group that is then used by the Average-Column and Total-Column external webjects:

```jsp
<%@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.customwebjects.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>
```
Using Double-Byte Characters

Info*Engine supports the use of double-byte characters. To use these characters on a JSP page, you must set the content type to use the appropriate character set designator in the page directive. For example, shift_JIS, for Japanese characters. For example, you can use the following page directive:

```jsp
<%@page language="java" 
    session="false" 
    errorPage="../IEError.jsp" 
    contentType="text/html; charset=Shift_JIS"%>
```

Info*Engine provides set of example JSP pages and tasks that show how code for double-byte characters. The example JSP pages are located in the codebase/infoengine/jsp/examples/ja directory where Info*Engine is installed. The example tasks are located in the tasks/infoengine/examples/ja directory where Info*Engine is installed.

Some examples, such as EchoFormMulti.jsp, can be run without modification. However, to use other examples, you must modify them for use in your environment. There are example JSP pages that create, delete and query JNDI objects. Using these examples, you can verify that double-byte data can be entered into an LDAP directory, be retrieved and displayed correctly in a web browser.

The example files that create and delete LDAP entries is one example that you must modify before you run it. The example contains double-byte characters, but will need to be updated with the valid double-byte characters from the current double-byte system. This example consists of the following files:

- JndiCreateEntry.jsp
- JndiCreateEntry1.jsp
- JndiCreateObject.xml
- JndiDeleteObject.xml

The JndiCreateEntry.jsp page contains a form in which there is a paragraph for a description. You can insert any set of multi-byte characters in the description to see that they display on the JSP form. Also, when you execute the JndiCreateEntry.jsp page, the form that is displayed has input fields where multi-byte characters can be entered. To actually create a new LDAP entry or delete one (as indicated by the form), you will need to have an LDAP directory that you can use and set up an JNDI adapter LDAP entry that accesses the directory. Then, you must modify the JndiCreateEntry1.jsp page and the JndiCreateObject.xml and JndiDeleteObject.xml tasks so that they contain the correct information to call the JNDI adapter.
Packages

A package is made up of a set of files (tasks or DCA configuration specifications) and a set of LDAP directory entries that are 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.

There are two types of packages used by Info*Engine:

- Task packages, made up of individual task files or task directories. Task package file names end with the extension .ptctar.
- DCA packages, made up of DCA configuration specifications, individual task files, or task directories. DCA package file names end with the extension .ptcdar.

Providing LDAP Directory Information

For tasks to be distributed as a part of a package, the LDAP directory information must be provided. This information, including delegate name and repository type, is needed for the creation of the task directory hierarchy when the package is installed. While this information can be provided using LDIF files, the LDIF file must be maintained and updated if tasks are redistributed for use in SOAP or DCA applications.

Note: DCA applications are only utilized when Info*Engine is installed along with a Windchill system.

With packages, this LDAP information can be provided by specifying properties in two ways: in properties files within the task directory structure, or within the task itself.

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.

typeld

 Specifies the type identifier under which delegates are created. If this property is not specified, then the installation tool will determine 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 file name (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.

Defining Properties Using 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
```

Defining Properties Within 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
```
Managing Packages

The Package Manager is a simple Java application which creates, installs, and uninstalls task and DCA packages. The main Java class for the Package Manager is com.infoengine.administration.packaging.UI. The Package Manager requires the following classpath to function properly:

${wt_home}/lib/servlet.jar;${wt_home}/codebase/WEB-INF/lib/ie.jar;${wt_home}/codebase/WEB-INF/lib/ie3rdpartylibs.jar

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 would be:

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 on the command line:

   java com.infoengine.administration.packaging.UI

   or

   java -DpropFile=${wt_home}/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 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 Info*Engine or Windchill virtual machine. This value must be the runtime service name of your Windchill or Info*Engine 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 good, the Package Manager opens.
The **Package Manager** window consists of a tool bar and three lists of information.

The lists include:

- **Packages** -- This list on the left of the window displays any packages installed on your system. Hovering your cursor over the package name displays the package description, if one is available.

- **Files** -- This list on the upper right of the window displays all files installed as a part of a selected package. All file names in this list are absolute.

- **Entries** -- This list on the lower left of the window 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 lists are empty, as no packages have been installed.

The buttons on the tool bar include:

- **Install Package**

- **Uninstall Package**
Creating Packages

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.

To create a package, use the following procedure.

1. Start the package manager. For additional detail, see Starting the Package Manager.

2. Determine whether you are creating a task package or a DCA package.

   – If you are creating a task package, click **New Task Package**.

   – If you are creating a DCA package, click **New DCA Package**.

---

**Note:** If you have not selected an installed package from the **Packages** list, then the **Uninstall Package** button is not enabled.

- **New Task Package**

- **New DCA Package**

**Note:** If you are running a stand-alone Info*Engine installation, then the New DCA Package button is not enabled.
The Select Tasks window opens.

The Select Tasks window contains three lists:

- The list on the upper left contains 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 list on the lower left contains a list of files that can be found within the directory selected in the upper left list.
- The list on the right contains the files and 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 \( \text{[>>]} \) 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 \( \text{[>>]} \) 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 `<<` to remove it from the package list.

4. Click **Next**.

The **Package Information** window opens.

Specify the identification and location information for the package:

– In the **Package Name** field, enter the package name. The package name should be descriptive, can contain spaces and any characters. The value entered for 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 file name. For the file name, any characters that are non-alphanumeric are removed. For example, if you enter the following value in the **Package Name** field when creating a DCA package:

  Ted’s Autostore Extravaganza

then the package file name will be:

  tedsAutostoreExtravaganza.ptcdar
– In the **Description** field, enter a description of the package. Any value entered in this field will display 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.

5. Click **Finish** to create the package.

**Installing Packages**

To install a package use the following procedure:

1. Start the Package Manager. For additional detail, see Starting the Package Manager.

2. Click **Install Package**.

   The **Open** window appears.

3. Browse to the package you want to install.

4. Select the package and click **Open**.

   The **Install** window appears, reporting on the progress of the installation. Any problems that occur during installation are registered in the progress report.
5. When the installation is complete, the OK button is enabled. Click OK to finish the installation.

Uninstalling Packages

Use the following procedure to uninstall a package:

1. Start the Package Manager. For additional detail, see Starting the Package Manager.

2. Click Uninstall Package.

The Uninstall window appears, reporting on the progress of the uninstall. Success or failure of the uninstall is indicated in the progress report. If the package was not successfully uninstalled, then the package name will remain listed in the Package Manager and contain only the contest that could not be successfully removed from the system.
3. When the uninstall is complete, the **OK** button is enabled. Click **OK** to finish the installation.
About the Info*Engine J2EE Connector

The Info*Engine Java 2 Enterprise Edition (J2EE) connector is Info*Engine’s implementation of J2EE Connector Architecture (JCA) version 1.0. JCA was designed to supply standard implementation for interaction between J2EE 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 J2EE application server environment (such as JBOSS or Sun ONE), or standalone Java client (command-line or swing application). The Info*Engine J2EE 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 additional information on configuring your application server environment for use with the Info*Engine J2EE 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:


The Info*Engine J2EE 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.

For additional information on SOAP, see Using SOAP Requests.

Info*Engine can describe a SOAP service to a client at runtime using Web Service Definition Language (WSDL). Delivered along with the J2EE 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 J2EE 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 J2EE connector can optionally implement. The Info*Engine J2EE connector implements the CCI.

**Note:** This information is provided as a general overview and is not required to write applications using the Info*Engine J2EE 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:

```java
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 further information, see Standalone Java SOAP Clients.
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 further information, see the Common Client Interface (CCI) section.

For example, a client could ask for metadata about a connection in the following manner:

```java
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 J2EE 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 J2EE 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 J2EE 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 the Standalone Java SOAP Clients section.

Transaction Contract
The Info*Engine J2EE connector contains LocalTransaction support only. Transactions are only supported if the connector is deployed against Windchill, and not if it is deployed against standalone Info*Engine. When using the Info*Engine J2EE connector from a J2EE 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 was started against. This must be used as the value of the INSTANCE parameter to Windchill adapter webobjects 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:

```java
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 will be ignored as if the given credentials are valid and are passed along to be used by supporting tasks. The task /com/infoengine/connector/validateUser.xml 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. For further information, see the Automatic EJB Generation section.

**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 further information on DAO generation see Standalone Java SOAP Clients.

The Info*Engine J2EE 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 J2EE 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
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.

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.

The javax.resource.cci.Interaction implementation supports execute with input record only. It does not support execute 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:

```java
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 ();
}

Standalone Java SOAP Clients

Standalone Java SOAP clients can use the J2EE 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 Example.

3. Create the required type identifiers and delegates in the LDAP directory. This is discussed in Registering Delegates.

4. Generate Data Access Objects (DAOs) for each CLASS from Step 1. DAOs are discussed further in Generating DAOs.

5. Decide how connections are to be created, and if necessary bind connection factory objects in the LDAP directory. This is described in Managing Connection Factories in the LDAP Directory.

6. Write client source code that uses the DAOs from Step 4 to access Info*Engine. This is described in Putting It All Together.

The following sections use a simple example application to demonstrate how a standalone Java SOAP client should communicate with Info*Engine.

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. For additional information, see Using SOAP Requests.

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.

Before You Begin

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:
  - $\{wt_home\}/lib/servlet.jar
  - $\{wt_home\}/codebase/WEB-INF/lib/ie.jar
  - $\{wt_home\}/codebase/WEB-INF/lib/ie3rdpartylibs.jar
  where wt_home is the Info*Engine installation directory.

- Extract the contents of $\{wt_home\}/ieconnector/ie.rar into a directory and add all JAR files found there to your CLASSPATH.

Implementing Tasks and Java Classes Example

For our example, the sum.xml and average.xml tasks are to be 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:

```xml
<%@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 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/taglib/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 additional information on registering tasks as delegates, see the Info*Engine Installation and Configuration Guide.

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:

```java
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):

```java
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 Making a Connection.

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 were unavailable, where a JMS SOAP connection issues a javax.jms.JMSException if some 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://localhost/Windchill/servlet/RPC". Depending on your configuration the default value may not be suitable, in which case you will need to 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 user name wcadmin and password wcadmin.

- The root of your Java source tree is at /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
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
<?xml version="1.0"?>
<project name="generateDAO" default="all" basedir="."/>

<property name="wt.home" value="/opt/ptc/Windchill"/>

<path id="cp">
  <pathelement location="${wt.home}/codebase/WEB-INF/lib/ie.jar"/>
  <pathelement location="${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>
```


### Making a Connection

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 re-use.
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 re-used. 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 re-configured in a single place as opposed to where ever a SOAP client resides. As long as the location of the LDAP directory does not change clients will 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**

The following variables are used within the tool:

- `-principal` -- The user name.
- `-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**
  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 that is discussed in the following section.

**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=My Company. This property is required.

provider.principal
If required by your LDAP access controls, specifies the principal needed to bind to provider.url. For example: cn=Manager.

provider.credentials
If required by your LDAP access controls, the password for provider.principal.

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
Specifications the alias of certificate to use. The default value for this property is "iesoap".

privateKeyAlias
Specifications the alias of the private key. The default value for this property is value of the certificateAlias property.

privateKeyPassword
Specifications 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=My Company
provider.principal=cn=Manager
provider.credentials=admin
```

Examples
Assuming the properties listed above 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=cxFactorHTTP,o=My Company":

```
java com.infoengine.connector.AdminTool -principal=cn=Manager -password=admin
 -bindConnectionFactory "ldap://localhost/o=My Company" cxFactory.HTTP
```

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=My Company" cxFactory.HTTP
```
Putting It All Together

The following 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.

```java
package org.myOrg.client;


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=My Company" );
        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.setConnectionFactoryImplementation ( "com.infoengine.connector.HTTPConnection" );
        // configure from properties file
        mcf.loadConnectionProperties ( new FileInputStream ( ".//http.properties" ) );
        // or via method call
        // mcf.setConnectionProperties ( ...
    }

    public static void main (String[] args) {
        // use either of the above factories
    }
}
```
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 );
    }
}
J2EE SOAP Clients

A J2EE SOAP client, most likely an EJB, can use the Info*Engine J2EE connector to interact with Info*Engine. Use of an application server can provide value added services to your connections, such as connection pooling, security features and transaction management.

An example of a J2EE SOAP client is provided in the following location:

\texttt{install/ieconnector/examples/bookstore.jar}

where \texttt{install} is the Info*Engine installation directory. For more information on the example J2EE SOAP client, see the \texttt{Info*Engine J2EE Book Viewer Example}.

An example of a J2EE SOAP client that interacts with Windchill and makes use of transactions is provided in the following location:

\texttt{install/ieconnector/examples/wtee.jar}

where \texttt{install} is the Windchill and Info*Engine installation directory.

For additional information on transactions, see the \textit{Windchill Adapter Guide}.

Writing a J2EE SOAP Client

Writing an Info*Engine J2EE SOAP client is only slightly more complicated than authoring a standalone client application. Info*Engine supplies a tool for generating Session Enterprise Java Bean (EJB) source code that can be deployed in your application server environment. These Session EJBs are very similar to the DAOs discussed in the standalone SOAP client section. In fact, the Session EJBs essentially wrap a generated DAO and expose the same method signatures to their clients.

Accessing Info*Engine using a Session EJB allows the J2EE application server to add value to your application. The J2EE application server can manage your connection factories, connections and connection pooling for you. The J2EE application server security features can be imposed on your connections to Info*Engine. The J2EE application server can be told to automatically manage transactions for your EJBs, beginning a transaction upon entering a transacted method and automatically committing or rolling back the transaction when the method is exited.

How an application server security mechanism applies varies from server to server. Some application servers support caller impersonation, while others may not. Using an application server's authentication mechanism may require synchronization between the server user database and that of Info*Engine. Some application servers support configuration allowing them to authenticate against the same user repository as Info*Engine, allowing for Single Sign On (SSO). This simplifies your J2EE application, allowing the application server to authenticate users for you.
The supplied EJB generation tool generates one DAO and EJB implementation per CLASS. The EJB home and remote method implementations are not generated for you. Utilities like Ant with XDOCLET can be used to generate this source code if you do not wish to do so by hand. When generating the EJB, source directives can be passed to XDOCLET for each method using the SOAP comments in a task.

**Note:** Transactions are only supported when deploying against Windchill and not standalone Info*Engine.

For example consider the following SOAP comment section:

```xml
<!--com.infoengine.soap.rpc.def
delete an object
@ejb:transaction type="Required"

@string obid - obid of the object to delete
--> 
```

**Note:** The @ejb comment line must be preceded by a single space.

This comment section would translate to a Session EJB method signature being generated similar to the following:

```java
/**
 * delete an object
 * @ejb:transaction type="Required"
 * @ejb:interface-method
 **/
public void deleteObject ( java.lang.String obid ) {
  ...
}
```

These comments then can be used as input to XDOCLET to generate EJB home and remote interface implementations, deployment descriptors, etc. In this case the J2EE application server would be required to wrap a deleteObject method invocation in a transaction.

**Automatic EJB Generation**

The class com.infoengine.connector.dao.EJBGGenerator can be used to automatically generate DAOs and Session EJBs for use in a J2EE application server. This tool can be used from a command-line or from within an Ant build script via the Ant extension com.infoengine.connector.dao.AntEJBGGenerator.

The usage of the command-line tool is:

```java
java com.infoengine.connector.dao.EJBGGenerator
  soapClass=<soap class>
  fileSystem=<directory>
  cxFactoryName=<string>
  [endPoint=<endPoint>] (default http://localhost/Windchill/servlet/RPC)
```
[cxFactoryJndiName=<string>] (defaults to cxFactoryName)
[package=<java package>] (default based on soapClass)
[baseClass=<java class name>] (default based on soapClass)
[ejbJndiName=<string>] (defaults to fully qualified ejb home)
[standAlone] (flag not specified means false)
[userCallerIdentity] (flag)
[runAs=<role-name>]
[securityRoleRefs=<role-maps> (comma delimited list of role-name=role-link)]
[unchecked]
[permittedRoles=<roles> (comma delimited list of role-names)]

The following is an example of an Ant target used to generate EJB source:

```
<target name="ejb.gen" depends="init.ie">
  <taskdef name="ejbgen"
    classname="com.infoengine.connector.dao.AntEJBGenerator">
    <classpath refid="xdoclet.path"/>
  </taskdef>
  <!-- generate the EJB source. -->
  <ejbgen endPoint="${ie.soap.wsdlUrl}"
    soapClass="infoengine.examples.bookstore.ejb"
    cxFactoryName="BookstoreInfoEngine"
    cxFactoryJndiName="java:/BookstoreInfoEngine"
    baseClass="BookstoreSession"
    fileSystem="${build.generate.dir}"
    package="infoengine.examples.bookstore.ejb"
    standalone="true"/>
</target>
```

If the application server is configured to authenticate users against the same repository as Info*Engine, and you wish to have those credentials passed on, then you can use the useCallerIdentity flag to signal this (your application server may require more configuration to allow passing of credentials to the connector). If you want to enforce role based access to your EJB then you can use the permittedRoles parameter to specify the roles that can access the EJB (and possibly specific methods). If your application maps LDAP groups to J2EE roles, then you can use the securityRoleRefs parameter to specify these mappings.

XDOCLET can then be used to generate the home and remote interfaces and J2EE deployment descriptors with an Ant target similar to the following:

```
<target name="ejbdoclet" depends="init.ie,compile.common,ejb.gen">
  <!-- use xdoclet to generate home/remote interfaces and deployment descriptors for JBoss -->
  <taskdef name="ejbdoclet" classname="xdoclet.ejb.EjbDocletTask">
    <classpath refid="xdoclet.path"/>
  </taskdef>
  <ejbdoclet
    sourcepath="${build.generate.dir}"
    destdir="${build.generate.dir}">
```

Advanced User Topics 6-69
Useful Client Side EJB Design Patterns

The following sections discuss client side EJB design patterns that you might find useful in writing J2EE SOAP clients.

EJB Home Factory

An EJB home factory is a class that is used to lookup and return EJB home objects. The idea behind an EJB home factory is to avoid the need for a client to deal with naming issues and naming related exceptions. In addition, an EJB home factory could introduce performance enhancements, such as the caching of home objects to avoid redundant lookups.

For an example of an EJB home factory see the infoengine.examples.bookstore.client.EJBHomeFactory class found in the following location:

```
install/ieconnector/examples/bookstore.jar
```

where install is the Info*Engine installation directory. For more information on the example J2EE SOAP client, see the Info*Engine J2EE Book Viewer Example.

Business Delegate

A business delegate is essentially a Java class that is used to insulate client code from interacting directly with a J2EE application server. There is typically a 1 to 1 correlation between a business delegate and a deployed Session EJB. A business delegate typically exposes methods identical to those exposed by the Session EJB.

Using a business delegate has several advantages:

- A business delegate can insulate the client code from EJB-related code. For example, use a business delegate for looking up and handling home objects, and exceptions (EJBException, RemoteException). The business delegate
A business delegate can allow the development of a client to be performed in parallel with the development of the server side EJBs. A business delegate can be used to return example data that can be used to develop clients before the actual server side EJBs are available.

- A business delegate can do intelligent error handling, such as the retrying of transactions that may have failed for non-fatal reasons, without the knowledge of the caller.

For an example of a business delegate see the infoengine.examples.bookstore.client.BookstoreSessionBusinessDelegate class found in the following location:

```
install/ieconnector/examples/bookstore.jar
```

where `install` is the Info*Engine installation directory. For more information on the example J2EE SOAP client, see the Info*Engine J2EE Book Viewer Example.
This chapter documents the example applications that are provided with Info*Engine.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK Example Application</td>
<td>7-2</td>
</tr>
<tr>
<td>Bookstore Example</td>
<td>7-6</td>
</tr>
<tr>
<td>Using the Bookstore Example</td>
<td>7-13</td>
</tr>
<tr>
<td>Info*Engine J2EE Book Viewer Example</td>
<td>7-49</td>
</tr>
</tbody>
</table>
SAK Example Application

The following example application provides a simple example that shows the use of the 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 that you supply when you run the application.

Compiling 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`
   `%WT_HOME%/lib/servlet.jar`
   `%WT_HOME%/codebase`
   `%WT_HOME%/codebase/WEB-INF/classes`
   `%WT_HOME%/codebase/WEB-INF/lib/ie.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/ie.jar
   ```

2. Create the directory `%WT_HOME%/codebase/WEB-INF/classes`

3. Change to the directory `%WT_HOME%/prog_examples/SAK_apps`, and execute the following:

   ```
   javac -d %WT_HOME%/codebase/WEB-INF/classes ExecuteTask.java
   ```

Running the Application

To run the application, use the following syntax:

```java
java com.infoengine.examples.applications.ExecuteTask -t <task_uri> -u <user name> -props <property resource> -n <naming service name>
```

where `<task_uri>` is the URI of the task to be executed, `<user name>` associates the user name specified with the task, `<property resource>` is the directory path to the property resource file, ie.properties, and `<naming service name>` uses the specified Naming Service.

For example:
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 <user name>
     * -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.out.println("Invalid option")
            }
        }
    }
}
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 ();
}
try {
    task = getTask (taskUri);
    if (task == null) {
        System.err.println ("Invalid task URI: "+ taskUri);
        System.exit (1);
        return;
    }
    // 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 <user name>");
    System.err.println ("\t-n <naming service name>");
    System.err.println ("\t-props <property resource>");
    System.exit (1);
}
**Bookstore Example**

Info*Engine provides a powerful means of accessing, manipulating, and displaying data from multiple sources. Previous sections of this User’s Guide have discussed the various components that make up Info*Engine and the various ways that those components are utilized.

This section is intended to provide you with a working example of a custom application created using Info*Engine. The bookstore example utilizes many Info*Engine features including display and task webjects, task execution, Info*Engine custom tags, JSP pages, and more to show you a sampling of what it is possible to achieve using Info*Engine.

**Introductory Topics**

The bookstore example makes the assumption that you have read the preceding sections of this guide, and have a basic understanding of the Java language, JSP and basic SQL.

The following is a list of references you may find helpful as you use the bookstore example:

- Using the Server Access Kit
- Info*Engine Tags
- Description of JSP directives, scriptlets, etc. as described in the Scriptlets, Expressions, Declarations, and Directives sections
- Display Webjects for HTML
- Management Webjects
- JSP Session scope as described in the page Directive section
- Dynamic Parameter Value Substitution
- JDBC adapter configuration and use, as described in the *JDBC Adapter Guide*
Bookstore Example Setup

Before exploring the bookstore example, you need to install the database. It is assumed that Info*Engine is already installed on your system. If not, please refer to the Info*Engine Installation and Configuration Guide. It is also assumed that the JDBC adapter is installed. The bookstore example makes extensive use of the JDBC adapter. Without the database and JDBC adapter, you can read the code, but not actually run the example.

The following sections describe the steps you need to take to get the bookstore ready for business.

Configure JDBC Adapter

The first step in the installation of the bookstore is to configure the necessary JDBC adapter. The adapter must be named "jdbcAdapter".

You can configure the adapter by creating an LDAP entry for the JDBC adapter using the Info*Engine Property Administrator, or by asking your Info*Engine administrator to create one for you. On the JDBC adapter form in the Property Administrator, enter "jdbcAdapter" as the Service Name. For help on creating the LDAP entry, click the Help link on the form.

Run Initialization Script

Run the following initialization script:

- For Windows systems:
  ```
  cd <ie_dir>/prog_examples/bookstore
  init.bat
  ```

- For UNIX systems:
  ```
  cd <ie_dir>/prog_examples/bookstore
  init.sh
  ```

where <ie_dir> is the location of the Info*Engine installation.

Running this script accomplishes the following actions:

- Compiles the example source code into
  ```
  <ie_dir>/codebase/WEB-INF/classes
  ```

- Uncompresses the book jacket images

- Runs a Java application that initializes the required bookstore tables using the jdbcAdapter.

**Note:** The bookstore example was created using an Oracle database. If you use a different database, you may need to change lines in the example code to utilize the bookstore example with your database.
Additional Steps for Configuring the Message Oriented Middleware (MOM)

The bookstore example makes use of a Java Message Service (JMS) queue to persist shopping basket information across multiple HTTP sessions. The administered object required to show this functionality is named “cn=bookBaskets”. The bookstore example also makes use of one JMS Topic to automatically re-stock books whose inventory has fallen below a configured threshold. The administered object required to try out this functionality is named “cn=bookReorder”.

**Note:** The bookstore example runs without a MOM configured, but the MOM dependant functionality is not then available. If you do not have a MOM or are not interested in trying out this functionality here, you do not need to read this section.

Info*Engine MOM functionality is provided by third-party software such as IBM MQSeries. For further information about installing MQSeries, refer to the *Info*Engine Installation and Configuration Guide.

You will use the Info*Engine Property Administrator for configuring properties. For a discussion on using the Administrator see the *Info*Engine Installation and Configuration Guide.

The following is a list of the steps to take to configure the MOM for use with the bookstore example.

1. Create a location in your LDAP directory to store your administered objects. For example:
   cn=MQSeries,dc=myHost,dc=myLocation,dc=myCompany,dc=com
2. Create a QueueConnectionFactory administered object and the corresponding queue using your MOMs administration tools.
3. Create a TopicConnectionFactory administered object
4. Create the cn=bookBaskets Queue administered object
5. Create the cn=bookReorder Topic administered object
6. Set the JMS Base URI property on the servlet form. For example:
   ldap://<host>/cn=MQSeries,dc=myHost,…
   (If credentials are required to access the administered objects in this location they should be supplied on the URI like:
   ldap://<user>:<password>@<host>/cn=MQSeries,dc=myHost,…)
7. Set the Queue Connection Factory property on the servlet form with the administered object reference created in step 2 (this value should be relative to the URI specified in step 6).
8. Set the Topic Connection Factory property on the servlet form with the administered object reference created in step 3 (this value should be relative to the URI specified in step 6).
9. Restart the servlet engine.
Survey of JSP Pages

The following JSP pages are used in this example:

- basket.jsp
- best_sellers.jsp
- book.jsp
- customer.jsp
- get_jacket.jsp
- index.jsp
- inventory.jsp
- mail_admin.jsp
- search.jsp

index.jsp

The index.jsp is the main display page for the bookstore. It provides the bookstore title header, the search form, the browse genre list, and the shopping basket links. Depending on the user’s activity, content from other JSP pages gets included on this page’s display. This other content includes the best-seller list, search results, individual book information, customer login and profiles, and the shopping basket.

basket.jsp

The basket.jsp page manages the bookstore’s shopping basket page. This page gathers the information for the books to be ordered and triggers the task that places the order. This page gets included in the index.jsp page display.

best_sellers.jsp

The best_sellers.jsp page displays the top five best-selling books, if any books have been purchased from the bookstore, with links to the individual books’ information. This page gets included in the index.jsp page display.

book.jsp

The book.jsp page gathers and displays information about a book, including title, publisher, cost, description, and reviews. This page gets included in the index.jsp page display.

customer.jsp

The customer.jsp page manages customer information, allowing customers to log in to an existing customer profile, update their profile, or create a new customer profiles. This page gets included in the index.jsp page display.

get_jacket.jsp

The get_jacket.jsp page is used to retrieve .jpeg images of the book jackets from the database, where they are stored as BLOBs.
inventory.jsp

The inventory.jsp page is a standalone page that allows the database administrator to check bookstore inventory, and to manually set the number of books in inventory.

mail_admin.jsp

The mail_admin.jsp page is a standalone page which allows the administrator to set mail settings, including server, authentication information, and the reply to address. The page also allows the administrator to set the re-order threshold for the bookstore inventory, and subscribe to the bookReorder event so that books can be automatically re-stocked (if a MOM is configured).

search.jsp

The search.jsp page performs searches and displays the search results. Searches are performed when the user either enters data into the search form, or clicks on a genre. This page gets included in the index.jsp page display.

Survey of XML Tasks

The following XML pages are used in this example:

- bestsellers.xml
- listen4restock.xml
- mailorder.xml
- placeorder.xml
- restock.xml
- updatecustomer.xml

bestsellers.xml

The bestsellers.xml task compiles a list of the current best-selling books in the bookstore.

listen4restock.xml

The listen4restock.xml task subscribes to the restock event so that the restock.xml task is automatically called when necessary.

mailorder.xml

The mailorder.xml task sends an e-mail to the customer following the placement of an order, if the information requested on the mail_admin.jsp page has been filled in.

placeorder.xml

The placeorder.xml task updates the bookstore tables to place an order, and, if the MOM is configured, emits a restock event if necessary.
restock.xml

The restock.xml task automatically resets the inventory of a book to 100 copies as a result of a restock event, if the MOM is configured.

updatecustomer.xml

The updatecustomer.xml task creates new customer profiles, or updates existing customer information.

Database Tables

The bookstore example database contains the following tables and columns:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns in Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOK</td>
<td>ISBN, PUBID, GID, AUTHOR, TITLE, DESCRIPTION, COST</td>
</tr>
<tr>
<td>PUBLISHER</td>
<td>PUBID, PUB_TITLE</td>
</tr>
<tr>
<td>REVIEW</td>
<td>ISBN, TEXT</td>
</tr>
<tr>
<td>GENRE</td>
<td>GID, GENRE_TITLE</td>
</tr>
<tr>
<td>JACKET</td>
<td>ISBN, NAME, FILECONTENT</td>
</tr>
<tr>
<td>INVENTORY</td>
<td>ISBN, QUANTITY</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>FNAME, SNAME, ADDRESS1, ADDRESS2, CITY, STATE, COUNTRY, ZIP, EMAIL</td>
</tr>
<tr>
<td>BOOKORDER</td>
<td>ORDERNUM, EMAIL, ISBN, QUANTITY</td>
</tr>
<tr>
<td>MAILSETTINGS</td>
<td>MAIL_FROM, REPLY_TO, MAIL_SERVER, USERNAME, PASSWORD, ROTHRESH</td>
</tr>
</tbody>
</table>
**Resource Bundles**

All of the text messages displayed by the JSP pages in this example are derived from resource bundles. In this case, the resource bundles are being stored as straight property .rbInfo files. Localized messages could also be retrieved from compiled Java resource bundles or .rbInfo-like properties stored in an LDAP directory. The language to display is determined by using the default language set by the Web browser.

As you look at the code sections in this bookstore example and view the actual code files, you will see many instances of the displayResource tag. This tag causes the localized content to be retrieved from a resource bundle for display.

The .rbInfo files used by this example are found in the following locations:

<ie_dir>/tasks/infoengine/examples/bookstore/BookstoreResource.rbInfo
<ie_dir>/tasks/infoengine/examples/bookstore/BookstoreResource_latin_PIG.rbInfo

The bookstore example currently supplies two languages in the .rbInfo file: English (US) and Pig Latin. To view the Pig Latin messages, follow the directions for your Web browser to change the default language.

To view the bookstore in Pig Latin with Microsoft Internet Explorer 5.x:

- Select **Tools > Internet Options**.
- On the **General** tab click **Languages**.
- In the **Language Preference** dialog box click **Add**....
- In the **User defined** text area type "latin-pig" and click **OK**.
- Move "latin-pig" to the top of the list using the **Move Up** button.
- Click **OK** to close the dialog box and **OK** to close **Internet Options**.
- Restart your browser and access the bookstore.

To view the bookstore in Pig Latin with Netscape Communicator:

- Select **Edit > Preferences**....
- Under Navigator/Languages click **Add**....
- In the **Others...** text area type "latin-pig" and click **OK**.
- Move "latin-pig" to the top of the list using the up arrow.
- Click **OK**.
- Access the bookstore.
**Note:** Setting Pig Latin as the default language will change only the messages retrieved from resource bundles. The graphics and data stored in the database are not localized.

### Using the Bookstore Example

You can run the bookstore example by entering a URL in the following form in your Web browser:

```
http://<hostname>/Windchill/infoengine/jsp/examples/bookstore/
```

where `<hostname>` is the name of the computer where Info*Engine resides. This form of the URL assumes that Info*Engine was installed using the default "Windchill" application prefix. If your Info*Engine administrator has configured your Info*Engine installation differently, then determine the bookstore example URL based on the customized configuration.

The code sections included in the following text are approximations of the actual code. Each section is identified with the name of the JSP page or task from which it was taken. The actual bookstore example code is found in the following locations:

- **JSP pages:**
  ```
  <ie_dir>/codebase/infoengine/jsp/examples/bookstore/
  ```

- **XML tasks and .rbInfo files:**
  ```
  <ie_dir>/tasks/examples/bookstore/
  ```

- **Command line source code**
  ```
  <ie_dir>/prog_examples/bookstore/
  ```

### Specifying Mail Server Settings

The mail_admin.jsp page can be reached by entering a URL into your browser in the following form:

```
http://<hostname>/Windchill/infoengine/jsp/examples/bookstore/mail_admin.jsp
```

where `<hostname>` is the name of the server where Info*Engine resides. If you have configured your Info*Engine installation differently, then you will have to determine the URL based on your customized configuration.

The mail_admin.jsp page performs two functions: updating the MAILSETTINGS database table, and executing the listen4restock.xml task.
The following screenshot displays the **Mail Server Settings** screen.

<table>
<thead>
<tr>
<th>Mail Server Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>From:</strong></td>
</tr>
<tr>
<td><strong>Reply To:</strong></td>
</tr>
<tr>
<td><strong>Mail Server:</strong></td>
</tr>
<tr>
<td><strong>Username:</strong></td>
</tr>
<tr>
<td><strong>Password:</strong></td>
</tr>
<tr>
<td><strong>Re-Order Threshold:</strong></td>
</tr>
</tbody>
</table>

The following lists the functions of the text boxes displayed on the **mail_admin.jsp** page:

- **From** -- E-mail address from which order confirmation e-mails are sent.
- **Reply To** -- E-mail address where the customer can send a reply to the order confirmation e-mail.
- **Mail Server** -- Name of the mail server the From address resides on.
- **Username** -- User name needed to connect to the Mail Server.
- **Password** -- Password of the user name specified.
- **Re-Order Threshold** -- Threshold number below which books will automatically be re-ordered, if the MOM is installed and configured.

**Note:** It is important that values are filled in for every text box, and none are left blank, due to lack of error checking in the code.

Clicking the **Update Mail Settings** button causes the information to be saved in the MAILSETTINGS database table. The old settings (if any) are deleted and the new settings are inserted using Do-SQL webjcts, similar to the manner in which the book inventory is updated by the placeorder.xml task discussed in the first scenario. For further details, please view the code for **mail_admin.jsp**.
If the MOM is installed and configured, clicking the **Automatically Re-Stock Books** button subscribes the Info*Engine servlet to handle reorder events by triggering the listen4restock.xml task.
Scenario Overview

The following sections take you through two scenarios of users interacting with the bookstore. Each scenario is introduced, then the actions and concepts involved are discussed. Interspersed with the discussions will be screen captures of the bookstore and sections of code showing the underlying Info*Engine functionality behind the display on the screen.

First Time Customer Scenario

Mr. Jones wants to purchase a book as a birthday present for his niece. He visits Info*Engine.books.com. Knowing that his niece is a Jane Austen fan, he searches by author for "Austen". From the resulting list, he picks *Pride and Prejudice*. He views the book’s information, and reads the review.

Having decided to make his purchase, Mr. Jones clicks the Add To Basket icon to add the book to his shopping basket. The Customer Information form appears. He enters his customer information to create a new customer account. He views his shopping basket, places his order, and receives e-mail confirmation of the order. After successfully placing the order, Mr. Jones signs out of the bookstore.

Survey of Included Topics

The preceding scenario incorporates the following topics:

- jsp:include tag
- simple JDBC query
- advanced JDBC query
- Info*Engine substitution from a VDB group
- task execution
- task webjects
- display webjects
- JDBC adapter webjects
- result sets
- forEach tag
- getValue tag

Persisting the contents of the Info*Engine VDB using a JMS Queue (if a MOM is configured)
Visit Bookstore

Mr. Jones wants to purchase a book as a birthday present for his niece. He visits Info*Engine.books.com.

The opening bookstore screen derives from the index.jsp page. If you haven’t visited the bookstore before you will not see a best sellers list since no orders have been placed, instead you will see a message relating the bookstore is open for business. The lower left hand corner contains the Browse list. The upper right hand corner contains the Search box. The lower right hand corner contains the View Shopping Basket icon. The content in the main section of the screen, in this case the Info*Engine Best Sellers List, is derived from an included .jsp page.

In the index.jsp code is the following scriptlet:

```java
String jsp_page = request.getParameter ( "page" );
if ( jsp_page == null ) jsp_page = "best_sellers";
jsp_page += ".jsp";
```

The "page" parameter is passed along with each request, depending on the action chosen by the user. The requested page is then included in the index.jsp page using the jsp:include tag.

```jsp
<jsp:include page="<%=jsp_page%>" flush="true"/>
```

For example, if the user chooses a genre from the Browse list, then the page parameter is set to "search" and the search.jsp page will be included as the main content on the displayed browser screen. If no action is chosen, the best_sellers.jsp is the default JSP page to included. In the case of this opening screen, the Info*Engine Best Sellers List is displayed as no other page parameter is supplied. The best_sellers.jsp page will be discussed in more detail in a later section.
Search for Author

Knowing that his niece is a Jane Austen fan, Mr. Jones searches by author for "Austen".

![Search for Author](image)

The drop-down menu items correspond to columns found in the BOOK database table.

![Search for Author](image)

The text entered in the text field is used to build a query against a particular database column. In this case, the AUTHOR column is used.

This search demonstrates the use of an advanced JDBC query. The search criteria entered by the user are used to select books from a database table for display.

Clicking the Search button causes a request to be made for the search.jsp page along with the column and text for which to search. This information is used to build an SQL query.

search.jsp:

```java
String whereClause;
whereClause =
"SELECT * FROM BOOK WHERE upper(" + column + ") LIKE upper('%" + query + '%")";
```

The JDBC adapter Do-SQL webect performs this query.

search.jsp:

```xml
<ie:webject name="Do-SQL" type="ACT">
  <ie:param name="INSTANCE" data="jdbcAdapter"/>
  <ie:param name="SQL" data="<%=whereClause%>"/>
  <ie:param name="GROUP_OUT" data="books"/>
</ie:webject>
```
The Info*Engine group containing the query results, named "books", remains in the VDB across multiple requests because the JSP pages make use of session scope and the Info*Engine tag library extensions exploit the JSP session to persist the VDB across multiple requests automatically. Session scope is enabled on a JSP page by setting session="true" on the @page directive at the top of the page.

If the query results exceed five titles, then the use of result sets is demonstrated. For example, the following picture shows a navigation bar for a result set of 18 books.

Displaying 6 - 10 of 18 books

Previous 0 1 2 3 Next

Embedded JSP scriptlets calculate and keep track of the location in the query results. The location is displayed using localized messages provided by the Display-Resource webject.

search.jsp:

```java
int ecount = ieObj.getElementCount( "books" );
int irsStart = Integer.parseInt ( rsStart );
int rsEnd = irsStart + 5;
if ( rsEnd > ecount ) rsEnd = ecount;
String dStart = ""+(irsStart+1);
String dEnd = ""+rsEnd;
String dCount = ""+ecount;
<font color=#6f6f6f size=-1>
<ie:webject name="Display-Resource" type="DSP">
<ie:param name="PROPERTIES" data="examples/bookstore/BookstoreResource"/>
<ie:param name="KEY" data="RESULT_SET_INFO"/>
<ie:param name="PARAM" data="\"%dStart\""/>
<ie:param name="PARAM" data="\"%dEnd\""/>
<ie:param name="PARAM" data="\"%dCount\""/>
</ie:webject></font>
```

A navigation bar is also displayed using localized messages provided by the displayResource tag.

search.jsp:

```java
if ( ecount > 5 ) {
    int pages = ( ecount / 5 ) + { ( ecount % 5 ) != 0 ) ? 1 : 0 };
    if ( rsPage > 0 ) {%
        <%=a href="index.jsp?page=search&rsPage=%"+(rsPage-1)%>"
        <ie:displayResource
            properties="examples/bookstore/BookstoreResource"
            key="PREVIOUS"/"></a>&nbsp; <%
    } else {%
        <font color=#ffffff><ie:displayResource
            properties="examples/bookstore/BookstoreResource"
            key="PREVIOUS"/></font>&nbsp; <%
    }
    for ( int i = 0; i < pages; i++ ) {%
        if ( i == rsPage ) {%
            <%=a href="index.jsp?page=search&rsPage=%"+i%>"
        } else {%
            <%=a href="index.jsp?page=search&rsPage=%"+i%>"
            <ie:displayResource
                properties="examples/bookstore/BookstoreResource"
                key="PREVIOUS"/></a>&nbsp; <%
        }
        if ( rsPage != (pages-1) ) {%
            <%=a href="index.jsp?page=search&rsPage=%"+(rsPage+1)%>"
            <ie:displayResource
                properties="examples/bookstore/BookstoreResource"
                key="NEXT"/></a>&nbsp; <%
        } else {%
            <font color=#ffffff><ie:displayResource properties="examples/bookstore/BookstoreResource" key="NEXT"/></font>
        }
    }
}
```

The actual display of the search results is performed by the Display-Object display webject. The START and MAX parameter values are passed from the embedded scriptlets to the Display-Object webject, causing the webject to display five records at a time, starting at an appropriate index into the search results.

search.jsp:

```xml
<ie:webject name="Display-Object" type="DSP">
    <ie:param name="GROUP_IN" data="books"/>
    <ie:param name="attribute" data="<%=href%>"/>
    <ie:param name="BORDER" data="0"/>
    <ie:param name="START" data="<%=rsStart%>"/>
    <ie:param name="MAX" data="5"/>
</ie:webject>
```
Select and View Book

From the resulting list, Mr. Jones picks *Pride and Prejudice*. He views the book’s information, and reads the review.

```
String isbn = request.getParameter ("isbn");
String whereClause = "ISBN='" + isbn + "'";
```


```
String isbn = request.getParameter ("isbn");
String whereClause = "ISBN='" + isbn + "'";
```

Picking a title from a result set uses a simple JDBC query. When the user clicks on the book title, a number of actions are initiated.


```
String isbn = request.getParameter ("isbn");
String whereClause = "ISBN='" + isbn + "'";
```

A request is made for get_jacket.jsp with the ISBN of the book. This JSP page uses the ISBN to select the appropriate image from the JACKETS database table and streams it back to the web browser to be displayed using an HTML IMG tag in the main bookstore display.

```
<!--
String whereClause = "ISBN='" + request.getParameter ("isbn") + "'";
String contentType = "image/jpeg";
response.setContentType (contentType);
contentType = "image/jpeg";
-->
```

<%String whereClause = "ISBN='" + request.getParameter ("isbn") + "'";
String contentType = "image/jpeg";
response.setContentType (contentType);
contentType = "image/jpeg";\n%>
<ie:webject name="Send-Blob-Stream" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="JACKET"/>
    <ie:param name="ATTRIBUTE" data="FILECONTENT"/>
    <ie:param name="MIMETYPE" data="%contentType%"/>
    <ie:param name="WHERE" data="%whereClause%"/>
    <ie:param name="FILENAME" data="down.jpg"/>
```
Note: The lack of extra spaces between %> and <% tags is very important when responding with binary content, such as BLOBs. Anything outside of the %> and <% tags is sent by the JSP processor and causes the content received by the client to contain extra bytes which do not reflect the actual binary content and may cause problems.

Two queries are performed to retrieve the textual information to display about the book.

book.jsp:

The first Query-Objects webject retrieves data from the BOOK database table and stores an Info*Engine group named "book" in the VDB. The second Query-Objects webject retrieves data from the PUBLISHER database table and stores an Info*Engine group named "publisher" in the VDB. The second webject makes use of Info*Engine substitution to extract data from the "book" group to select the appropriate data from the PUBLISHER database table. For information about substitution, see the section titled Dynamic Parameter Value Substitution.

The Join-Groups webject is used to join the data from the two queries for display:

book.jsp:

This webject joins the data retrieved by the previous queries forming a single Info*Engine group of their results. The two groups are joined by the common column named PUBID. As a result of this operation, a new Info*Engine group named "book2"is added to the VDB. This consolidated Info*Engine group is then used to display the resulting data.
Finally the ISBN is used to query the REVIEW database table creating an Info*Engine group in the VDB named "reviews".

```java
String isbn = request.getParameter ( "isbn" );
String whereClause = "ISBN='" + isbn + "'";
...
<ie:webject name="Query-Objects" type="OBJ">
  <ie:param name="INSTANCE" data="jdbcAdapter"/>
  <ie:param name="CLASS" data="REVIEW"/>
  <ie:param name="WHERE" data="\"%\"+whereClause\"\"/>
  <ie:param name="GROUP_OUT" data="reviews"/>
</ie:webject>
```

The forEach and getValue custom tags are used to display the data. The query is very similar to those shown above.

```java
<h3>Reviews</h3>
<ie:forEach groupIn="reviews" groupOut="review">
  <blockquote><ie:getValue name="TEXT"/></blockquote>
</ie:forEach>
```

The forEach tag loops through each element in the "reviews" group creating a new group named "review". The getValue tag could specify a groupIn parameter with a value of "review". If groupIn isn’t specified, this tag and any Info*Engine display webjects use the last group added to the VDB as their default input group. the getValue tag in the above code is taking advantage of this.
The following screenshot shows the resulting display.

Add Book to Shopping Basket

Having decided to make his purchase, Mr. Jones clicks the "Add To Basket" icon to add book to his shopping basket.
The Add To Basket and Look In Basket icons are located on the bottom right of the browser window. The following code is located in index.jsp.

index.jsp:

```jsp
<% if ( isbn != null ) { %>
<a href="index.jsp?page=<%=((signed_in) ? "basket" : "customer")%>&isbn=<%=%isbn%>">
<img src="img/buy.gif" border=0 alt="<ie:displayResource properties="examples/bookstore/BookstoreResource" key="ADD_TO_BASKET"/>">
</a>
<% } %>

<a href="index.jsp?page=<%=((signed_in) ? "basket" : "customer")%>">
<img src="img/basket.gif" border=0 alt="<ie:displayResource properties="examples/bookstore/BookstoreResource" key="LOOK_IN_BASKET"/>">
</a></td></tr>
```

If the user is not already signed in or is a new customer, as is the case in this scenario, then the user is redirected to the customer.jsp page.

Enter Customer Information

The Customer Information form appears. Mr. Jones enters his customer information to create a new customer account.
Customer information and sign-in are managed by the customer.jsp page.

First, an empty customer record is created.

customer.jsp:

```java
if ( create ) {
    String element = "fname=:sname=:address1=:address2=:city=:state=:country=USA:zip=:email=";

    <ie:webject name="Create-Group" type="GRP">
        <ie:param name="ELEMENT" data="<%=element%>"/>
        <ie:param name="GROUP_OUT" data="customerInfo"/>
    </ie:webject>
```

The following code allows the user to fill in the empty customer record.

customer.jsp:

```java
com.infoengine.object.factory.Group custinfo =
    ieObj.getGroup ( "customerInfo" );
updateAttribute ( custinfo, "fname", request.getParameter ( "fname" ) );
updateAttribute ( custinfo, "sname", request.getParameter ( "sname" ) );
updateAttribute ( custinfo, "address1", request.getParameter ( "address1" ) );
updateAttribute ( custinfo, "address2", request.getParameter ( "address2" ) );
updateAttribute ( custinfo, "city", request.getParameter ( "city" ) );
updateAttribute ( custinfo, "state", request.getParameter ( "state" ) );
updateAttribute ( custinfo, "country", request.getParameter ( "country" ) );
updateAttribute ( custinfo, "zip", request.getParameter ( "zip" ) );
updateAttribute ( custinfo, "email", request.getParameter ( "email" ) );
```

The updateAttribute function in the above code section is specified in the declaration block (within `<%! and `%>` tags) of the JSP page. It manually manipulates an Info*Engine group replacing an attribute with the requested value.

After filling in the customer information, the user then clicks the Create button, causing the updatecustomer.xml task to execute.

customer.jsp:

```xml
<ie:task uri="examples/bookstore/updatecustomer.xml">
    <ie:param name="GROUP_IN" data="customerInfo"/>
</ie:task>
```

The updatecustomer.xml task updates customer information through a two step process. The task first deletes the existing information in the customer profile (in this case nothing happens because the customer profile is empty).

updatecustomer.xml:

```xml
<ie:webject name="Do-SQL" type="ACT">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="SQL" data="DELETE FROM CUSTOMER WHERE EMAIL='${customerInfo[0].EMAIL[0]}';"/>
    <ie:param name="GROUP_OUT" data="sqlResults"/>
</ie:webject>
```
The task then inserts the new customer information into the blank customer profile.

updatecustomer.xml:

```
<ie:webject name="Do-SQL" type="ACT">
  <ie:param name="INSTANCE" data="jdbcAdapter"/>
  <ie:param name="SQL" data="INSERT INTO CUSTOMER VALUES
    ('${customerInfo[0]FNAME[0]}','${customerInfo[0]SNAME[0]}','
    ${customerInfo[0]ADDRESS1[0]}','${customerInfo[0]ADDRESS2[0]}','
    ${customerInfo[0]CITY[0]}','${customerInfo[0]STATE[0]}','
    ${customerInfo[0]COUNTRY[0]}','${customerInfo[0]ZIP[0]}','
    ${customerInfo[0]EMAIL[0]}')"/>
  <ie:param name="GROUP_OUT" data="sqlResults"/>
</ie:webject>
```

The browser page is then redirected from customer.jsp to basket.jsp using jsp:include tags.

Place Order

Mr. Jones views his shopping basket, places his order, and receives e-mail confirmation of the order.
The shopping basket is displayed by the basket.jsp page.

```jsp
<ie:webject name="Create-Group" type="GRP">
    <ie:param name="GROUP_OUT" data="shoppingBasket"/>
</ie:webject>

com.infoengine.object.factory.Group basket = 
    ieObj.getGroup ("shoppingBasket");
if ( isbn != null ) {
    String ordernum = (String)basket.getAttributeValue ( 0, "ordernum" );
    if ( ordernum == null )
        ordernum = "" + (new java.util.Date()).getTime();
}
```

The customer information is displayed at the bottom of the shopping basket.

```jsp
<ie:getValue groupIn="customerInfo" name="FNAME"/>
<ie:getValue groupIn="customerInfo" name="SNAME"/>
<br>
<ie:getValue groupIn="customerInfo" name="ADDRESS1"/>
<br>
<% String addr2 = ieObj.getAttributeValue ("customerInfo", 0, "ADDRESS2");
    if ( addr2 != null && !addr2.trim().equals("")) {
        <%=addr2%>
    }
%>
<ie:getValue groupIn="customerInfo" name="CITY"/>
<ie:getValue groupIn="customerInfo" name="STATE"/>
<ie:getValue groupIn="customerInfo" name="ZIP"/>
<br>
<ie:getValue groupIn="customerInfo" name="EMAIL"/>
```

Adding the requested book to the shopping basket involves a number of steps. First, a call to the Query-Objects JDBC adapter webject creates a group in the VDB named "basketItem".

```jsp
<ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="BOOK"/>
    <ie:param name="WHERE" data="<%=filter%>"/>
    <ie:param name="GROUP_OUT" data="basketItem"/>
</ie:webject>
```

A call to the Query-Objects JDBC adapter webject creates a group in the VDB named "itemQuantity" including the order number and an automatic quantity of 1.

```jsp
String element = "ordernum=" + ordernum + ":isbn=" + isbn + ":quantity=1";
<ie:webject name="Create-Group" type="GRP">
    <ie:param name="GROUP_OUT" data="itemQuantity"/>
    <ie:param name="ELEMENT" data="<%=element%>">
</ie:webject>
```
The Join-Groups webject combines the "itemQuantity" and "basketItem" groups, resulting in a new "basketItem" group.

```xml
<ie:webject name="Join-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="basketItem"/>
  <ie:param name="GROUP_IN" data="itemQuantity"/>
  <ie:param name="JOINBY" data="isbn"/>
  <ie:param name="GROUP_OUT" data="basketItem"/>
</ie:webject>
```

The new basket item is then added to the shopping basket using the Concat-Groups webject.

```xml
<ie:webject name="Concat-Groups" type="GRP">
  <ie:param name="GROUP_IN" data="shoppingBasket"/>
  <ie:param name="GROUP_IN" data="basketItem"/>
  <ie:param name="GROUP_OUT" data="shoppingBasket"/>
</ie:webject>
```

The user places the order by clicking the **Place Order** button. This action triggers the `placeorder.xml` task.

```xml
<ie:task uri="examples/bookstore/placeorder.xml">
  <ie:param name="GROUP_IN" data="shoppingBasket"/>
  <ie:param name="GROUP_IN" data="customerInfo"/>
</ie:task>
```

The `placeorder.xml` task updates the database tables to place the order. The task checks to make sure that each item in the basket has sufficient inventory for the order. If the quantity ordered is greater than the inventory amount, an error message is displayed and an exception is thrown.

```xml
<ie:forEach groupIn="shoppingBasket" groupOut="basketItem">
  <ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="INVENTORY"/>
    <ie:param name="WHERE" data="ISBN=#{basketItem[0].ISBN[0]}"/>
    <ie:param name="GROUP_OUT" data="itemInventory"/>
  </ie:webject>
  <% int ordered = Integer.parseInt ((String)getGroup ( "basketItem" ).getAttributeValue ( 0, "QUANTITY" ).toString() );
    int stocked = Integer.parseInt ((String)getGroup ( "itemInventory" ).getAttributeValue ( 0, "QUANTITY" ).toString() );
    if ( ordered > stocked ) {
      RbInfoResourceBundle rbinfo = new RbInfoResourceBundle ( 
        "examples/bookstore/BookstoreResource" );
```
String message = rbinfo.getLocalizedMessage("INSUFFICIENT_INVENTORY", new Object[] { getGroup("basketItem").getAttributeValue(0, "TITLE") });

%><ie:webject name="Throw-Exception" type="MGT">
  <ie:param name="CLASS" data="examples.bookstore.InsufficientInventoryException"/>
  <ie:param name="MESSAGE" data="<%=message%>"/>
</ie:webject>

If the quantity ordered drops the inventory amount below the inventory threshold, then an element is added to the VDB group named "reorder" which will later be used to emit bookReorder events.

placeorder.xml:

    Group reorder = new Group("reorder");

    ...  
    if ( threshold > 0 && (stocked-ordered) < threshold ) {
        com.infoengine.object.factory.Element e = new com.infoengine.object.factory.Element ();
        e.addAtt ( new com.infoengine.object.factory.Att ("ISBN", (String)getGroup ( "basketItem" ).getAttributeValue ( 0, "ISBN" ) ) );
        reorder.addElement ( e );
    }

This validation is performed for each basket item using the forEach custom Info*Engine tag.

The task then begins the order process. The inventory for each item is checked again.

placeorder.xml:

    <ie:forEach groupIn="shoppingBasket" groupOut="basketItem">
        <ie:webject name="Query-Objects" type="OBJ">
            <ie:param name="INSTANCE" data="jdbcAdapter"/>
            <ie:param name="CLASS" data="INVENTORY"/>
            <ie:param name="WHERE" data="ISBN='${basketItem[]ISBN[0]}')'"/>
            <ie:param name="GROUP_OUT" data="itemInventory"/>
        </ie:webject>

        An SQL statement is manually built. Standard substitution such as:
        
        ${basketItem[]ISBN[0]}
        
        does not work in this instance, as it would be buried within a java variable.

placeorder.xml:

    String sql = "INSERT INTO INVENTORY VALUES ('" +
                getGroup ( "basketItem" ).getAttributeValue ( 0, "ISBN" ) +
                ");

    String sql = "INSERT INTO INVENTORY VALUES ('" +
                getGroup ( "basketItem" ).getAttributeValue ( 0, "ISBN" ) +
                ");
The Do-SQL webject is used in series to delete the existing inventory data, and replace it with the updated, post order inventory data.

placeorder.xml:

```xml
<ie:webject name="Do-SQL" type="ACT">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="SQL" data="DELETE FROM INVENTORY WHERE ISBN='${basketItem[0].ISBN[0]}'/">
    <ie:param name="GROUP_OUT" data="sqlResults"/>
</ie:webject>
<ie:webject name="Do-SQL" type="ACT">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="SQL" data="<%=sql%>">
    <ie:param name="GROUP_OUT" data="sqlResults"/>
</ie:webject>
<ie:webject name="Do-SQL" type="ACT">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="SQL" data="INSERT INTO BOOKORDER VALUES ('${basketItem[0].ORDERNUM[0]}','${customerInfo[0].EMAIL[0]}','
${basketItem[0].ISBN[0]}',{basketItem[0].QUANTITY[0]})"/>
    <ie:param name="GROUP_OUT" data="sqlResults"/>
</ie:webject>
```
This action is performed for each order item using the forEach custom Info*Engine tag.

The bookstore then attempts to send an e-mail notification of the order to the customer using the mailorder.xml task.

basket.jsp:

```xml
<ie:task uri="examples/bookstore/mailorder.xml">
    <ie:param name="GROUP_IN" data="customerInfo"/>
    <ie:param name="GROUP_IN" data="shoppingBasket"/>
</ie:task>
```

The mailorder.xml task first checks to see if the mail settings have been set for the bookstore. If the MAIL_SERVER is set, then the task assumes that the other mail settings are also set.

mailorder.xml:

```xml
Object eaddr = getGroup ( "customerInfo" ).getAttributeValue ( 0, "EMAIL" );
if ( eaddr != null ) { %>
<ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="MAILSETTINGS"/>
    <ie:param name="WHERE" data="()"/>
    <ie:param name="GROUP_OUT" data="mailSettings"/>
</ie:webject>

Object mserver = getGroup ( "mailSettings" ).getAttributeValue ( 0, "MAIL_SERVER" );
if ( mserver != null && !((String)mserver).trim().equals ( "" ) ) {
```

Custom Applications 7-31
A message is then sent to the e-mail address in the customer’s profile from the **From:** e-mail address set in the mail_admin.jsp page.

mailorder.xml:

```xml
<ie:webject name="Send-Mail" type="MSG">
  <ie:param name="FROM" data="$\{mailSettings[MAIL_FROM[0]]\}"/>
  <ie:param name="REPLY_TO" data="$\{mailSettings[REPLY_TO[0]]\}"/>
  <ie:param name="TO" data="$\{customerInfo[EMAIL[0]]\}"/>
  <ie:param name="MAIL_SERVER" data="$\{mailSettings[MAIL_SERVER[0]]\}"/>
  <ie:param name="USERNAME" data="$\{mailSettings[USERNAME[0]]\}"/>
  <ie:param name="PASSWORD" data="$\{mailSettings[PASSWORD[0]]\}"/>
  <ie:param name="SUBJECT" data="Info*Engine dot com order # \$
    \{shoppingBasket[ORDERNUM[0]]\}"/>
  <ie:param name="CONTENT" data="Your order has been received and will be
    processed promptly.\n    Order Number: \n    \{shoppingBasket[ORDERNUM[0]]\}\n    Save this information for later use.\n    Thank You for visiting Info*Engine.books.com"/>
</ie:webject>
```

**Note:** In the content of the e-mail message carriage return line feeds are inserted using the SGML escape syntax "\&#13;&#10;". By placing the Java escape character sequences directly in the data parameter, like "\r\n", the backslash is interpreted literally and the four characters "\r\n" are then displayed in the e-mail content. In other cases you could specify the string with Java escape sequences as a variable, and then include the variable as a parameter value, for example `<%=variable%>`. However to take advantage of Info*Engine substitution, this example does not use this method.

The following picture shows the successful order screen.
Once the order is successfully placed, the shopping basket is emptied. This is accomplished by removing the "shoppingBasket" group from the VDB.

basket.jsp:

<ie:getService varName="ieObj"/>

...

ieObj.removeGroup ( "shoppingBasket" );

The ieObj Java variable references the Info*Engine service object. It was retrieved using the getService tag.

Since the customer is still signed into the bookstore, the customer profile remains in the VDB and can continue to be used. The following picture shows the empty shopping basket with the customer profile still active.
Sign Out of Bookstore

After successfully placing the order, Mr. Jones signs out of the bookstore.

Clicking the **Sign Out** link on the **Browse** box makes a request of index.jsp. The "customerInfo" and "shoppingBasket" groups are removed from the VDB, and the browser is returned to the opening bookstore screen.

```
basket.jsp:
if ( request.getParameter ( "signout" ) != null ) {
   ieObj.removeGroup ( "customerInfo" );
   ieObj.removeGroup ( "shoppingBasket" );
}
```

**Returning Customer Scenario**

Happy with his first experience at Info*Engine.books.com, Mr. Jones comes back to find reading material for an upcoming trip. He picks a title from the **Info*Engine Best Sellers List**, and adds this book to his shopping basket. He signs in using his e-mail address.

Not content with just one title, Mr. Jones browses through the various genres, and adds additional books to his basket. Upon viewing his shopping basket, he decides that he does not want to purchase one title, but wants an extra copy of a different title. He changes the quantities appropriately, and updates his shopping basket. Then, he updates his customer profile to reflect his vacation address. Mr. Jones places his order. The bookstore confirms the order and sends a confirmation e-mail.

**Survey of Included Topics**

The following topics will be discussed in addition to topics discussed in the first scenario:

-  `best_sellers.jsp`
-  `custom webobjects`
Return Visit to Bookstore

Happy with his first experience at Info*Engine.books.com, Mr. Jones comes back to find reading material for an upcoming trip. He picks a title from the Info*Engine Best Sellers List, and adds this book to his shopping basket.

The following picture shows the opening bookstore screen.

As discussed previously, the **Info*Engine Best Sellers List**, displayed by the best_sellers.jsp page, is included in the index.jsp page using the jsp:include tag.

From the best_sellers.jsp page, the bestsellers.xml task executes to display any best-selling books.

best_sellers.jsp:

```
<ie:task uri="examples/bookstore/bestsellers.xml"/>%>
  if ( ieObj.getGroup ( "bestSellers" ).getElementCount() > 0 ) { %>
      <h3 color=navy><ie:displayResource properties="examples/bookstore/BookstoreResource" key="BEST_SELLERS"/></h3>
      <ie:forEach groupIn="bestSellers" groupOut="bestSeller">
        The bestsellers.xml task compiles the five best selling books based on orders placed. The BOOKORDER table is queried using a Do-SQL webject. A VDB group is created named "orders" containing ISBN and quantity for each book ordered.
The sales are then compiled using the ISBN.

```xml
<%  Hashtable sales = new Hashtable();  String isbn;  Integer quantity,total;%>
<ie:forEach groupIn="orders" groupOut="order">
  <%  isbn = getGroup ( "order" ).getAttributeValue ( 0, "ISBN" ).toString();  quantity = new Integer ( getGroup ( "order" ).getAttributeValue ( 0, "QUANTITY" ).toString() );  total = (Integer)sales.get ( isbn );  if ( total == null )  sales.put ( isbn, quantity );  else {  total = new Integer ( quantity.intValue() + total.intValue() );  sales.put ( isbn, total );  }
%>
</ie:forEach>
```

The compiled information is placed in a VDB group named "bestsellers" using a Create-Group webject.

```xml
<ie:webject name="Create-Group" type="GRP">  <ie:param name="GROUP_OUT" data="bestSellers"/>
  <%  for ( Enumeration keys = sales.keys();  keys.hasMoreElements () ; ) {  String key = (String)keys.nextElement();  String elem = "ISBN="+key+";QUANTITY="+sales.get(key);  %><ie:param name="ELEMENT" data="<%=elem%>"/></%  }
%>
</ie:webject>
```
The group is sorted using the Sort-Group webject so that the best sellers are on the top, and all but the top five are deleted from the group.

The Query-Objects JDBC adapter webject is used to retrieve the ISBN, title, and author of the best-selling books from the BOOK database table, creating a VDB group named "book".

A link is displayed for each book on the best-sellers list, using getValue and forEach tags.
If no books have yet been purchased from the bookstore, a localized message displays which in English reads, "Info*Engine.books.com open for business!"

Picking a book from the Info*Engine Best Sellers List uses a simple JDBC query, discussed in the previous scenario. Adding a book to the shopping basket is also discussed in the previous scenario.

Sign In To Bookstore

Mr. Jones signs in using his e-mail address.
The bookstore prompts the returning customer to sign in by entering their e-mail address in the **Sign In** textbox. Signing into the bookstore demonstrates a simple JDBC query used to populate the Info*Engine VDB with the customer's information.

customer.jsp:

<% if ( !signed_in ) { %>
<form action="index.jsp?page=customer" method=POST>
<input type=hidden name=isbn value="<%=isbn%>">
<input type=text name=email value=""></input>
<input type=submit value="<ie:displayResource properties="examples/bookstore/BookstoreResource" key="SIGN_IN"/>">
</form>
<% } %>

Upon clicking the **Sign In** button, a Query-Objects JDBC adapter webject is triggered that looks up the customer profile containing the specified e-mail address. Embedded JSP scriptlets analyze the query results to decide whether or not matching customer information is found in the database.

customer.jsp:

if ( !signed_in && email != null ) {
    String where = "EMAIL='' + email + '';";
    <ie:webject name="Query-Objects" type="OBJ">
        <ie:param name="INSTANCE" data="jdbcAdapter"/>
        <ie:param name="CLASS" data="CUSTOMER"/>
        <ie:param name="WHERE" data="<%=where%>"/>
        <ie:param name="GROUP_OUT" data="customerInfo"/>
    </ie:webject>
    com.infoengine.object.factory.Group cinfo = ieObj.getGroup ( "customerInfo" );
    if ( cinfo.getElementCount() == 0 )
        ieObj.removeGroup ( "customerInfo" );

If no customer profile with the supplied e-mail address is found, then a "Sign In Failed!" message is displayed.

customer.jsp:

<%if ( email != null ) { %>
    <font class=warning><ie:displayResource properties="examples/bookstore/BookstoreResource" key="SIGN_IN_FAILED"/></font>
<% } %>

If the customer profile is found then the sign in is successful and the browser window is then redirected to the shopping basket screen.

customer.jsp:

<jsp:include page="basket.jsp" flush="true"/>
    return;
Browse By Genre

Not content with just one title, Mr. Jones browses through the various genres, and adds additional books to his basket.

The **Browse** box is displayed by the index.jsp page.

**index.jsp:**

```html
<table width=100%>
<tr>
    <td width=20% valign=top bgcolor=#FFFFFF align=left>
        <ie:webject name="Query-Objects" type="OBJ">
            <ie:param name="INSTANCE" data="jdbcAdapter"/>
            <ie:param name="CLASS" data="GENRE"/>
            <ie:param name="WHERE" data="()"/>
            <ie:param name="GROUP_OUT" data="genres"/>
        </ie:webject>
    </td>
    <table border=0 cellpadding=0 cellspacing=0 bgcolor=#FF8400>
        <tr valign=bottom align=center width=171 height=19>
            <td><img src=img/browse.jpg></td>
        </tr>
        <tr valign=top align=center width=171>
            <td>
                <table><tr><td>
                    <ie:forEach groupIn="genres" groupOut="genre">
                        <li><nobr><font color=#003399><a href="index.jsp?page=search&gid=<ie:getValue name="GID">" title="""" + ie:getValue name="GENRE"">"" + ie:getValue name="GENRE_TITLE"></a></font></nobr>
                    </ie:forEach>
                </td></tr></table>
            </td>
        </tr>
    </table>
</table>
```

The Query-Objects JDBC adapter webject queries the GENRE database table, and produces a VDB group named "genres". The genre names are then displayed using forEach and getValue custom tags.

When the customer clicks on a genre link, a request is made of search.jsp, along with the genre ID (gid) of the selected genre. The gid corresponds to the GID column in the BOOK database table and can be used to build an SQL query.

**search.jsp:**

```java
String whereClause;
if ( gid != null )
    whereClause = "SELECT * FROM BOOK WHERE GID='" + gid + '"';
```
The Do-SQL webject is used to perform the query.

search.jsp:

```xml
<ie:webject name="Do-SQL" type="ACT">
  <ie:param name="INSTANCE" data="jdbcAdapter"/>
  <ie:param name="SQL" data="<%=whereClause%>"/>
  <ie:param name="GROUP_OUT" data="books"/>
</ie:webject>
```

The rest of the process is carried out in the same manner as the search by author discussed in the previous scenario.

Adding additional books to the shopping basket follows the same process outlined in the previous scenario. A book is selected from a search result set, and added to the shopping basket by clicking on the Add To Basket icon. Because the user is already signed in to the bookstore, the Add To Basket icon already links to the basket.jsp page.
View and Update Shopping Basket Contents

Upon viewing his shopping basket, Mr. Jones decides that he does not want to purchase one title, but wants an extra copy of a different title. He changes the quantities appropriately, and updates his shopping basket.

To update the quantity of books on an order, the customer changes the number in the Quantity box to the desired amount, and clicks the Update Basket button. The Update-Basket custom webject is used to update the order. The code for this custom webject can be found in the following location:

```
<ie_dir>/prog_examples/bookstore/CustomWebobjects.java
```
For more information on custom webobjects see the Advanced User Topics chapter in this guide.

```java
for ( Enumeration pnames = request.getParameterNames();
     pnames.hasMoreElements(); ) { 
    String pname = (String)pnames.nextElement();
    if ( pname.startsWith ( "isbn_" ) ) { 
        <ie:webject name="Update-Basket" type="EXT"
                     use="examples.bookstore.CustomWebjects">
            <ie:param name="GROUP_IN" data="shoppingBasket"/>
            <ie:param name="ISBN" data="\$=pname.substring ( 5 )\""/>
            <ie:param name="QUANTITY" data="\$=request.getParameter ( pname )\""/>
        </ie:webject>
    }
}
```

The database is then checked to see if the ordered items are in the inventory.

```java
if ( !order && ( isbn != null || update ) )
    <ie:forEach groupIn="shoppingBasket" groupOut="basketItem"><%
    String tisbn = ieObj.getAttributeValue ( "basketItem", 0, "ISBN" );
    String where = "ISBN='" + tisbn + "';
    int quantity = Integer.parseInt ( ieObj.getAttributeValue
                                       { "basketItem", 0, "QUANTITY" } );
```

The inventory is then checked to see if there is sufficient quantity in stock to supply the amount ordered.

```java
<ie:webject name="Query-Objects" type="OBJ">
    <ie:param name="INSTANCE" data="jdbcAdapter"/>
    <ie:param name="CLASS" data="INVENTORY"/>
    <ie:param name="WHERE" data="\$=where\""/>
    <ie:param name="GROUP_OUT" data="itemInventory"/>
</ie:webject>
```

```java
    int count = Integer.parseInt ( ieObj.getAttributeValue
                                   { "itemInventory", 0, "QUANTITY" } );
```
If the inventory is not sufficient to supply the quantity specified, a message is displayed, and the order quantity is updated to the remaining inventory amount.

```java
if (quantity > count) {
    RbInfoResourceBundle rbinfo = new RbInfoResourceBundle
        ("examples/bookstore/BookstoreResource");
    warning += rbinfo.getLocalizedMessage("INSUFFICIENT_INVENTORY",
        new Object[] { ieObj.getAttributeValue
            ("basketItem", 0, "TITLE") }) + "<br>
    String scount = "+"+count;

    <ie:webject name="Update-Basket" type="EXT"
        use="examples.bookstore.CustomWebobjects">
        <ie:param name="GROUP_IN" data="shoppingBasket"/>
        <ie:param name="ISBN" data="<%=tisbn%>"/>
        <ie:param name="QUANTITY" data="<%=scount%>"/>
    </ie:webject>
</ie:forEach>
```

The following picture displays the updated shopping basket.

---

**Customer Information:**

William Jones  
456 Main Street  
Anytown, MN 55555  
jones@anytown.com

© Parametric Technology Corporation 2001, all rights reserved.
Update Customer Profile

Mr. Jones updates his customer profile to reflect his vacation address.

Clicking on the Edit link in the Browse box causes the browser to display the Customer Information screen.

The following picture displays the Customer Information screen with the current customer profile.
After making any necessary changes to the customer profile, the customer clicks the **Update** button, causing the `updatecustomer.xml` task to execute. The process continues in the same manner as creating a new customer profile, discussed in the previous scenario.

**Place Order, Receive Order Confirmation E-Mail**

Mr. Jones places his order. The bookstore confirms the order and sends an confirmation e-mail.

The order process is the same for this scenario as for the first scenario, with one exception. If the customer changes the order quantities and clicks the **Place Order** button without first updating the basket, and if the quantity specified for an item exceeds the inventory amount, then the `placeorder.xml` task throws an exception.

```xml
placeorder.xml:
<ie:webject name="Throw-Exception" type="MGT">
  <ie:param name="CLASS" data="examples.bookstore.InsufficientInventoryException"/>
  <ie:param name="MESSAGE" data="<%=message%>"/>
</ie:webject>
```

The exception is caught and re-thrown by the task failure block.

```xml
placeorder.xml:
<ie:failure exception="examples.bookstore.InsufficientInventoryException">
  <ie:webject name="Throw-Exception" type="MGT"/>
</ie:failure>
```

The exception is caught by the `basket.jsp` page.

```jsp
basket.jsp:
if ( order ) {
  warning = null;
  try {
    <ie:task uri="examples/bookstore/placeorder.xml">
      <ie:param name="GROUP_IN" data="shoppingBasket"/>
      <ie:param name="GROUP_IN" data="customerInfo"/>
    </ie:task>
  } catch ( examples.bookstore.InsufficientInventoryException iie ) {
    warning = iie.getMessage();
  }
}
```

On screen confirmation and notification by e-mail happen in the same manner as outlined in the previous scenario.
Restocking the Bookstore

After utilizing the bookstore example for a period of time, you may discover that book inventories are dwindling. The inventory.jsp page allows you to manually restock the shelves of the bookstore.

The inventory.jsp page can be reached by entering a URL into your browser in the following form:

http://<hostname>/Windchill/infoengine/jsp/examples/bookstore/inventory.jsp

where <hostname> is the name of the server where Info*Engine resides. If you have configured your Info*Engine installation differently, then you will have to determine the URL based on your customized configuration.

![Inventory OK](image)

If the inventory for any books has an inventory of fifty or less, they will be displayed, allowing you to manually update the inventory quantities. The concepts involved have been discussed in previous sections of this example. For further details, please view the code for inventory.jsp.

MOM Topics

A MOM is used in two ways in the bookstore example: queues and events.

Queues

Under normal circumstances the contents of a user’s shopping basket will only be available as long as his or her HTTP session is valid. If the user opens a new Web browser, and logs in, their shopping basket will be empty. However, if a MOM is available, a queue has been created for the bookstore and Info*Engine has been configured to communicate with that MOM the contents of a user’s shopping basket can be persisted across multiple HTTP sessions in a queue. Info*Engine server MSG webjects are used to store and retrieve the contents of the session VDB in a MOM queue.
Events

As discussed in the second scenario, if ordering a book causes the book inventory to drop below a specified threshold, the placeorder.xml task will emit a bookReorder event. If the user clicks the **Automatically Re-Stock Books** button, the listen4restock.xml task is triggered, subscribing them to the bookReorder event. As a result, each time a bookReorder event occurs the restock.xml task will be executed causing the book’s inventory to be reset to 100 copies.

**listen4restock.xml:**

```xml
<ie:webject name="Subscribe-Event" type="WES">
  <ie:param name="EVENT" data="bookReorder"/>
  <ie:param name="EXECUTE_TASK" data="examples/bookstore/restock.xml"/>
  <ie:param name="GROUP_OUT" data="subscribeResults"/>
</ie:webject>

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

The restock.xml task automatically updates the inventory quantity of any book whose inventory has fallen below the threshold to 100 using a Do-SQL webject. The MAIL_SETTINGS database table is then queried to see if mail settings have been set. If mail settings are found, then an e-mail is sent using the Send-Mail webject to the e-mail address specified in the **From** box of the mail_admin.jsp page indicating that the book has been restocked.

**restock.xml:**

```xml
<ie:webject name="Send-Mail" type="MSG">
  <ie:param name="FROM" data="${mailSettings[MAIL_FROM][0]}"/>
  <ie:param name="REPLY_TO" data="${mailSettings[REPLY_TO][0]}"/>
  <ie:param name="TO" data="${mailSettings[MAIL_FROM][0]}"/>
  <ie:param name="MAIL_SERVER" data="${mailSettings[MAIL_SERVER][0]}"/>
  <ie:param name="USERNAME" data="${mailSettings[USERNAME][0]}"/>
  <ie:param name="PASSWORD" data="${mailSettings[PASSWORD][0]}"/>
  <ie:param name="SUBJECT" data="Info*Engine dot com Re-Stock Event"/>
  <ie:param name="CONTENT" data="The book with ISBN {BOOK[ISBN][0]} has been re-stocked."/>
</ie:webject>
```
**Info*Engine J2EE Book Viewer Example**

The Info*Engine J2EE book viewer provides an example of a J2EE SOAP client which uses the Info*Engine J2EE connector to interact with Info*Engine.

The book viewer example files can be found in the following location:

```
install/ieconnector/examples/bookstore.jar
```

where `install` is the Info*Engine installation directory.

The README.html files contained in the bookstore.jar provide requirements information and instructions for setting up the book viewer application. Refer to the README.html file in the appropriate directory for your application server environment (JBoss or Sun ONE).

**Requirements**

The following are required for the J2EE book viewer example to be successfully run:

- Windchill Info*Engine 7.0
- Info*Engine J2EE connector (ie.rar), located at `install/ieconnector/ie/rar`, where `install` is the Info*Engine installation directory.
- An application server environment:
  - JBoss 3.0.8, integrated with Jetty. For additional information on JBoss, see [http://www.jboss.org](http://www.jboss.org).
  - or
  - Sun ONE 7.0
- JDK 1.4 or higher
- Jakarta Ant 1.4.1

For additional information on Jakarta Ant, see [http://ant.apache.org/index.html](http://ant.apache.org/index.html)

- XDoclet 1.1.2

For additional information on xdoclet, see [http://sourceforge.net/projects/xdoclet/](http://sourceforge.net/projects/xdoclet/)

**Note:** The J2EE book viewer example utilizes the database installed for the Info*Engine bookstore example. For further information on setting up the bookstore example, see the *Info*Engine User’s Guide.
This chapter contains reference topics for the tags contained in the Info*Engine core and directory tag libraries. It also contains an overview of common JSP element types with tips for using the elements on Info*Engine JSP pages.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Overview</td>
<td>8-2</td>
</tr>
<tr>
<td>Scriptlets</td>
<td>8-3</td>
</tr>
<tr>
<td>Expressions</td>
<td>8-5</td>
</tr>
<tr>
<td>Declarations</td>
<td>8-6</td>
</tr>
<tr>
<td>Directives</td>
<td>8-7</td>
</tr>
<tr>
<td>Info*Engine Tags</td>
<td>8-10</td>
</tr>
</tbody>
</table>
Usage Overview

The elements described in this chapter can be used when creating Info*Engine JSP pages and Info*Engine standalone tasks.

In addition to these elements, Info*Engine JSP page authors can use any of the features available through JSP on pages that access Info*Engine. To learn about these features, you can access information on the Web or in the many books written on the subject.

The following element types are described in this chapter:

- Scriptlets
- Expressions
- Declarations
- Directives
- Info*Engine Tags

For information on how to create Info*Engine JSP pages and Info*Engine standalone tasks, see the Authoring Info*Engine JavaServer Pages and Authoring Info*Engine Tasks sections.
Scriptlets

A scriptlet is a standard JavaServer Page 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 (using if/then/else), iteration (using for/while), and nontrivial computation.

Tasks can include embedded Java scriptlets (<% … %> ). However, you must be careful when positioning scriptlets in a task if the task will be 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 before the first webject or task tag or after the last one.

– 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 at the time when Info*Engine encounters the code instead of being 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 in which 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 imbed 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 the param tag, you can use expressions in only the data attribute.

**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:

```html
<ie:param name="ELEMENT" data="ENAME=<%=ENAME%>"/>
```

Instead, you can code this as follows:

```html
<% String elValue = "ENAME=" + ENAME; %>
<ie:param name="ELEMENT" data="<%=elValue%>"/>
```

**Tag Syntax**

```html
<%= expression %>
```

**Example**

In the following paragraph, two expressions are used to insert the java variables table and where:

```html
<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 outer-most scope of the Java program it generates for the task. Therefore, the declarations become global definitions in the program. 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 declares that the page or task uses a tag library, uniquely identifies the tag library using a URI, and associates a tag prefix that will distinguish 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.

Standalone task use:

The task compiler recognizes a standard page directive, but uses only the import attribute for importing class files that are needed by the task.

JSP use:

Use this directive to specify the characteristics of the page. In particular, you identify the Java language used in scripts on the page, specify the page scope, import any class files or packages needed for the page, determine what page is displayed for errors, and identify the page as the error page.

Tag Syntax for Common Page Characteristics

```%@ page language="java" 
  session="boolean"
  import="list_of_classes"
  errorPage="error_page_url"
  isErrorPage="boolean" %>
  ENABLERECOVERY=[true|on]
```

Attribute Descriptions

Required attributes: language

ENABLERECOVERY

Controls the generation of guaranteed task recovery code. To generate task recovery code, set this attribute to "true". By default, recovery code is not generated.
**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 on 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 session, imports the Info*Engine factory and object packages, and sets the error page to "IEError.jsp":

```html
<%@page language="java" session="true"
    import="com.infoengine.object.factory.*, com.infoengine.object.*"
    errorPage="IEError.jsp"%>
```

For additional information about error pages, see the section titled Catching Exceptions.
**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:** *uri* and *prefix*

  - **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 "ie_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 will contain custom tags from the Info*Engine* core tag library and that the tags will 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 Info*Engine 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

In standalone tasks, you can only use the tags from the core tag library. The tags in the directory tag library are not supported.

To use the Info*Engine tags, you 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:

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

The following table provide a summary of the Info*Engine tags that are in the core library.

<table>
<thead>
<tr>
<th>Core Library Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prevents anonymous access to the JSP page containing the tag and optionally allows you to provide an HTTP protocol challenge and an authentication task for authenticating users who access the JSP page.</td>
</tr>
<tr>
<td></td>
<td>Extracts a string from a group created with the Get-Resource webject or directly from a resource bundle and displays it.</td>
</tr>
<tr>
<td></td>
<td>Allows you to supply code for failure processing within a unit.</td>
</tr>
<tr>
<td></td>
<td>Allows you to iterate through an existing Info*Engine group, one element at a time.</td>
</tr>
<tr>
<td></td>
<td>Establishes a variable reference to the Info<em>Engine object (com.infoengine.jsp.InfoEngine) that is being used by the Info</em>Engine custom tags on a JSP page.</td>
</tr>
<tr>
<td></td>
<td>Retrieves the string value of the specified attribute from the first element in the input group.</td>
</tr>
</tbody>
</table>
The following table provides a summary of the Info*Engine tags that are in the directory library.

### Core Library Tag

<table>
<thead>
<tr>
<th>Core Library Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Supplies the initialization for a unit.</td>
</tr>
<tr>
<td></td>
<td>Allows you to define a set of webobjects or tasks that are executed concurrently.</td>
</tr>
<tr>
<td></td>
<td>Defines a parameter for use within <strong>webobject</strong> tags, <strong>task</strong> tags and directory service tags.</td>
</tr>
<tr>
<td></td>
<td>Resets the Info<em>Engine object (com.infoengine.jsp.InfoEngine) that is being used by the Info</em>Engine custom tags on a JSP page or in a session.</td>
</tr>
<tr>
<td></td>
<td>Allows you to supply code for success processing within a unit.</td>
</tr>
<tr>
<td></td>
<td>Identifies an XML task that you want to execute.</td>
</tr>
<tr>
<td></td>
<td>Allows you to group a sequence of webobjects, tasks, or Java code so that the group is executed as a unit.</td>
</tr>
<tr>
<td></td>
<td>Identifies the webobject you want to execute.</td>
</tr>
</tbody>
</table>

The following table provides a summary of the Info*Engine tags that are in the directory library.

### Directory Library Tag

<table>
<thead>
<tr>
<th>Directory Library Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates new entries in the specified LDAP directory service.</td>
</tr>
<tr>
<td></td>
<td>Deletes an LDAP entry from the LDAP directory service.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Creates an Info*Engine group containing elements that are the LDAP directory entries matching the search criteria specified.</td>
</tr>
<tr>
<td></td>
<td>Updates existing entries in an LDAP directory service</td>
</tr>
</tbody>
</table>

---

Info*Engine Custom Tag Reference  8-11
The following section describes some rules you need to be familiar with. Then you will find a section describing each tag in detail. The syntax for each tag assumes that the `taglib prefix` attribute is set to "ie" for tags in the Info*Engine core tag library and set to "iedir" for tags in the Info*Engine directory tag library.

**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 in order 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"/>
  
  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, 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 can not 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'/>
• Five 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.

<table>
<thead>
<tr>
<th>Symbol Name</th>
<th>Appearance</th>
<th>Character Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>ampersand</td>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>apostrophe</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

• Info*Engine maintains the @FORM, @SERVER, and @COOKIE context groups as part of the VDB. Therefore, you can not 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 the Authentication through Credentials Mapping section.
authenticate Tag

The authenticate tag prevents anonymous access to the JSP page containing the tag and optionally allows you to provide an HTTP protocol challenge and an authentication task for authenticating users who access the JSP page. Additionally, you can specify which task processor executes the authentication task.

When you include an authentication task as an attribute on this tag, ensure that the task returns the following:

- For a successful validation, either return a status group that has a status of 0 or return a non-empty data group with a status value of 0.
- For a failed validation, either return an empty data group or return any type of group that has a status value that is not 0.

If the task validation fails, the authentication tag returns an authentication challenge.

If you use Web server authentication and include this tag on a JSP page, Info*Engine creates an authorization element in the SERVER context group. The content of this element comes directly from the HTTP protocol request provided by the Web server. It includes the authentication method used and any parameters associated with the authentication (for example, a valid user name and password).

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to 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.

Tag Usage

Only use this tag on JSP pages; do not use it in standalone Info*Engine tasks.

This tag cannot be embedded within other Info*Engine tags.

Tag Syntax

```<ie:authenticate challenge="protocol_challenge"
  task="task_name" processor="task_processor"/>```
Attribute Descriptions

challenge

Specifies the HTTP protocol challenge that is returned to the client when a connection needs authentication parameters (such as user name and password) or when the authentication parameters supplied fail.

This attribute is the full challenge string that is returned in the authorization header. The challenge string is returned in an HTTP protocol "WWW-Authenticate" header, so its syntax must conform to the requirements for that header, as defined in the Internet protocol specification for HTTP.

If you omit this attribute, the challenge string stored in the Info*Engine servlet .jsp.taglibs.authenticationChallenge property is used. If this property is not set, the following default challenge is constructed:

Basic realm="hostDomain"

where hostDomain is the Internet domain name of the host on which the JSP page is executing.

processor

Specifies the name of a remote Info*Engine task processor to which the authentication task is sent for execution. The name you specify must map to a task processor that is available from your current environment. The name you specify in this attribute is the service name defined for the task processor through the Info*Engine Property Administrator or during installation.

This attribute is optional. If omitted, the name of the remote task processor is obtained from the Info*Engine Servlet .jsp.taglibs.authenticationProcessor property. If this property is not defined and the processor attribute is not specified, the authentication task is executed locally within the JSP page itself.

To direct the authentication 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 or you must set the Info*Engine Servlet .jsp.taglibs.authenticationProcessor property. For example, if the task processor has the default name of "com.myCompany.ieServer.taskProcessor" you can include the following processor attribute:

processor="com.myCompany.ieServer.taskProcessor"
task

Specifies a URI that is the location of the XML task file to execute for authentication. 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 optional. If omitted, the URI of the task is obtained from the Info*Engine Servlet .jsp.taglibs.authenticationTask property. If this property is not defined and the task attribute is not specified, no authentication task is executed. In this case, any user name provided by the Web server is accepted. However, if the Web server does not provide a user name, Info*Engine returns an authentication challenge. Therefore, anonymous access is not allowed.

For example URI locations, see the section titled Specifying URIs and URLs.

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.

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 authenticate tag specifies that authentication be done in the "authenticate.xml" task that is located in the root file directory of the local task processor:

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

<ie:authenticate task="authenticate.xml"/>
```
createObjects Tag

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn</td>
<td>The distinguished name of the entry, where you specify this name starting with dn=ptcServiceName=ServiceName,additional attributes.</td>
</tr>
<tr>
<td>objectClass</td>
<td>The object class to associate with the entry. Determine valid values from the directory schema configured for the LDAP server.</td>
</tr>
</tbody>
</table>

You can include additional attributes in each element, if they are required by the object class.

You name the Info*Engine group using the GROUP_IN parameter on a nested `param` tag. For additional information about this tag, see the section titled param Tag.

Tag Library

This tag is included in the Info*Engine directory tag library. To use this tag, you must include a taglib directive similar to the following directive:

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

The syntax for the tag 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.

Tag Usage

You specify the required parameter by nesting the `param` tag in this tag block.

**Note:** This tag can only be used in JSP pages; it is not recognized in standalone tasks.

Tag Syntax

```xml
<iedir:createObjects uri="service_URL">
  <iedir:param name="GROUP_IN" data="group_name" />
</iedir:createObjects>
```
Attribute Description

**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 the "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:

```xml
<%@ 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 Tag**

The *deleteObject* tag deletes one LDAP leaf entry from the LDAP directory service.

**Tag Library**

This tag is included in the Info*Engine directory tag library. To use this tag, you must include a taglib directive similar to the following directive:

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

The syntax for the tag 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.

**Tag Usage**

**Note:** This tag can only be used in JSP pages; it is not recognized in standalone tasks.

**Tag Syntax**

```html
<iedir:deleteObject uri="entry_URL"/>
```

**Attribute Description**

**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.jAdapter" entry from the directory service located at "myCompany.com":

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

The `displayResource` tag extracts a string from a group created with the Get-Resource webobject or directly from a resource bundle. The tag then replaces one variable placeholder in the extracted string (if you supply replacement text) and displays resulting string.

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

Tag Usage

This tags is provided as an alternative to using the Display-Resource webobject. You can use this tag in conjunction with the Get-Resource webobject.

Note: Tags do not support multi-valued 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 webobject.

Tag Syntax

```html
<ie:displayResource bundle="bundle_name"
    groupIn="group_name"
    key="bundle_key"
    param="text"/>
```

Attribute Descriptions

- **bundle**: Indicates the java class resource bundle from which the localized string will be 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 a 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 webject in the example specifies that the messages associated with line 19 of the resource bundle retrieved using the Get-Resource webject be displayed:

```xml
<%@ 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 Tag

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.

To manage exceptions within a **failure** tag block, you can include the Throw-Exception webject. Including this webject is a way to propagate (re-throw) exceptions that are caught by **failure** tags. You can simply add the following in the block:

```xml
<ie:webject name="Throw-Exception" type="MGT"/>
```

Including this webject causes the caught exception to be re-thrown. Re-throwing 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.

**Note:** 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 the section entitled Scriptlets.

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```xml
<%@ 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.

Tag Usage

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.

Tag 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 Tag

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 in the output group named in the **groupOut** attribute and processing continues on to the next line after the **forEach** end tag.

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```xml
<%@ 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.

Tag Usage

This tag cannot be nested in other Info*Engine tags.

Other Info*Engine tags can be nested under this tag.

Tag Syntax

```xml
<ie:forEach groupIn="group_name" groupOut="group_name">  
  .  
  .  (other Info*Engine tag blocks)  
  .  
</ie:forEach>
```
Attribute Descriptions

**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 each element using the caption "One Employee".

```xml
<%@ 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 Tag**

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 webobject 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 user name 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.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to 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.
Tag Usage

Only use this tag on JSP pages; do not use it in standalone Info*Engine tasks.

**Note:** Each page can only have one `getService` tag on it.

This tag cannot be embedded within other Info*Engine tags.

Tag Syntax

```
<ie:getService varName="variable_name"/>
```

Attribute Descriptions

**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()%></b> employees.</p>
</body>
</html>
```
**getValue Tag**

The `getValue` tag retrieves the string value of the specified attribute from the first element in the input group.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```xml
<%@ 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.

**Tag Usage**

This tag can be nested in other Info*Engine tags.

Do not imbed this tag in scriptlets; doing so will cause a compiler error because the tag is read as plain text. For example, the following scriptlet does not compile:

```jsp
<% 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.

**Tag Syntax**

```xml
<ie:getValue name="attr_name" groupIn="group_name"/>
```

**Attribute Descriptions**

- **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.

```html
<%@ 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.

```html
<%@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"/>
  <b>employee name:</b><ie:getValue name="ENAME"/>
  <b>salary:</b>$<ie:getValue name="SAL"/>
  <hr>
  <%
    String ssal = pie.getAttributeValue ( "one-element", 0, "SAL" );
    tot_sal += Float.parseFloat ( ( (ssal != null && !ssal.equals("")) ? ssal : "0" ) );
  %>
</ie:forEach>
```
<b>Salary Total:</b>$<%=tot_sal%><br>

**init Tag**

The **init** tag supplies the initialization for a unit. The code between the start and end **init** tag 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 will be 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:

```html
<!-- 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:unit>
  <ie:failure>
    <!-- First Unit failure processing accessed when -->
    <!-- there is an error in the body of the first unit-->
  </ie:failure>
</ie:unit>
```
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.

**Note:** 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 the section entitled Scriptlets.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

**Tag Usage**

You nest this tag in `unit` tag blocks.

You can nest multiple `webject`, `task`, `unit`, and `parallel` tags in this tag block.

**Tag Syntax**

```html
<ie:init>
  .
  .  (webject, task, unit, or parallel tag blocks)
  .
</ie:init>
```
listObjects Tag

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 additional information about this tag, see the section titled param Tag.

Tag Library

This tag is included in the Info*Engine directory tag library. To use this tag, you must include a taglib directive similar to the following directive:

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

The syntax for the tag 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.

Tag Usage

You specify the required parameter by nesting the param tag in this tag block.

Note: This tag can only be used in JSP pages; it is not recognized in standalone tasks.

Tag Syntax

```<iedir:listObjects uri="base_level_URL">  
<iedir:param name="GROUP_OUT" data="group_name"  
<iedir:listObjects>
```

Attribute Description

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:

```xml
<%@ 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>
```
**parallel Tag**

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 the VDBs of the individual processes are merged with the VDB in use by the page or task, and 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.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

**Tag Usage**

You can nest this tag in unit, init, success, failure tag blocks.

You can nest multiple webject and task tags in this tag block.

**Note:** Display webjects cannot be nested in a parallel tag block.

**Tag Syntax**

```html
<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:

```html
<%@ 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 Tag

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 information relating to their use, see the section entitled Expressions.

Tag Library

The param tag is included in the Info*Engine core and directory tag libraries. To use this tag, you must include a taglib directive similar to one of the following directives:

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

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.

Tag Usage

You can nest this tag in the webject, task, createObjects, listObjects, queryObjects, and updateObjects tags.

Tag Syntax

```xml
<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.

This attribute is optional. However, 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="*"/>
```
**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:

\[
data=\"value1,value2,\ldots,valueN\"\ delim=\","\]

The following *param* tag example has three values defined in the *data* attribute and uses the comma as the delimiter:

```xml
<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:

```xml
<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.

**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.

This is an optional attribute. By default, Info*Engine uses the semicolon as the element separator. For more information about defining substitution expressions, see the section titled Dynamic Parameter Value Substitution.

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 "#":

```xml
<ie:param name="ATTRIBUTE" data="${grp1[*]attr[0]}" elementSeparator="#"/>
```

**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.

This is an optional attribute. By default, Info*Engine uses the comma as the value separator. For more information about defining substitution expressions, see the section titled Dynamic Parameter Value Substitution.

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 "|":

```xml
<ie:param name="ATTRIBUTE" data="\${grp1[0]attr[*]}" valueSeparator="|"/>
```

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:

```xml
<%@ 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>
```
queryObjects Tag

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn</td>
<td>The distinguished name of the entry.</td>
</tr>
<tr>
<td>objectClass</td>
<td>The object class to associate with the entry.</td>
</tr>
</tbody>
</table>

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 additional information about this tag, see the section titled `param` Tag.

Tag Library

This tag is included in the Info*Engine directory tag library. To use this tag, you must include a taglib directive similar to the following directive:

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

The syntax for the tag 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.

Tag Usage

You specify the required parameter by nesting the `param` tag in this tag block.

**Note:** This tag can only be used in JSP pages; it is not recognized in standalone tasks.

Tag Syntax

```html
<iedir:queryObjects uri="query_URL">
  <iedir:param name="GROUP_OUT" data="group_name">
  <iedir:queryObjects>
  </iedir:queryObjects>
</iedir:param>
</iedir:queryObjects>
```
**Attribute Description**

**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 the section titled Specifying URIs and URLs.

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:

```xml
<%@ 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>
```
**resetService Tag**

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 [object](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:

- To avoid 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 having other information to have a session scope.
- To clean 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.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a `taglib` directive similar to the following directive:

```jsp
<%@ 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.

Tag Usage

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 on a getService tag and this tag.

This tag cannot be embedded within other Info*Engine tags.

Tag 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:

```html
<%@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 will still include the EMPLOYEEdata group.
success Tag

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.

**Note:** 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 the section entitled Scriptlets.

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

Tag Usage

You nest this tag in **unit** tag blocks.

You can nest multiple **webject**, **task**, **unit**, and **parallel** tags in this tag block.

Tag Syntax

```html
<ie:success>
  
  (webject, task, unit, or parallel tag blocks)

</ie:success>
```

Example

See the **unit** tag example.
task Tag

The **task** tag identifies an XML task that you want to execute.

**Note:** Using this tag in conjunction with the Info*Engine task processor .secret.text or .secret.text2 and .secret.algorithm properties allows for validation to occur before a remote processor will execute the task. For information about configuring these properties, see the *Info*Engine Installation and Configuration Guide.

Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```xml
<%@ 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.

Tag Usage

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.

Tag Syntax

When there are parameters, you can use the following syntax:

```xml
<ie:task uri="uri_task_source"
    processor="processor1"
    processor="processor2"
    .
    .
    processor="processorn" >
    resumable="[true|false]"
    .
    .
    .
</ie:task>
```

When there are no parameters, you can use the following syntax:

```xml
<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 Administrator.

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.taskProcessor" 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:

```xml
<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 prior to 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 prior to 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.

For example URI locations, see the section titled Specifying URIs and URLs.

**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.
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":

```xml
<%@ 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 Tag

The **unit** tag allows you to group a sequence of webobjects, tasks, or Java code so that the group is executed as a unit. Within the unit, you supply the main body and can also define 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 webobjects, 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, that code then 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 catch through a **failure** tag, the error is not handled.

For additional information about the **init**, **success**, and **failure** tags, see the section that corresponds to the tag in this chapter.

**Note:** 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 the section entitled Scriptlets.
Tag Library

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

Tag Usage

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 wobject, task, unit, and parallel tags in this tag block.

Tag Syntax

```html
<ie:unit>
  
  . (wobject, 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 complete successfully, then the success block executes. If it doesn’t 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>
```
updateObjects Tag

The **updateObjects** tag updates existing entries in an LDAP directory service.

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, which is the distinguished name of the entry. The additional attributes in each element identify names and values of the LDAP attributes to modify.

Using this tag, you can add new attributes to existing LDAP entries, replace existing attributes, or delete existing attributes.

You name the Info*Engine group that contains the LDAP attributes to update using the GROUP_IN parameter on a nested **param** tag.

Tag Library

This tag is included in the Info*Engine directory tag library. To use this tag, you must include a taglib directive similar to the following directive:

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

The syntax for the tag 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.

Tag Usage

You specify the required parameter by nesting the **param** tag in this tag block.

**Note:** This tag can only be used in JSP pages; it is not recognized in standalone tasks.

Tag Syntax

```html
<iedir:updateObjects uri="service_URL" modification="type">
<iedir:param name="GROUP_IN" data="group_name">
<iedir:updateObjects>
```

Attribute Descriptions

**uri**

Specifies an LDAP URL that identifies the directory service and the base entry to use when creating the new entries. For example, entering the following URL identifies the "myCompany.com" directory service and sets the base entry to "myHost.myState.myCompany.com":

```text
ldap://myCompany.com/dc=myHost,dc=myCompany,dc=com
```

This attribute is required.
**modification**

Specifies the type of modification to do. Enter one of the following types:

<table>
<thead>
<tr>
<th>Modification Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADD</strong></td>
<td>Adds the LDAP attributes specified in the group element to the LDAP entry identified by the &quot;dn&quot; 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.</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>Deletes the LDAP attributes other than &quot;dn&quot; that are named in group element attributes for the LDAP entry. Each LDAP entry is identified by the &quot;dn&quot; group element attribute. To delete an entire LDAP entry, use the <code>deleteObject</code> 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.</td>
</tr>
<tr>
<td><strong>REPLACE</strong></td>
<td>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 &quot;dn&quot; 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.</td>
</tr>
</tbody>
</table>

The default for this optional attribute is REPLACE.
Example

The following example declares that the page uses tags from the Info*Engine "directory" and the "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:

```xml
<%@ 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>
```
**webject Tag**

The **webject** tag identifies the webject you want to execute.

On JSP pages, you can name any defined webject in the webject tag.

**Note:** In standalone tasks, display webjects are not allowed.

**Tag Library**

This tag is included in the Info*Engine core tag library. To use this tag, you must include a taglib directive similar to the following directive:

```html
<%@ 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.

**Tag Usage**

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:** Display webjects are not allowed to be embedded within parallel tag blocks.

**Tag Syntax**

When there are parameters, you can use the following syntax:

```xml
<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:

```xml
<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 prior to 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 prior to 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:

<table>
<thead>
<tr>
<th>Type Constant</th>
<th>Webject Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td>Display</td>
</tr>
<tr>
<td>IMG</td>
<td>Image</td>
</tr>
<tr>
<td>ACT</td>
<td>Action</td>
</tr>
<tr>
<td>OBJ</td>
<td>Query</td>
</tr>
<tr>
<td>GRP</td>
<td>Group</td>
</tr>
<tr>
<td>EXT</td>
<td>External</td>
</tr>
<tr>
<td>MGT</td>
<td>Management</td>
</tr>
<tr>
<td>MSG</td>
<td>Messaging</td>
</tr>
<tr>
<td>WES</td>
<td>Web Event Service</td>
</tr>
</tbody>
</table>

This attribute is required.

**Note:** Display webjects (type=DSP) cannot be used in Info*Engine tasks.
Also, 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 file 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:

```html
<%@ 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>
```
This chapter contains information about Info*Engine Display 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.

In tables at the beginning of the parameter descriptions, the parameters are categorized as being either Required, Select, or Optional:

- A parameter is listed in the Required column when it is always required.
- A parameter is listed in the Select column when there is a relationship between the specified parameter and another parameter. 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.

- A parameter is listed in the Optional column when it is always optional and when it is not related to another parameter.

Info*Engine supports HTML, and JPEG formats for display. The display webjects for each language are in separate sections as follows.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Webjects for HTML</td>
<td>9-2</td>
</tr>
<tr>
<td>Image Webjects for JPEG</td>
<td>9-58</td>
</tr>
</tbody>
</table>
Display Webjects for HTML

The following webjects can be used to display output in an HTML format.

Apply-XSL
Display-Object
Display-Resource
Display-Selection
Display-Table
Display-Value
Display-XML
Echo-Request
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="XSLPARAM" data="name_value_pair"/>
   <ie:param name="XSL_URL" data="url_location"/>
</ie:webject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSL_URL</td>
<td></td>
<td>CONTENT_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSL_PARAM</td>
</tr>
</tbody>
</table>

CONTENT_TYPE

Specifies the MIME content type to be associated with the result being displayed. Using 'text/html' as the content type returns an HTML-formatted document to the calling application or Web browser.

The default for this parameter is 'text/xml', which produces an XML-formatted document. This parameter is optional.

DBUSER

Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.
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 user name when connecting to the MOM, and an anonymous connection will be attempted.

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, refer to the chapter titled How Info*Engine Manages Data.

The default for this parameter is to use the last group defined in the VDB. 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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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.

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 de-referenced using Auth-Map context group data. The Auth-Map context group is searched for a user name 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/examples/createGroupData.xml. The Auth-Map context group is searched for a user name and password with http://machine.com as an INSTANCE name. If a user name and password are found, BASIC authentication information is used when accessing the URL. If no user name 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.
EXAMPLES

The following Apply-XSL webobject examples are contained in the ApplyXsl.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/ApplyXsl.jsp

The group displayed in each of these examples is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

Browser Example

Example webobject:

The following Apply-XSL webobject formats the output from a task, then displays the output in a browser:

```xml
<ie:webject name="Apply-XSL" type="DSP">
  <ie:param name="CONTENT_TYPE" data="text/html"/>
  <ie:param name="XSL_URL" data="infoengine/jsp/examples/ApplyXsl.xsl"/>
</ie:webject>
```

XSL stylesheet:

In the example webobject, the following XSL stylesheet is applied to the default output group of the "infoengine/jsp/examples/CreateGroup.xml" task:

```xml
<?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:

<table>
<thead>
<tr>
<th>name</th>
<th>Home Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
</tr>
</tbody>
</table>

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:

```jsp
<%@page language="java" session="false" errorPage="/IEError.jsp" contentType="application/ms-excel"%>
<%@ taglib uri="http://www.ptc.com/infoengine/taglib/core" prefix="ie"%>
<%
/*********************************************
* DESCRIPTION
* *********************************************/
// execute create group task to get group info
%
<ie:task uri="infoengine/examples/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="infoengine/examples/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 from the default output group of the "infoengine/jsp/examples/CreateGroup.xml" task.

The Apply-XSL webject on the JSP page calls the following XSL stylesheet:

```xml
<?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, $EQUALS)"/>
          <xsl:value-of select="substring-before($TEMP1, $DELIM)"/>
        </xsl:if>
        <xsl:if test="not(contains($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 resides in the JSP files directory that is accessed through the "infoengine" application URL prefix on your local host, enter the following:

```
http://localhost/infoengine/jsp/examples/dspwebobjects/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.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Employee</strong></td>
<td>Home Address</td>
<td>EMAIL</td>
</tr>
<tr>
<td>2</td>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>3</td>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:harderson@somewhere.com">harderson@somewhere.com</a></td>
</tr>
<tr>
<td>4</td>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>
**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**

```xml
<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. 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;$\text{namecode}$";"/>
```

```
<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 "").

<table>
<thead>
<tr>
<th>Parameter</th>
<th>HTML Default Value Used with a Border</th>
<th>HTML Default Value Used with No Border</th>
</tr>
</thead>
</table>
| PRE_TABLE_TEXT             | "<table border=BORDER
                cellspacing=CELLSPACING
                cellpadding=CELLPADDING>\n
                "</table>\n" |
| POST_TABLE_TEXT            | ""                                    | ""                                    |
| PRE_HEADER_TEXT            | "<th>"
                ""                                    |
| HEADER_SEPARATOR           | ""                                    | ""                                    |
| POST_HEADER_TEXT           | "</th>"                                | ""                                    |
| PRE_OBJECT_TEXT            | "<tr>"
                ""                                    |
| POST_OBJECT_TEXT           | "</tr>\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, refer to the chapter titled How Info*Engine Manages Data.

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 the Dynamic Parameter Value Substitution section.

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 the Dynamic Parameter Value Substitution section.

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 non-existent 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 are contained in the DisplayObject.jsp file that is located in the Info*Engine infoengine/jsp/examples/ dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/DisplayObject.jsp

The group displayed in each of these examples is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

Default display

Webject:

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

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

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.com
Harvy Anderson, 1234 Amber St. handerson@somewhere.com
James 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:

```xml
<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="&lt;br&gt;"/>
</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 <joco@somewhere.com>
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 = 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvey Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>
**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:**

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:tjohnson@somewhere.com">tjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>
The following example shows how JavaScript can be incorporated into the Display webject.

**Webject:**

```html
<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="&lt;script&gt; itemcnt++;
        if (itemcnt % 2 == 0) document.write('&lt;tr
            bgcolor=#FFFFFF&gt;'); else document.write('&lt;tr
            bgcolor=#D3D3D3&gt;'); &lt;/script&gt;"/>
    &lt;ie:param name="ATTRIBUTE" data="&lt;td width=10&gt;
        &nbsp; &lt;/td&gt;"/&gt;
    &lt;ie:param name="ATTRIBUTE" data="&lt;td align=left
        valign=middle&gt; &lt;input type=radio
        name=rowid>$[NAME] &lt;/td&gt;"/&gt;
    &lt;ie:param name="ATTRIBUTE" data="&lt;td width=25&gt;
        &nbsp; &lt;/td&gt;"/&gt;
    &lt;ie:param name="ATTRIBUTE" data="&lt;td align=left
        valign=middle&gt;${EMAIL} &lt;/td&gt;"/&gt;
    &lt;ie:param name="POST_OBJECT_TEXT" data="&lt;/tr&gt;"/>
  </ie:webject>
</table>
```

**Output:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>GROUP_IN</td>
<td>PARAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BUNDLE</td>
</tr>
</tbody>
</table>

**BUNDLE**

Indicates the Java class resource bundle from which the localized string will be 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.
The following Display-Resource webject example is contained in the DisplayResource.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/DisplayResource.jsp

**Note:** The group used by the Display-Resource webject in this example is created with the Get-Resource webject. For additional information on the Get-Resource webject, please refer to the chapter in this guide titled Task Webject Reference.

**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>

<ie:webject name="Display-Resource" type="DSP">
  <ie:param name="GROUP_IN" data="IEResource"/>
  <ie:param name="KEY" data="19"/>
</ie:webject>
```

**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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST_HEADER</td>
<td>ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td>LIST_HEADER_VALUE</td>
<td>ATTRIBUTE_VALUE</td>
<td></td>
</tr>
<tr>
<td>MULTIPLE</td>
<td>GROUP_IN</td>
<td></td>
</tr>
<tr>
<td>SELECTED</td>
<td>LEADER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WIDTH</td>
<td></td>
</tr>
</tbody>
</table>

ATTRIBUTE

Specifies the name of the object attribute whose value will be used as the value attribute in the generated HTML OPTION element and as the value displayed,
and whose attribute name will be 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 will be determined by ATTRIBUTE-VALUE, while the value displayed by the HTML OPTION element will remain 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 will be 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 will determine the name value for the generated HTML SELECT element and the value attribute in the generated HTML OPTION element, while the value displayed will be 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, refer to the chapter titled How Info*Engine Manages Data.

**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 pre-select 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 pre-selected.

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 will be 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 will be applied on a one-to-one basis beginning with the first attribute, with the remaining attributes using the default value. This parameter is optional.
The following Display-Selection webject examples are contained in the DisplaySelection.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/DisplaySelection.jsp

The group displayed in each of these examples is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

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:

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:

<table>
<thead>
<tr>
<th>Sam Johnson</th>
<th>Sam Johnson</th>
<th>Harvy Anderson</th>
<th>James O'Connor</th>
<th>Harvey Hampton</th>
</tr>
</thead>
</table>

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 will be 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:

![Select dropdown menu with options]

Generated HTML:

```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:**

```html
<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:**

```text
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>
```

**Generated HTML:**

```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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St. <a href="mailto:sjjohnson@somewhere.com">sjjohnson@somewhere.com</a></td>
<td></td>
</tr>
<tr>
<td>Harvey Anderson</td>
<td>1234 Amber St. <a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
<td></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St. <a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
<td></td>
</tr>
</tbody>
</table>

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. sjjohnson@somewhere.com</option>
    <option value='1234 Amber St.'>Harvey 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:**

```html
<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:**

```
All Employees
Sam Johnson
Harvy Anderson
James O'Connor
Harvey Hampton
```

**Generated HTML:**

```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:

```html
<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:

```
All Employees
No Employees
Sam Johnson
Harvy Anderson
James O'Connor
Harvey Hampton
```

Generated HTML:

```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:

```html
<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:

```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>
</select>
</form>
```

Generated HTML:

```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>
</select>
</form>
```
Display Single Attribute With All Options Pre-Selected

Webject:

```html
<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:

```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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPERLINK</td>
<td>ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td>HYPER_VALUE</td>
<td>BORDER</td>
<td></td>
</tr>
<tr>
<td>PATH_INFO</td>
<td>FOOTER</td>
<td></td>
</tr>
<tr>
<td>TEMPLATE_URL</td>
<td>GROUP_IN</td>
<td></td>
</tr>
<tr>
<td>HEADER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIDTH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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, refer to the chapter titled How Info*Engine Manages Data.

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 will be 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 will cause 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 will be 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 the Dynamic Parameter Value Substitution section.

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 will be 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 will be 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 the Dynamic Parameter Value Substitution section.

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 will be 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 e-mail 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 preformatted 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 are contained in the DisplayTable.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/DisplayTable.jsp

The group displayed in each of these examples is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

Default Display
In the following example, all of the information from the CreateGroup task is displayed using the default values for the table.

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

Output:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>

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'Conner</td><td>775 Main St.</td><td>hhampton@somewhere.com</td></tr>
<tr><td>Harvey Hampton</td><td>775 Main St.</td><td>hhampton@somewhere.com</td></tr>
</table>
No Border, 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.  
&nbs;
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:**

```xml
<ie:webject name="Display-Table" type="DSP">
  <ie:param name="BORDER" data="10"/>
  <ie:param name="ATTRIBUTE" data="email,address" delim="","/>
</ie:webject>
```

**Output:**

<table>
<thead>
<tr>
<th>email</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
<td>1234 Main St.</td>
</tr>
<tr>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
<td>1234 amber St.</td>
</tr>
<tr>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
<td>775 Main St.</td>
</tr>
</tbody>
</table>

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:**

```xml
<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:

<table>
<thead>
<tr>
<th>Email Address</th>
<th>Home Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234 Main St.</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>1234 Amber St.</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>

Hyperlinked Columns

In the following example, the HYPERLINK parameter identifies the columns that contain hyperlinks. The WIDTH parameter overrides the default.

Webject:

```xml
<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="class=employees"/>
  <ie:param name="HYPER_VALUE" data="Name,Address,Email" delim="","/>
</ie:webject>
```

Output:

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>EMAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St</td>
<td><a href="mailto:sjohnson@somewhere.com">sjohnson@somewhere.com</a></td>
</tr>
<tr>
<td>Harry Anderson</td>
<td>1234 Amber St</td>
<td><a href="mailto:handerson@somewhere.com">handerson@somewhere.com</a></td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
<td></td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
<td><a href="mailto:hhampton@somewhere.com">hhampton@somewhere.com</a></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE</td>
<td>NAME</td>
<td>GROUP_IN</td>
</tr>
<tr>
<td>PROMPT</td>
<td>PRE_TABLE_TEXT</td>
<td></td>
</tr>
<tr>
<td>SIZE</td>
<td>POST_TABLE_TEXT</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>UNDEFINED</td>
<td></td>
</tr>
</tbody>
</table>

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, refer to the chapter titled How Info*Engine Manages Data.

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 you will display. 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 displaying 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:

```xml
<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:

<table>
<thead>
<tr>
<th>ATTRIBUT E</th>
<th>NAME</th>
<th>PROMPT</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKBOX</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HIDDEN</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RADIO</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**UNDEFINED**

Sets the value to display for an undefined value. An undefined value is either a non-existent 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.
The following Display-Value webject examples are contained in the DisplayValue.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/DisplayValue.jsp

The group displayed in each of these examples is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

**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:**

```xml
<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:**

```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:

```html
<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.
Harvy Anderson Address: 1234 Amber St.
James O'Connor Address: 775 Main St.
Harvey Hampton Address: 775 Main St.

Generated HTML:

```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:

```html
<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 Johnson
Harvy Anderson
James O'Connor
Harvey Hampton

Generated HTML:

```html
<form>
  Sam Johnson
  Harvy Anderson
  James O'Connor
  Harvey Hampton
</form>
```

Multiple Attributes With Prompts, Default Names, and TYPE=TEXT

Webject:

```html
<form>
  ie:webject name="Display-Value" type="DSP">
    <ie:param name="ATTRIBUTE" data="NAME,ADDRESS" delim="","/>
    <ie:param name="PROMPT" data="EmployeeNAME - ,
                                 EmployeeAddress - " delim="","/>
    <ie:param name="TYPE" data="text"/>
    <ie:param name="SIZE" data="20"/>
    <ie:param name="CELLPADDING" data="5"/>
  </ie:webject>
</form>
```

Output:

<table>
<thead>
<tr>
<th>EmployeeNAME</th>
<th>EmployeeAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
</tr>
<tr>
<td>Harvy Anderson</td>
<td>1234 Amber St.</td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
</tr>
</tbody>
</table>
Multiple Attributes With Prompts, Specified Names, and TYPE=TEXT

Webject:

```xml
<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:

<table>
<thead>
<tr>
<th>NamePrompt</th>
<th>Sam Johnson</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NamePrompt</td>
<td>Harvy Anderson</td>
<td>AddressPrompt</td>
</tr>
<tr>
<td>NamePrompt</td>
<td>James O'Connor</td>
<td>AddressPrompt</td>
</tr>
<tr>
<td>NamePrompt</td>
<td>Harvey Hampton</td>
<td>AddressPrompt</td>
</tr>
</tbody>
</table>
Multiple Attributes With Prompts, Specified Names, and Multiple TYPE Values

Webjet:

```html
<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,password" delim="","/>
  <ie:param name="SIZE" data="20/20,20/20"/>
  <ie:param name="CELLPADDING" data="5"/>
</ie:webject>
</form>
```

Output:

<table>
<thead>
<tr>
<th>Name Prompt</th>
<th>Sam Johnson</th>
<th>Address Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Prompt</td>
<td>Harvey Anderson</td>
<td>Address Prompt</td>
</tr>
<tr>
<td>Name Prompt</td>
<td>James O'Conner</td>
<td>Address Prompt</td>
</tr>
<tr>
<td>Name Prompt</td>
<td>Harvey Hampton</td>
<td>Address Prompt</td>
</tr>
</tbody>
</table>

Multiple Attributes With Prompts and Specified Names, TYPE = CHECKBOX

Webjet:

```html
<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.
- [ ] Harvey Anderson 1234 Amber St.
- [ ] James O'Connor 775 Main St.
- [ ] Harvey Hampton 775 Main St.
Multiple Attributes with Prompts, TYPE=TEXTAREA

Webject:

```html
<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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam Johnson</td>
<td>1234 Main St.</td>
</tr>
<tr>
<td>NamePrompt =</td>
<td>AddressPrompt =</td>
</tr>
<tr>
<td>Harvey Anderson</td>
<td>1234 Amber St.</td>
</tr>
<tr>
<td>NamePrompt =</td>
<td>AddressPrompt =</td>
</tr>
<tr>
<td>James O'Connor</td>
<td>775 Main St.</td>
</tr>
<tr>
<td>NamePrompt =</td>
<td>AddressPrompt =</td>
</tr>
<tr>
<td>Harvey Hampton</td>
<td>775 Main St.</td>
</tr>
<tr>
<td>NamePrompt =</td>
<td>AddressPrompt =</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Password Prompt:</th>
<th>~~~~~~~~~~~~~~~~~~~</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password Prompt:</td>
<td>~~~~~~~~~~~~~~~~~~~</td>
</tr>
<tr>
<td>Password Prompt:</td>
<td>~~~~~~~~~~~~~~~~~~~</td>
</tr>
<tr>
<td>Password Prompt:</td>
<td>~~~~~~~~~~~~~~~~~~~</td>
</tr>
</tbody>
</table>
**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 will corrupt 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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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, refer to the chapter titled How Info*Engine Manages Data.

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.
EXAMPLE

The following Display-XML webject example is contained in the DisplayXml.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebobjects directory.

Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

http://localhost/Windchill/infoengine/jsp/examples/dspwebobjects/DisplayXml.jsp

The group displayed in this example is the group generated by executing the "infoengine/jsp/examples/CreateGroup.xml" task.

For an example that shows the XML output using MODE set to FULL, see the Set-Metadata webject.

Webject:

```java
<%@page language="java" session="false" errorPage="../IEError.jsp"%>
<%@ taglib uri=http://www.ptc.com/infoengine/taglib/core prefix="ie" %>
<ie:task uri="infoengine/jsp/examples/CreateGroup"/>
<ie:webject name="Display-XML" type="DSP"/>
```

Generated XML:

```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’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 Echo-Request 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.

**EXAMPLE**
The following Echo-Request webject example is contained in the EchoRequest.jsp file that is located in the Info*Engine infoengine/jsp/examples/dspwebjects directory. Assuming that the Info*Engine application URL prefix is "Windchill" and your Web server is your local host, you can execute this file by entering the following URL:

```
http://localhost/Windchill/infoengine/jsp/examples/dspwebjects/EchoRequest.jsp
```

The group displayed in this example is the group generated by executing the "infoengine/examples/CreateGroup.xml" task.

**Webject:**

```
<ie:task uri="infoengine/examples/CreateGroup"/>
<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 "infoengine/examples/CreateGroup.xml" task.
Image Webjects for JPEG

The following webjects can be used to display images in a JPEG format.

- Bar-Graph
- Line-Graph
- Pie-Chart
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="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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
<td>BACKDROP_COLOR</td>
<td></td>
</tr>
<tr>
<td>GROUP_IN</td>
<td>BACKGROUND_COLOR</td>
<td>COLOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECIMAL_PLACES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HEIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_BOTTOM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_LEFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_RIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_TOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSETS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LABEL_ANGLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCALE_INCREMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TEXT_COLOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WIDTH</td>
</tr>
</tbody>
</table>

**BACKDROP_COLOR**
Specifies the color, in #RRGGBB, format of back drop in the generated image. The default value is black 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 will be iterated over until all data in the input group has been graphed. Example: specifying two colors parameters like:

```
<ie:param name="COLOR" data="#FF0000,#0000FF" delim="",/>
```

will cause 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. 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. 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 over-rides 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. Default is not to display labels.

**LABEL_ANGLE**
Specifies the integer angle at which to display the labels. This can be used to avoid over-lapping 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 will start at 0 and appear every SCALE_INCREMENT until the largest data point in the data to be graphed. Default is not to 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. Default value is 600.

EXAMPLE
Given the following input data (in GROUP_IN named Employees)

<table>
<thead>
<tr>
<th>Name</th>
<th>Sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>7000</td>
</tr>
<tr>
<td>fred</td>
<td>5000</td>
</tr>
<tr>
<td>king</td>
<td>15000</td>
</tr>
<tr>
<td>bill</td>
<td>2500</td>
</tr>
<tr>
<td>frank</td>
<td>7500</td>
</tr>
</tbody>
</table>

and the following JSP:

```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**

```xml
<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="GROUP_IN" data="input_groupname"/>
  <ie:param name="GRAPH" data="column_name"/>
  <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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
<td>BACKDROP_COLOR</td>
<td></td>
</tr>
<tr>
<td>GROUP_IN</td>
<td>BACKGROUND_COLOR</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>COLOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DECENTRAL_PLACES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HEIGHT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSET_BOTTOM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSET_LEFT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSET_RIGHT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSET_TOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSETS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAXY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MINY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT_COLOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WIDTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XSCALE_INCREMENT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YSCALE_INCREMENT</td>
<td></td>
</tr>
</tbody>
</table>

**BACKDROP_COLOR**
Specifies the color, in #RRGGBB, format of back drop in the generated image. The default value is black 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 lines 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 will be iterated over until each data set in the input group has been graphed. Example: specifying two colors parameters like:

```
<ie:param name="COLOR" data="#FF0000,#0000FF" delim="","/>
```

will cause 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. 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. 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 over-rides 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. 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 will start at the smallest X value and appear every XSCALE_INCREMENT until the largest X value. Default is not to 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 will 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. Default is not to display a scale on the Y axis.

EXAMPLE

Given the following input data (in GROUP_IN named dataSet)

<table>
<thead>
<tr>
<th>X</th>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>100.0</td>
<td>200.0</td>
</tr>
<tr>
<td>1.0</td>
<td>200.0</td>
<td>50.0</td>
</tr>
<tr>
<td>2.0</td>
<td>300.0</td>
<td>400.0</td>
</tr>
<tr>
<td>3.0</td>
<td>150.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4.0</td>
<td>50.0</td>
<td>10.0</td>
</tr>
<tr>
<td>5.0</td>
<td>400.0</td>
<td>30.0</td>
</tr>
<tr>
<td>6.0</td>
<td>375.0</td>
<td>40.0</td>
</tr>
</tbody>
</table>

and the following JSP:

```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: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. For an example see the JSP infoengine/jsp/examples/img/lineGraphExample.jsp.
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:weobject 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:weobject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
<td></td>
<td>BACKGROUND_COLOR</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>COLOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECIMAL_PLACES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HEIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_BOTTOM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_LEFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_RIGHT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSET_TOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSETS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LABEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TEXT_COLOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WIDTH</td>
</tr>
</tbody>
</table>
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 will be iterated over until all data in the input group has been graphed. Example: specifying two colors parameters like:

<ie:param name="COLOR" data="#FF0000,#0000FF" delim="","/>

will cause 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. 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 over-rides 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. 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. Default value is 600.

EXAMPLE
Given the following input data (in GROUP_IN named Employees)

<table>
<thead>
<tr>
<th>Name</th>
<th>Sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>7000</td>
</tr>
<tr>
<td>fred</td>
<td>5000</td>
</tr>
<tr>
<td>king</td>
<td>15000</td>
</tr>
<tr>
<td>bill</td>
<td>2500</td>
</tr>
<tr>
<td>frank</td>
<td>7500</td>
</tr>
</tbody>
</table>

and the following JSP:

```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:
This chapter contains 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.

In tables at the beginning of the parameter descriptions, the parameters are categorized as being either Required, Select, or Optional:

- A parameter is listed in the Required column when it is always required.
- A parameter is listed in the Select column when there is a relationship between the specified parameter and another parameter. For example, the ELEMENT and XML_URL parameters of the Create-Group webject are select because at least one of the parameters is required although both can be specified.
- 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.
- A parameter is listed in the Optional column when it is always optional and when it is not related to another parameter.
The Info*Engine task webobjects described in this chapter are grouped according to their type in the following sections:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrative Webobjects</td>
<td>10-3</td>
</tr>
<tr>
<td>Group Webobjects</td>
<td>10-6</td>
</tr>
<tr>
<td>Management Webobjects</td>
<td>10-78</td>
</tr>
<tr>
<td>Message Webobjects</td>
<td>10-104</td>
</tr>
<tr>
<td>Web Event Service Webobjects</td>
<td>10-149</td>
</tr>
</tbody>
</table>
Administrative Webjectives

Administrative webjectives provide a quick way to perform specific administrative tasks, such as gathering simple statistics, or causing service properties to reload at runtime.

All administrative webjectives 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
  - 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 ADM webobjects are not included in the statistics.

**Note:** This webobject is only supported by processes listening on a port, such as an out-of-process adapter or a task processor.

SYNTAX

```xml
<ie:webobject name="Get-Statistics" type="ADM">
  <ie:param name="INSTANCE" data="instance"/>
</ie:webobject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTANCE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INSTANCE**

Specifies the name of the process in which this webobject is to be run. An adapter that is running out-of-process or a task processor are both processes. This parameter is required.

EXAMPLE

```xml
<ie:webobject name="Get-Statistics" type="ADM">
  <ie:param name="INSTANCE" data="com.myCompany.myHost.server.taskProcessor"/>
</ie:webobject>
```
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 IeSevlet, 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SERVICE_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTANCE</td>
</tr>
</tbody>
</table>

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 re-loaded. If this parameter is omitted the webject re-loads the properties for 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 Webobjects

Group webobjects compare, combine, or sort one or more existing groups of data that have been generated as a result of query, action, or other group webobjects.

All group webobjects use the GRP type attribute value in the webject tag.
Call-Task

DESCRIPTION

Calls an Info*Engine task from within another Info*Engine task. If the called task is executed within the same task processor as the calling task, then they will share the same VDB. If the called task is executed in a remote task processor (see the PROCESSOR parameter), then a snapshot of the calling task's VDB is sent to the called task, and any groups produced by the called task (see the Return-Groups webject) are added to the VDB of the calling task.

SYNTAX

<iw_class name="Call-Task" type="GRP">
  <ie:param name="GROUP_IN" data="value"/>
  <ie:param name="PARAM" data="name=value"/>
</iw_class>

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK</td>
<td>GROUP_COOKIE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_FORM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_SERVER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PROCESSOR</td>
<td></td>
</tr>
</tbody>
</table>

GROUP_COOKIE

Specifies the name of a group whose attributes represent HTTP cookies sent by a web browser. The called task can then find these attributes in its @COOKIE group. If this parameter is omitted, the called task inherits the @COOKIE group of the calling task.

GROUP_FORM

Specifies the name of a group whose attributes represent parameters to be passed to the called task. The called task can then find these parameters as attributes of its @FORM group. If this parameter is omitted, the called task’s @FORM group will be empty (except see the PARAM parameter below).

GROUP_SERVER

Specifies the name of a group whose attributes represent server-provided context information. The called task can then find these values as attributes of its @SERVER group. If this parameter is omitted, the called task inherits the @SERVER group of the calling task.
PARAM
Specifies name=value pairs to be added as attributes to the called task's @FORM group. This parameter can be specified multiple times to provide multiple @FORM attributes. If an attribute specified by a PARAM value matches the name of an attribute already contained in the called task's @FORM group, the value specified by PARAM replaces the one in the @FORM group.

PROCESSOR
Specifies the name of a remote task processor. The value specified by the TASK parameter references a task available in that remote task processor, and it will be executed in that remote task processor. This parameter may be specified multiple times to provide the names of multiple, alternative, remote task processors in which the task may be executed. In this case, the webject will execute the task in the first task processor that is accessible to it. If this parameter is omitted, the called task will be executed in the task processor of the calling task (the usual case).

TASK
Specifies the URI (usually, a relative file URL) of the task to execute. This parameter is required.
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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td>RENAME</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 will be ignored when the parameter searches for the existing attribute name.

This parameter is required.
EXAMPLE TASK

The following ChangeGroup.xml file that is located in the Info*Engine tasks "examples" directory defines the Change-Group webject. The webject changes the attribute names in the input group named "createdgroup" and adds the "results" group to the local output group collection:

```xml
<?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’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>
```

The XML output from executing the ChangeGroup.xml file is the following:

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

<!-- Creates input group named createdgroup. -->
<ie:task uri="examples/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>
```
**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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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:

```xml
<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 TASK

The following ConcatGroup.xml file that is located in the Info*Engine tasks "examples" directory links the two specified groups together.

```xml
<%@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="examples/CreateGroup.xml"/>
<ie:task uri="examples/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

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 TASK

The following CopyGroup.xml file that is located in the Info*Engine tasks "examples" directory defines the Copy-Group webject. The webject copies the input group named "createdgroup" and to the "results" group and returns both groups (using the Return-Groups webject) to the local output group collection:

```xml
<%@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="examples/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 CopyGroup.xml file is the following:

```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’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’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 will return 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 the section titled XML Output for Info*Engine Groups in the Info*Engine Tasks chapter.

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**

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELEMENT</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>DBUSER</td>
<td></td>
</tr>
<tr>
<td>XML_URL</td>
<td>DELIMITER</td>
<td></td>
</tr>
<tr>
<td>XSL_PARAM</td>
<td>PASSWD</td>
<td></td>
</tr>
<tr>
<td>XSL_URL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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:

```xml
<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 user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

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 the section titled Dynamic Parameter Value Substitution.

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 will create 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 will be 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 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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.

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 will parse all of them and add them to its collection of output groups.

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 will be 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 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.

If the data value is "input:", the webject reads on XML object from the task's BLOB input stream. This allows XML objects to be submited 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 URL’s are relative to the Info*Engine Server task template root.

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 will be 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 template root directory.
EXAMPLE TASK

Internal Group Creation

The following CreateGroup.xml file that is located in the Info*Engine tasks examples directory defines the Create-Group webject. The webject specifies the elements in the output group named "createdgroup" and adds the group to the local output group collection:

```xml
<%@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="CLASS" data="EmployeeData"/>
  <ie:param name="GROUP_OUT" data="createdgroup"/>
</ie:webject>

The XML output from executing the CreateGroup.xml file is the following:

```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'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>
```
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:

```xml
<%@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>
```

</EmployeeData>
</wc:COLLECTION>
**Diff-Groups**

**DESCRIPTION**

Computes the difference of two groups and places that difference in an output group.

In general, an element is in the difference of two groups if it is part of the first group but not both groups. More specifically, 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. Finally, 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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIFFBY</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td>COMPARISON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORTBY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORTED</td>
</tr>
</tbody>
</table>

**CASE_IGNORE**

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

```xml
<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 will be 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 will be 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 TASK

The following DiffGroups.xml file that is located in the Info*Engine tasks "examples" directory creates two groups using 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:

```xml
%@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=125 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 DiffGroups.xml file is the following:

```xml
<?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 the Subset-Group webject description.

When used in combination with the EXTRACT parameter, a specific set of attributes will be 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 will 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 will 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 will contain fewer ELEMENT_COUNT elements (it will contain 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 parameters EXTRACT, ELEMENT_OFFSET, or ELEMENT_COUNT 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="GROUP_IN" data="group_name"/>
  <ie:param name="GROUP_OUT" data="output_group"/>
</ie:webject>
```

**PARAMETERS**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td>ELEMENT_COUNT</td>
<td>CLASS</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>ELEMENT_OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXTRACT</td>
</tr>
</tbody>
</table>

**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:
The default for this parameter is "Unknown-Class-Name". This parameter is optional.

**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.

**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.

**EXTRACT**
Specifies the name of the attribute to be extracted. Multiple values can be specified for this parameter. This parameter is required.

**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 TASK**

The following ExtractGroup.xml file that is located in the Info*Engine tasks "examples" directory searches for the specified attribute names and values and puts them into a new group named "extractresults".

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

<!-- Extract attributes from a group. -->
<ie:task uri="examples/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 tag 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
<td>LOCALE</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ATTRIBUTE

Identifies the string of data in a group to be formatted. These define the attributes that will be 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 will have 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 will be substituted. If n is 1, the second unconsumed attribute value is substituted. If n is 2, the third unconsumed attribute value is substituted, etc. 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 TASK

The following FormatGroup.xml file that is located in the Info*Engine tasks "examples" directory 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".

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

<!-- Format elements of a group. -->
<ie:task uri="examples/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="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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>INTERSECTBY</td>
<td>COMPARISON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORTBY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORTED</td>
<td></td>
</tr>
</tbody>
</table>

**CASE_IGNORE**

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

```xml
<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:

```xml
<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 will be 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 TASK

The following IntersectGroups.xml file that is located in the Info*Engine tasks "examples" directory 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:

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

<!--Create a group named createdgroup. -->
<ie:task uri="examples/CreateGroup.xml"/>

<!--Create a group named createdhrgroup. -->
<ie:task uri="examples/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 IntersectGroups.xml file is the following:

```xml
<?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’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 will also be included in the new group.

Note: The difference between the Join-Groups webject and the Merge-Groups webject revolves around 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td>JOINBY</td>
<td></td>
<td>COMPARISON</td>
</tr>
<tr>
<td>JOIN_TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORTBY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORTED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDEFINED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CASE_IGNORE
Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

```xml
<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:

```xml
<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 will be 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. See the EXAMPLES heading under the Join-Groups webject for more information.

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.

TASK

The following JoinGroups.xml file that is located in the Info*Engine tasks "examples" directory 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:

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

<!--Create a group named createdgroup. -->
<ie:task uri="examples/CreateGroup.xml"/>

<!--Create a group named createdhrgroup. -->
<ie:task uri="examples/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="createdhrgroup"/>
  <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 will be joined. Notice that the information in Column 3, Row 3 is not the same in these two groups.

**Table 1 Group A**

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>a3</td>
<td>b3</td>
<td>c3</td>
</tr>
</tbody>
</table>

**Table 2 Group B**

<table>
<thead>
<tr>
<th></th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>2</td>
<td>c2</td>
<td>d2</td>
<td>e2</td>
</tr>
<tr>
<td>3</td>
<td>c4</td>
<td>d4</td>
<td>e4</td>
</tr>
</tbody>
</table>

Assume that the following webject executes:

```xml
<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**

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>Row 2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td>e2</td>
</tr>
<tr>
<td>Row 3</td>
<td>a3</td>
<td>b3</td>
<td>c3</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Row 4</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>c4</td>
<td>d4</td>
<td>e4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>a1</td>
<td>b1</td>
</tr>
<tr>
<td>Row 2</td>
<td>C2</td>
<td>d2</td>
<td>e2</td>
<td>a2</td>
<td>b2</td>
</tr>
<tr>
<td>Row 3</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>a3</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>Row 4</td>
<td>C4</td>
<td>d4</td>
<td>e4</td>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>
**Merge-Groups**

**DESCRIPTION**

Combines two groups into one and allows duplication of information contained in either or both groups.

*Note:* The difference between the Join-Groups webject and the Merge-Groups webject revolves around 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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMPARISON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORTBY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORTED</td>
</tr>
</tbody>
</table>

**CASE_IGNORE**

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:
Correlation (Optional)

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:

```xml
<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 TASK

The following MergeGroups.xml file that is located in the Info*Engine tasks "examples" directory 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".

```xml
<%@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="examples/CreateGroup.xml"/>
<ie:task uri="examples/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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 TASK

The following ReturnGroups.xml file that is located in the Info*Engine tasks "examples" directory creates two groups using the CreateGroup and CreateGroupHr task tags and then returns the groups using the Return-Groups webject:

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

<!-- Create a group named createdgroup. -->
<ie:task uri="examples/CreateGroup.xml"/>

<!-- Create a group named createdhrgroup. -->
<ie:task uri="examples/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
<?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’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’Connor</NAME>
      <POSITION>Management</POSITION>
      <PHONE></PHONE>
    </wc:INSTANCE>
    <wc:INSTANCE>
      <NAME>Harvey Hampton</NAME>
      <POSITION></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 applied easily to the group by providing it as the value of the GROUP_IN parameter of the Dispatch-Tasks webject.

**SYNTAX**

```xml
<ie:webject name="Set-Identity" type="GRP">
  <ie:param name="GROUP_IN" data="input_group_name"/>
</ie:webject>
```

**PARAMETERS**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td>DOMAIN</td>
<td>CLASS</td>
</tr>
<tr>
<td></td>
<td>DOMAIN_ATTRIBUTE</td>
<td>CLASS_ATTRIBUTE</td>
</tr>
<tr>
<td>UFID</td>
<td>GUID</td>
<td>GUID_ATTRIBUTE</td>
</tr>
<tr>
<td>LOCAL_ID</td>
<td>LOCAL_ID_ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UFID_ATTRIBUTE</td>
<td></td>
</tr>
</tbody>
</table>

**CLASS**

Specifies a logical object class name. This value will be 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 will be 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 will be 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 such value will be combined with LOCAL_ID and GUID to form a UFID (unique federation identifier) value that will be 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 will be added. This group is modified in place (i.e., this object 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 will be 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 will be 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 will be 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 both LOCAL_ID_ATTRIBUTE and
UFID are also omitted, then no UFID attribute will be 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 will be 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 will be added to any elements of the input group. This parameter is optional.

**UFID**
Directory specifies the value that will be added as a UFID 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 both LOCAL_ID and LOCAL_ID_ATTRIBUTE are also omitted, then no UFID attribute will be added to any elements of the input group. This parameter is optional.

**UFID_ATTRIBUTE**
Specifies the name of the attribute that will be added to each element of the input group to identify each element's UFID. The default value of this parameter is "oid". 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td>SCOPE</td>
<td>VALUE</td>
</tr>
<tr>
<td>NAME</td>
<td>ELEMENT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTRIBUTE</td>
</tr>
</tbody>
</table>

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 will be 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 will set 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 SetMetadata.jsp file that is located in the Info*Engine JSP "examples" directory registers metadata with a name of "meta_data" and a value of "met_value" on the group named "createdgroup" at the group level.

```xml
<%@ 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="infoengine/examples/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:

```jsp
<%@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: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 SetMetadataLevels.jsp is as follows:

```xml
<?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</NAME>
      <wc:Meta>
        <testattribute>attribute metadata</testattribute>
      </wc:Meta>
      <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</NAME>
      <wc:Meta>
        <testattribute>attribute metadata</testattribute>
      </wc:Meta>
      <ADDRESS>1234 Amber St.</ADDRESS>
      <EMAIL>handerson@somewhere.com</EMAIL>
      <wc:Meta>
        <testelement>element metadata</testelement>
      </wc:Meta>
    </wc:INSTANCE>
    <wc:INSTANCE>
      <NAME>'</NAME>
      <wc:Meta>
        <testattribute>attribute metadata</testattribute>
      </wc:Meta>
      <EMAIL>joconnor@somewhere.com</EMAIL>
      <wc:Meta>
        <Class>Unknown-Class-Name</Class>
        <testgroup>group metadata</testgroup>
        <Status>0</Status>
      </wc:Meta>
    </wc:INSTANCE>
  </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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td>SORTBY</td>
<td></td>
<td>COMPARISON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SORTED</td>
</tr>
</tbody>
</table>

**CASE_IGNORE**

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

```xml
<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. Keep in mind 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 SortGroup.xml file that is located in the Info*Engine tasks "examples" directory 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".

```xml
<%@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="examples/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>
```

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.

```xml
<%@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:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>Smith</td>
</tr>
<tr>
<td>300</td>
<td>Johnson</td>
</tr>
<tr>
<td>200</td>
<td>Reilly</td>
</tr>
<tr>
<td>100</td>
<td>Sinclair</td>
</tr>
<tr>
<td>300</td>
<td>Michaels</td>
</tr>
<tr>
<td>500</td>
<td>King</td>
</tr>
</tbody>
</table>

The group "employees" is then sorted by department number, in ascending order.

```xml
<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:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Sinclair</td>
</tr>
<tr>
<td>200</td>
<td>Reilly</td>
</tr>
<tr>
<td>300</td>
<td>Smith</td>
</tr>
<tr>
<td>300</td>
<td>Johnson</td>
</tr>
<tr>
<td>300</td>
<td>Michaels</td>
</tr>
<tr>
<td>500</td>
<td>King</td>
</tr>
</tbody>
</table>

Note that 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.

```xml
<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:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Sinclair</td>
</tr>
<tr>
<td>200</td>
<td>Reilly</td>
</tr>
<tr>
<td>300</td>
<td>Johnson</td>
</tr>
<tr>
<td>300</td>
<td>Michaels</td>
</tr>
<tr>
<td>300</td>
<td>Smith</td>
</tr>
<tr>
<td>500</td>
<td>King</td>
</tr>
</tbody>
</table>

Note that 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.

```xml
<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:

<table>
<thead>
<tr>
<th>DEPT</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Sinclair</td>
</tr>
<tr>
<td>200</td>
<td>Reilly</td>
</tr>
<tr>
<td>300</td>
<td>Smith</td>
</tr>
<tr>
<td>300</td>
<td>Michaels</td>
</tr>
<tr>
<td>300</td>
<td>Johnson</td>
</tr>
<tr>
<td>500</td>
<td>King</td>
</tr>
</tbody>
</table>

Note that 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILTER</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td>CLASS</td>
<td>FILTER_MODE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>FILTER_TYPE</td>
<td></td>
</tr>
</tbody>
</table>

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case will be ignored when searching for matches. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

<table>
<thead>
<tr>
<th>IE</th>
<th>Regular Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>.</td>
</tr>
<tr>
<td>*</td>
<td>.*</td>
</tr>
<tr>
<td>pattern</td>
<td>^pattern$</td>
</tr>
</tbody>
</table>

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 will be stored. This parameter is required.
EXAMPLE TASK

The following SubsetGroup.xml file that is located in the Info*Engine tasks "examples" directory 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".

```xml
<%@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.
Note: This example requires the gnu-regexp.jar file. See the
Info*Engine User Guide Subset-Group Webject documentation on how
to obtain the gnu-regexp.jar file. -->

<ie:task uri="examples/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**

```xml
<ie:weobject 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:weobject>
```

**PARAMETERS**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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:

```xml
<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 will have 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 TASK

The following SummarizeGroups.xml file that is located in the Info*Engine tasks "examples" directory provides descriptive information about the specified GROUP_INs and places the resulting data in an output group named "result".

```xml
<%@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="examples/CreateGroup.xml"/>
<ie:task uri="examples/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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td>COPY_UNTRANSLATED</td>
<td>CLASS</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>DEFAULT_TABLE</td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td>NAME_TRANS_TABLE</td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td>TYPE_TRANS_TABLE</td>
<td>XSL_PARAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSL_URL</td>
</tr>
</tbody>
</table>

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:

```xml
<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 user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

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 with name "Factory" and value "Unknown", and after translation a GROUP_OUT element does not contain an attribute named "Factory", then an attribute with name "Factory" and value "Unknown" will be 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 will be 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 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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.

**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 will be 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. 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 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 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/servlet/IE/createGroupData.xml, the Auth-Map context group will be searched for a user name 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 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, the stylesheet is assumed to be a local file relative to the current task template root directory.

This parameter is optional.

EXAMPLE TASK

The following TranslateGroup1.xml file that is located in the Info*Engine tasks "examples" directory translates the data from the group named "createdgroup" by renaming the attributes. The resulting data is placed in the group named "RenamedGroup".

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

<!-- Create a test Group -->
<ie:task uri="examples/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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td></td>
<td>CASE_IGNORE</td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>UNIONBY</td>
<td>COMPARISON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORTBY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORTED</td>
<td></td>
</tr>
</tbody>
</table>

**CASE_IGNORE**

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:

```xml
<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:

```xml
<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 TASK

The following UnionGroups.xml file that is located in the Info*Engine tasks "examples" directory 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".

```xml
<%@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="examples/CreateGroup.xml"/>
<ie:task uri="examples/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, 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP_IN</td>
<td>CASE_IGNORE</td>
<td></td>
</tr>
<tr>
<td>GROUP_OUT</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>XORBY</td>
<td>COMPARISON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SORTBY</td>
<td>SORTED</td>
</tr>
</tbody>
</table>

CASE_IGNORE

Acts as a flag for case. If TRUE is specified, case will be ignored. If FALSE is specified, then case is significant. The default for this parameter is TRUE. 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:
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:

```xml
<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 will be compared.

The attribute name placed in the output group is the attribute name of the first XORBY parameter used.

This parameter is required.
EXAMPLE TASK

The following XorGroups.xml file that is located in the Info*Engine tasks "examples" directory finds the "exclusive or" of the two specified input groups, and places the resulting data in an output group named "results".

```xml
<%@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

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.

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="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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>ID_ATTRIBUTE</td>
<td>CLIMBER</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td>ID_META_ATTRIBUTE</td>
<td>GROUP_VDB</td>
</tr>
<tr>
<td>TASKS</td>
<td>TYPE_ATTRIBUTE</td>
<td>MAX_CONCURRENT</td>
</tr>
<tr>
<td></td>
<td>TYPE_META_ATTRIBUTE</td>
<td>PARAM</td>
</tr>
</tbody>
</table>

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 will be 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 will be provided to each of the called tasks in their VDB's. These names populate each of the called tasks' VDB's 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 fail 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 will be 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 item 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.
<%@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/examples/CreateGroup.xml"/>
<ie:param name="element" data="action=query:type=part:domain=s2.ptc.com:task=infoengine/examples/CreateGroupA.xml"/>
<ie:param name="element" data="action=query:type=part:domain=s3.ptc.com:task=infoengine/examples/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>
<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>
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.

The generated WSDL contains SOAP information specific to the Soap RPC servlet that allows it to properly route requests. The WSDL generated 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 further information on SOAP, see Using SOAP Requests.

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

```xml
<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="SOAP_URI" data="uri"/>
  <ie:param name="TASKS" data="task_location"/>
  <ie:param name="REPOSITORY" data="repository"/>
  <ie:param name="REPOSITORY_TYPE" data="repository_type"/>
</ie:webject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>GROUP_OUT</td>
<td>TASKS</td>
</tr>
<tr>
<td></td>
<td>SOAP_URI</td>
<td>REPOSITORY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REPOSITORY_TYPE</td>
</tr>
</tbody>
</table>

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 becomes 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 will be stored as metadata with the identifier "com.infoengine.soap.wsdl.comment". Task comments will be stored as metadata on the corresponding element. Parameter comments will be 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 will be generated based on the virtual machine name of the running JVM. For example: a virtual machine name of 'com.myCompany.host.server' would result 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 will be 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 URI's 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, then the WSDL is generated and returned to the client as a BLOB. If GROUP_OUT is not specified, then SOAP_URI must be specified.

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.

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

```xml
<ie:webject name="Get-Properties" type="MGT">
  <ie:param name="GROUP_OUT" data="group_name"/>
  <ie:param name="SOURCE" data="uri_or_file_pathname"/>
</ie:webject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GROUP_OUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOURCE</td>
<td></td>
</tr>
</tbody>
</table>

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 will contain 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 file system pathname 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.
THE FOLLOWING Get-Properties example is in the GetProperties.jsp file located in the infoengine/jsp/examples/mgtwebobjects directory. The example gets properties from a predetermined source and returns those properties in a group that you have named.

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

<html>
<body>
<br>
<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>
<br>
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE</td>
<td></td>
<td>GROUP_OUT</td>
</tr>
</tbody>
</table>

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 is in the GetResource.jsp file located in the infoengine/jsp/examples/mgtwebobjects directory. The example creates a group using localized strings from the specified resource bundle.

```html
<%@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.
Lookup-Services

DESCRIPTION

This webject performs directory searches to find Info*Engine services. The resulting group will contain 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 you can only count on dn and ptcServiceName being found in all elements, presence of the others will be dependant 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CLASS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FILTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_OUT</td>
</tr>
</tbody>
</table>
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:

```xml
<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 (e.g. 
"(ptcServiceName=com.company.host.server)"). The default value is 
"(objectclass=ptcApplicationService)" which will return all services within the Naming Service's search base (the search is always performed within the VM's 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 will *only* contain services (i.e. LDAP directory entries who specify ptcApplicationService as an objectClass). The output group will contain one element per service found. Each element will contain 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 multi-valued if specified; ptcServiceName, ptcServiceAddress, ptcCoresidentService.
Map-Credentials

DESCRIPTION

Reads a file and/or executes a task to establish a group as the credentials map for the task in which the webject executes. 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 user name 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, then 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 the section titled Authentication through Credentials Mapping.

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILES</td>
<td></td>
<td>USERNAME</td>
</tr>
<tr>
<td>TASK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FILES

Specifies the pathname of a file system 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 user name,

- **passwd** specifies the credentials that will be 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 via 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 will not be 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 user name is supplied, it is assumed that the credentials mapping task will take appropriate action to reject anonymous access (for example, by throwing an exception), or provide a default credentials map for anonymous access.
The default for this parameter is for the webject to obtain a user name from the attribute named "auth-user" of the SERVER context group. This parameter is optional.

**EXAMPLE**

This example attempts to find a file in the "C:\ptc\windchill\tasks\infoengine\newExamples\auth" directory and use that file to perform an authentication mapping.

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

<!-- set up authentication information for accessing back-end information systems -->

<ie:webject name="Map-Credentials" type="MGT">
  <ie:param name="USERNAME" data="jDoe"/>
  <ie:param name="FILES"
    data="C:\ptc\windchill\tasks\infoengine\newExamples\auth"/>
</ie:webject>
```
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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>GROUP_OUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCHEMA_GENERATOR</td>
</tr>
</tbody>
</table>

**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 will be generated and an instance of the generated SchemaObject bean will be stored on the collection as the SOAP response.

**SCHEMA_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 will default 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.
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 will exit.

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUNDLE</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 is in the ThrowExceptionDefault.jsp file located in the infoengine/jsp/examples/mgtwebjects directory. This example throws an exception using the localized message identified by KEY number 19 in the specified resource bundle.

```html
<%@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 will be written to the JSP processor log.

SYNTAX

```xml
<ie:webject name="Write-Log" type="MGT">
  <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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 was written.
- `host` -- The Internet domain name of the host from which the message was 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 was 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 will be 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 trouble-shooting 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.
The following Write-Log example is in the WriteLog.jsp file located in the infoengine/jsp/examples/mgtwebjects directory. This example creates a group, writes messages to two different logs, and displays the output.

```html
<%@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="infoengine/examples/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 Webjeks**

Message webjeks provide a set of webjeks that can be used in conjunction with a third-party MOM for generic messaging functions and task queuing functions.

All message webjeks use the MSG type attribute value in the webjekt tag.
**Browse-Queue**

**DESCRIPTION**

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**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td>ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBUSER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_OUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PASSWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td></td>
</tr>
</tbody>
</table>

**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 may 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 will be returned. This parameter is optional.

**DBUSER**
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

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 default for this parameter is "browse-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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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**
Specifies the LDAP relative distinguished name of the managed queue to be browsed. The value can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP distinguished name. If relative, the "cn=" (common name attribute) is implicit if not explicitly specified. This parameter is required.

**WHERE**
Specifies the JMS Where clause used to select the messages from the queue. If no where clause is specified all messages in the queue will be returned. This parameter is optional.

**EXAMPLE TASK**
The following browseQueue.xml file that is located in the Info*Engine tasks "examples" directory defines the Browse-Queue webject. The webject browses a specified queue for all attributes, then returns the default "browse-results" group using the Return-Groups webject.

```xml
<%@page language="java" session="false"
       errorPage="../../IEError.jsp" %>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
         prefix="ie"%>

<ie:webject name="Browse-Queue" type="MSG">
   <ie:param name="QUEUE" data="${@FORM[0]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 would need to provide a form where the QUEUE variable is identified.
Call-RemoteProcedure

DESCRIPTION

Makes a remote procedure call (RPC) to a 3rdparty 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 will mimic 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 will be 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 will be returned. If the return XML is deeply nested then the resulting Info*Engine group will also be deeply nested containing attributes whose values are elements. Dealing with such responses may require the use of scriptlet code within the handling task or JSP.

SYNTAX

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION</td>
<td>AUTHORIZATION</td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PARAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SERVICE</td>
</tr>
</tbody>
</table>
AUTHORIZATION
Base64 encoded user name and password as passed from a web browser to a web server via 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
Username to use 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 will be ignored. This is an optional parameter.

GROUP_IN
Specifies 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 will be 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
Specifies the name of the Info*Engine group the RPC invocation results should be stored in. Data found in the output group will be strongly typed based on type information found in the SOAP response. This is an optional parameter. The default value is "rpcOutput".

OPERATION
Specifies 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
Specifies a parameter name and value that should be sent with the RPC. The value must be in the form of name=data. Call-RemoteProcedure will verify 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 require 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 in such a case an exception will be 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 will be ignored. This is an optional parameter.

**PORT**
Specifies the name of the SOAP port that should be used. A SOAP service may 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 will be used.

**SERVICE**
Specifies the name of the SOAP service that should be used. A WSDL document may 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 will be 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/supports, how the SOAP request should be generated (parameter order, namespaces to use, etc.). This is a required parameter. If a specific SOAP service does not have a WSDL document that describes it then one will need to be crafted by hand. If the URL specified is http or https and requires authorization it can be specified like: http://<user>:<password>@host/path/to/wsdl/.

To facilitate ease of use of this webject a utility JSP, /infoengine/jsp/tools/examineWsdl.jsp, can be used to examine WSDL documents. This JSP will describe the services, ports, operations, parameters and responses of 3rdparty web services.

A large set of JSP examples can be found at /infoengine/jsp/examples/rpc/. From the index page each example can be executed and its source and supporting WSDL examined.
EXAMPLE JSP

```jsp
<%@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><ie:getValue name="LastUpdated"></td></tr>
<tr><td align=right><strong>Conditions:</strong><td><ie:getValue name="Conditions"></td></tr>
<tr><td align=right><strong>CurrentTemp:</strong><td><ie:getValue name="CurrentTemp"></td></tr>
<tr><td align=right><strong>Humidity:</strong><td><ie:getValue name="Humidity"></td></tr>
<tr><td align=right><strong>Barometer:</strong><td><ie:getValue name="Barometer"></td></tr>
<tr><td align=right><strong>Barometer Direction:</strong><td><ie:getValue name="BarometerDirection"></td></tr>
</table><%
}%>
</body></html>
```
Create-Object

DESCRIPTION

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td>PUT_BOOLEAN</td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td>PUT_BYTE</td>
<td>GROUP_IN</td>
</tr>
<tr>
<td></td>
<td>PUT_BYTES</td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td></td>
<td>PUT_INT</td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td>PUT_CHAR</td>
<td>PRIORITY</td>
</tr>
<tr>
<td></td>
<td>PUT_DOUBLE</td>
<td>PROPERTY</td>
</tr>
<tr>
<td></td>
<td>PUT_FLOAT</td>
<td>TIME_TO_LIVE</td>
</tr>
<tr>
<td></td>
<td>PUT_LONG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUT_SHORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PUT_STRING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
<td></td>
</tr>
</tbody>
</table>

**DBUSER**

Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be 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 create a group named "create-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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 normally being the default. 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 may 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 can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP distinguished name. 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 will remain valid indefinitely. 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 may be specific to a third party recipient. Incoming messages are processed in the same manner as Unknown. The contents of a BytesMessage will be 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 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 supports several PUT_type parameters for building a message that may 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.

or the class name of a user defined message processor.
The following table indicates which PUT<sub>type</sub> parameters can be specified with which message types:

<table>
<thead>
<tr>
<th></th>
<th>BytesMessage</th>
<th>TextMessage</th>
<th>MapMessage</th>
<th>StreamMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT_BOOLEAN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_BYTE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_BYTES</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_CHAR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_DOUBLE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_FLOAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_INT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_LONG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_SHORT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PUT_STRING</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**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" which means an Info*Engine request will be 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).
EXAMPLE TASK

The following createObject.xml file that is located in the Info*Engine tasks "examples" directory defines the Create-Object webject. The webject creates a JMS message and submits it to the specified queue.

```xml
<%@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 would need to provide a form where the dbuser, passwd, queue, time, priority, and string variables are identified.
Delete-Object

DESCRIPTION

Removes a message from a JMS queue.

Note: This webject can be used only after your environment has been set up for messaging.

SYNTAX

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHERE</td>
</tr>
</tbody>
</table>

DBUSER

Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

This parameter is optional.
GROUP_OUT
Specifies 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
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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP distinguished name. 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 TASK

The following deleteObject.xml file that is located in the Info*Engine tasks "examples" directory defines the Delete-Object webject. The webject deletes a JMS message from the specified queue.

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

DESCRIPTION
Retrieves and deletes a message from an Info*Engine task queue. It is assumed that the message will contain a known InfoEngine 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

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHERE</td>
</tr>
</tbody>
</table>

DBUSER
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.
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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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**

Specifies the LDAP distinguished name of a managed queue. The value can be an LDAP distinguished name relative to a configured baseURI 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.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 users messages.

This parameter is optional.
EXAMPLE TASK

The following deleteResults.xml file that is located in the Info*Engine tasks "examples" directory defines the Delete-Results webject. The webject retrieves a message from the specified queue, then deletes the message.

```xml
<%@page language="java" session="false"
    errorPage="../IEError.jsp"
%>
<%@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: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, and where variables are identified.
Query-Object

DESCRIPTION

Retrieves a message from a JMS message queue. If there is no message to return, an instance of com.infoengine.exception.nonfatal.IERequestTimeOutException will be thrown.

Note: This webject can be used only after your environment has been set up for messaging.

SYNTAX

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUE</td>
<td>BLOB_COUNT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBUSER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_BOOLEAN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_BYTE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_BYTES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_CHAR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_DOUBLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_FLOAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_INT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_LONG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_SHORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET_STRING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PASSWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHERE</td>
<td></td>
</tr>
</tbody>
</table>

**BLOB_COUNT**

Specifies 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**

Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted. 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 may 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
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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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
Specifies the LDAP relative distinguished name of the JMS Queue from which to retrieve the message. The value can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP distinguished name. 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 TASK

The following queryObject.xml file that is located in the Info*Engine tasks "examples" directory defines the Query-Object webject. The webject retrieves a message from the specified queue.

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

DESCRIPTION

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 will contain 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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORRELATION_ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBUSER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PASSWD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIT_TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WHERE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**CORRELATION_ID**

Specifies 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 webobject. This parameter is optional.

**DBUSER**

Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

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**

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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 can be an LDAP distinguished name relative to a configured baseURI 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.defaultResultsQueue property.

This parameter is optional.

**WAIT_TIME**
Specifies 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 users messages. This parameter is optional.
EXAMPLE TASK

The following queryResults.xml file that is located in the Info*Engine tasks "examples" directory defines the Query-Results webject. The webject retrieves a message from the specified queue and returns the results.

```xml
<%@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

DESCRIPTION

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK</td>
<td></td>
<td>CORRELATION_ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESTINATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GROUP_IN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRIORITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESULT_TIME_TO_LIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TASK_TIME_TO_LIVE</td>
</tr>
</tbody>
</table>
**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 webobjects 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**
Specifies 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**
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted. This parameter is optional.

**GROUP_IN**
Specifies 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**
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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this
User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 normally being the default. 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**

Specifies 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 will 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 the section titled Specifying URIs and URLs.

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 will not discard the message. This parameter is optional.

**EXAMPLE TASK**

The following queueTask.xml file that is located in the Info*Engine tasks "examples" directory defines the Queue-Task webject. The webject adds the specified task to the specified queue.

```xml
<%@page language="java" session="false"
  errorPage="/ieError.jsp"
%>
<%@taglib uri="http://www.ptc.com/infoengine/taglib/core"
  prefix="ie"%>

<ie:webject name="queue-task" type="MSG">
  <ie:param name="TASK" data="${@FORM[TASK][0]}"
    default="infoengine/examples/CreateGroup.xml"/>
  <ie:param name="QUEUE" data="${@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

DESCRIPTION

Creates and sends an e-mail 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="CONTENT" data="message_body"/>
<ie:param name="CONTENT_TYPE" data="MIME_type"/>
<ie:param name="FROM" data="e-mail_address"/>
<ie:param name="MAIL_SERVER" data="host_name"/>
<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="TO" data="e-mail_address"/>
<ie:param name="USERNAME" data="user_name"/>
</webject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO</td>
<td>BLOB_COUNT</td>
<td>CC</td>
</tr>
<tr>
<td>CONTENT</td>
<td>BCC</td>
<td></td>
</tr>
<tr>
<td>CONTENT_LINE_BREAK</td>
<td>DBUSER</td>
<td></td>
</tr>
<tr>
<td>CONTENT_TYPE</td>
<td>FROM</td>
<td></td>
</tr>
<tr>
<td>CONTENT_URL</td>
<td>MAIL_SERVER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PASSWD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PASSWORD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REPLY_TO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUBJECT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USERNAME</td>
<td></td>
</tr>
</tbody>
</table>
BCC
Specifies the e-mail address or addresses of the blind copy recipients of the message. Each of these recipients will receive a copy of the message, but none of these addresses will be 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
Specifies 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
Specifies the e-mail address or addresses of non-primary recipients to which copies of the message will be 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
Specifies a textual 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 text/plain unless the BLOB_COUNT parameter is specified. Multiple values may be specified for this parameter.

CONTENT_LINE_BREAK
The CONTENT_LINE_BREAK parameter can be multi-valued. Each value is associated with a corresponding CONTENT parameter value and specifies the character or character sequence that represents a line break in the CONTENT value. If CONTENT_LINE_BREAK is not specified, the CONTENT is handled as one long line

CONTENT_TYPE
Specifies 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 text/plain. Similarly, if there are fewer CONTENT_TYPE parameters than CONTENT parameters, text/plain 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 will be opened and read. The content read from the URL will become a body part (main body or attachment) for the message that will be sent. For
example, this will allow the Send-Mail webobject to obtain a message body by reading a URL that executes a Windchill DCA configuration specification.

**DBUSER**
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

This parameter is optional.

**FROM**
Specifies the e-mail address of the originator of the message. The default for this parameter is to use the value specified in 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.

**HEADER_CHARSET**
Specifies the name of the character set that is used in email headers such as the subject header.

**MAIL_SERVER**
Specifies the Internet domain name of the SMTP mail server to which the message will be 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.

**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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.
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.

**PASSWORD**

Specifies 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**

Specifies the e-mail 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 e-mail address. The default for this parameter is for replies to be sent to the address specified by FROM. This parameter is optional.

**SUBJECT**

Specifies the subject header of the message. If it is not specified, the message will not contain a subject header. This parameter is optional.

**TO**

Specifies the e-mail address or addresses of the primary recipients of the message. It may be specified multiple times to specify multiple primary recipients. Each instance of the parameter specifies exactly one address. This is a required parameter.

**USERNAME**

Specifies the user name 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 user name 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 TASK

The following sendMail.xml file that is located in the Info*Engine tasks "examples" directory defines the Send-Mail webject. The webject adds the specified task to the specified queue.

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

DESCRIPTION
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:webject>
```

PARAMETERS

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXECUTE_TASK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHERE</td>
</tr>
</tbody>
</table>

DBUSER
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted. This parameter is optional.

**EXECUTE_TASK**
Specifies the task to be executed when a message is received in the subscribed queue. If the queued message already contains a task to execute, that task is executed; otherwise the value of EXECUTE_TASK is used. 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 .msg.password and .jms.password properties of the service that is used to connect to the MOM. For further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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**
Specifies the LDAP distinguished name of an administered queue. The value can be an LDAP distinguished name relative to a configured baseURI 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.

**WHERE**
Specifies 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 TASK

The following subscribeQueue.xml file that is located in the Info*Engine tasks "examples" directory defines the Subscribe-Queue webject. The webject registers the event handler to the specified queue.

```xml
<%@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

DESCRIPTION

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHERE</td>
</tr>
</tbody>
</table>

**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.

**QUEUE**

Specifies the LDAP distinguished name of an administered queue. The value can be an LDAP distinguished name relative to a configured baseURI 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.

**WHERE**

Specifies 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.
EXAMPLE TASK

The following unsubscribeQueue.xml file that is located in the Info*Engine tasks "examples" directory defines the Unsubscribe-Queue webject. The webject unregisters the event handler from the specified queue.

```xml
<%@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 Webobjects

Web Event Service (WES) webobjects provides a set of webobjects that can be used in conjunction with a third-party MOM for handling Info*Engine events.

All Web Event Service webobjects use the WES type attribute value in the webobject tag.
Emit-Event

DESCRIPTION

Causes the Info*Engine to send a named event via the JMS to be handled by any subscribers. When the execution of the webject completes successfully, an empty status group with status of 0 will be returned with a message stating that the event was successfully sent. When the execution of the webject completes unsuccessfully, an appropriate exception is thrown.

Note: This webject can be used only after your environment has been set up to use the Web Event Service.

SYNTAX

```xml
<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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td>PUT_BOOLEAN</td>
<td>DBUSER</td>
</tr>
<tr>
<td>PUT_BYTE</td>
<td>GROUP_IN</td>
<td></td>
</tr>
<tr>
<td>PUT_BYTES</td>
<td>GROUP_OUT</td>
<td></td>
</tr>
<tr>
<td>PUT_CHAR</td>
<td>PASSWD</td>
<td></td>
</tr>
<tr>
<td>PUT_DOUBLE</td>
<td>PRIORITY</td>
<td></td>
</tr>
<tr>
<td>PUT_FLOAT</td>
<td>PROPERTY</td>
<td></td>
</tr>
<tr>
<td>PUT_INT</td>
<td>TIME_TO_LIVE</td>
<td></td>
</tr>
<tr>
<td>PUT_LONG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT_SHORT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT_STRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DBUSER**
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

This parameter is optional.

**EVENT**
Specifies the LDAP distinguished name of the event to emit. It is suggested that event names be of the syntax `<event_name>@<domain>` to give additional meaning to the name and avoid name conflicts. The value can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP
distinguished name. 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 "*" will cause the entire VDB to be sent. The default for this parameter will be 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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.

**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 normally being the default. 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, 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.

**TYPE**
Specifies the type of message to be sent by selecting the processor used to generate the outgoing message. 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 may be specific to a third party recipient. Incoming messages are processed in the same manner as Unknown. The contents of a BytesMessage will be 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 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.

- **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.

or the class name of a user defined message processor.
The following table indicates which PUT_type parameters can be specified with which message types:

<table>
<thead>
<tr>
<th>PUT_BOOLAEN</th>
<th>BytesMessage</th>
<th>TextMessage</th>
<th>MapMessage</th>
<th>StreamMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_BYTE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_BYTES</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_CHAR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_DOUBLE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_FLOAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_INT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_LONG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_SHORT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PUT_STRING</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**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” which means an Info*Engine request will be 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).
EXAMPLE

The following Emit-Event example is contained in the EmitEvent.jsp file located in the infoengine/jsp/examples/weswebobjects directory. This example emits a specified event to be handled by any subscribers.

<%@page language="java" session="false"
    errorPage="../IEError.jsp"%>
<%@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 would 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 status of 0 will be returned with a message relaying successful subscription to the event. When the execution of the webject completes unsuccessfully, an appropriate exception is thrown.

*Note:* This webject can be used only after your environment has been set up to use the Web Event Service.

**SYNTAX**

```xml
<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**

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td></td>
<td>DBUSER</td>
</tr>
<tr>
<td>EXECUTE_TASK</td>
<td></td>
<td>GROUP_OUT</td>
</tr>
<tr>
<td>FAILURE_TASK</td>
<td></td>
<td>PASSWD</td>
</tr>
<tr>
<td>GROUP_IN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DBUSER
Specifies the user name 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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 user name when connecting to the MOM, and an anonymous connection will be attempted.

This parameter is optional.

EVENT
Indicates the LDAP distinguished name of the event for which to listen. It is suggested that event names be of the syntax <event_name>@<domain> to give additional meaning to the name and avoid name conflicts. The value can be an LDAP distinguished name relative to a configured baseURI or a fully qualified LDAP distinguished name. 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 a GROUP_IN is not supplied.

FAILURE_TASK
Indicates the task to be executed in the event of failure executing EXECUTE_TASK. This parameter can only be specified if no GROUP_IN is specified. The default for this parameter will be 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 columns EVENT and EXECUTE_TASK. It can also have a FAILURE_TASK column. A subscription will occur 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 further information see the section titled Credentials Mapping for MOMs in this User's Guide. For additional information about credentials mapping, see the section titled Authentication through Credentials Mapping.

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.

EXAMPLE

The following Subscribe-Event example is contained in the SubscribeEvent.jsp file located in the infoengine/jsp/examples/weswebects directory. This example subscribes the current Info*Engine process to the specified event.

```jsp
<%@page language="java" session="false"
errorPage="../IEError.jsp"%>
<%@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 would need to provide a form where the dbuser, passwd, event, group_in, and group_out variables are identified.
Unsubscribe-Event

DESCRIPTION

Causes the 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 status of 0 will be returned with a message relaying successful un-subscription from the event. When the execution of the webject completes unsuccessfully, an appropriate exception is thrown.

Note: This webject can be used only after you environment has been set up to use the Web Event Service.

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

<table>
<thead>
<tr>
<th>Required</th>
<th>Select</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td></td>
<td>GROUP_OUT</td>
</tr>
</tbody>
</table>

EVENT

Specifies the LDAP distinguished name of the event from which to unsubscribe. This parameter is required.

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.
EXAMPLE

The following Unsubscribe-Event example is contained in the UnsubscribeEvent.jsp file located in the infoengine/jsp/examples/weswebobjects directory. This example subscribes the current Info*Engine process to the specified event.

```java
<%@page language="java" session="false"
    errorPage="../IEError.jsp"%>
<%@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 would need to provide a form where the event and group_out variables are identified.
Symbols

@COOKIE group, 2-14, 8-13
@FORM group, 2-14, 6-5, 8-13
@SERVER group, 2-14, 8-13

A

ACT type, 3-12, 4-3
action webjects, 2-4
adapter
   connections, 1-14
   co-resident, 1-15
   distributed, 1-16
   native, 1-14
   non-native, 1-14
   retrieving data, 2-13
   sending data, 2-13
   webjects, 3-7, 3-11
ADM type, 3-12
ADM webjects, 10-3
administrative webjects, 2-4, 10-3
ampersand character, 8-13
apostrophe character, 8-13
application, custom, 7-1
Apply-XSL webject, 9-3
authenticate tag, 6-22, 6-24, 8-14
authentication
   credentials mapping, 6-25
   mapping, 6-25
   servlet-based, 6-22
types, 6-22
Web server, 6-22
Auth-Map group, 6-26
Auth-User value, 2-14, 6-26, 8-26

B

Binary Large Objects, 6-13
BLOB_COUNT parameter, 6-17
BLOBs, 6-13
bookstore example
   introduction, 7-6
   MOM topics, 7-47
   restocking bookstore, 7-47
   setup, 7-7
   using example, 7-13
Browse-Queue webject, 10-105

C

capitalization
   JSP pages, 3-4
tasks, 4-11
Change-Group webject, 10-9
CLASS parameter, 4-12, 4-13
comments
   JSP pages, 3-4
tasks, 4-11
Concat-Groups webject, 10-11
configuring
   bookstore JDBC adapter, 7-7
   bookstore MOM, 7-8
context groups, 2-14, 6-5, 8-13
COOKIE group, 2-14, 8-13
Copy-Group webject, 10-13
copying data groups, 2-4, 10-13
co-resident
   adapters, 1-15
Create-Group webject, 10-16
Create-Object webject, 10-112
createObjects tag, 8-17
creating
   data groups, 2-4, 2-6, 10-16
   credentials mapping, 6-25, 10-94
custom applications, 7-1
custom tags, 8-2
custom webjects, 2-4, 6-31, 7-42

D
data
   copying, 2-4
creating, 2-4
   manipulating, 2-4, 10-6
   searching, 2-4
   updating, 2-4
DBUSER parameters, 6-25
declarations, 8-6
deleteObject tag, 8-19
Delete-Object webject, 10-119
Delete-Results webject, 10-122
Diff-Groups webject, 10-23
directives, 8-7
Dispatch-Tasks webject, 10-79
display
  webjects, 2-3, 2-7, 2-10, 2-12, 3-13
display webjects
  HTML, 9-2
  overview, 9-1
Display-Object webject
  HTML, 7-20
displayResource tag, 7-20, 8-20
Display-Resource webject, 7-19, 9-23
Display-Selection webject
  HTML, 9-25
Display-Table webject
  HTML, 2-7, 2-10, 2-12, 2-15, 9-37
Display-Value webject
  HTML, 9-44
Display-XML webject, 9-54
distributed
  adapters, 1-16
Documentation conventions, xvii
Do-SQL webject, 7-18, 7-31
double quote character, 8-13
double-byte characters, 6-39
downloading files, 6-13
DSP type, 3-13

E

Echo-Request webject, 9-56
e-mail, 1-8
E-Mail Broker, 5-9
  architecture, 1-8
Emit-Event webject, 10-150
error page, 3-21
exceptions, 3-21, 5-4
executing tasks, 3-23, 4-7, 5-3
expressions, 8-5
EXT type, 3-12, 4-3, 6-31
external custom webjects, 2-4, 6-31
Extract-Group webject, 10-27

F

failure
  processing, 3-19, 8-22
tag, 3-19, 8-22
file location, 3-2, 3-16, 4-6
files, BLOBs, 6-13
forEach tag, 7-23, 8-24
FORM group, 2-14, 6-5, 8-13
Format-Group webject, 10-29

G

gateway
  architecture, 1-14
Generate-WSDL webject, 10-85
Get-Properties webject, 10-88
Get-Resource webject, 10-90
getAddress tag, 7-33, 8-26
Get-Statistics webject, 10-4
getValue tag, 7-23, 8-28
greater than character, 8-13
group webjects, 10-6
GROUP_OUT parameter, 4-13
groups
  changing attributes, 10-9
  concatenating, 10-11
  context, 2-14, 6-5, 8-13
  copying, 10-13
  creating, 10-16
difference between, 10-23, 10-75
  extracting attributes, 10-27
  formatting, 10-29
  intersecting, 10-32
  joining, 10-35
  manipulating, 2-8, 2-11
  merging, 10-41
  overview, 2-5
  registering metadata, 10-50
  returning, 10-44
  sorting, 10-55
  subset, 10-61
  substitution, 6-8
  summarizing, 10-64
  translating, 10-66
  union, 10-72
  webjects, 2-4, 2-13, 10-6
  XML output, 4-13, 9-54
GRP type, 3-12, 4-3
GRP webjects, 10-6

H

Heading
  1, 5-23
HTML, 2-2, 3-13, 9-2
IEException class, 5-4
Include-Task webject, 4-16
Info*Engine
    adapters, 1-14
    architecture, 1-2
E-Mail Broker, 1-8
gateways, 1-14
JSP pages, 2-2
Naming Service, 1-13
Service Access Kit, 5-2
tags, 3-6, 8-10
tasks, 2-2, 4-2
templates, 2-2
Info*Engine J2EE Connector, 6-51
init tag, 8-30
Intersect-Groups webject, 10-32

J
J2EE Connector
    See Info*Engine J2EE Connector
Java expressions, 8-5
JavaServer Pages, 2-2, 3-2
JDBC adapter, 7-7
JDBC query
    advanced, 7-18
    simple, 7-21, 7-38
Join-Groups webject, 7-22, 7-29, 10-35
JSP pages
    accessing, 3-3
capitalization, 3-4
comments, 3-4
common tags, 8-2
creating, 3-4
eexample, 3-2
introduction, 3-2
parts, 3-4
storing, 3-2
using, 2-2
XML, 2-3

L
LDAP
    search base, 3-16
    URL, 3-15
less than character, 8-13
listObjects tag, 8-32
locating components
    architecture, 1-13
mailbox, 1-8
management webjects, 2-4, 10-78
managing
    BLOBs, 2-16
    large groups, 2-15
manipulating groups, 2-4, 2-8, 2-11, 10-6
Map-Credentials webject, 10-94
Merge-Groups webject, 2-9, 10-41
message webjects, 2-4, 10-104
metadata, 4-15, 5-6, 6-3, 9-54, 10-50, 10-82
Metaphase adapters, 1-14
MGT type, 3-12, 4-3
MGT webjects, 10-78
MOM
    bookstore configuration, 7-8
    bookstore example, 7-47
MSG type, 3-12, 4-3
MSG webjects, 10-104

N
Naming Service
    architecture, 1-13
native adapters, 1-14
nesting
    tasks, 4-16
non-native adapters, 1-14

O
OBJ type, 3-12, 4-3
Object-Display webject
    HTML, 9-11
Oracle adapters, 1-14

P
page directive, 8-7
parallel tag, 3-18, 8-34
param tag, 8-36
parameters
    substitution, 6-2
tag, 8-36
PASSWD parameters, 6-25
POST request, 2-14, 8-26

Q
query webjects, 2-4, 2-6, 2-8
Query-Object webject, 10-126
queryObjects tag, 8-40
Query-Objects webject, 2-6, 2-8, 7-22, 7-23, 7-28
Query-Results webject, 10-135
Queue-Task webject, 10-131
quotation marks, 8-12

R
rblInfo files
  bookstore, 7-12
  reference, 8-20
Reload-Properties webject, 10-5
requests, 1-6, 1-8
resetService tag, 8-42
resource bundle
  bookstore, 7-12
  reference, 8-20
result sets, 7-19
Return-Groups webject, 2-13, 10-44

S
SAK, 5-2, 7-2
scriptlets, 7-19, 7-20, 7-39, 8-3
search base, 3-16
searching for data, 2-4
Send-Mail webject, 7-48, 10-139
servelet-based authentication, 6-22
SERVER group, 2-14, 8-13
server-based authentication, 6-25
Service Access Kit, 5-2, 7-2
Set-Metadata webject, 10-50
Simple Object Access Protocol
  see SOAP
SOAP, 5-11
  comments, 5-16
  sample SOAP client, 5-13
  SOAP RPC servlet, 5-11
  writing tasks, 5-14
SOAP clients
  J2EE, 6-67
  standalone Java client, 6-56
SOAP RPC servlet, 5-11
Sort-Group webject, 7-37, 10-55
special characters, 8-13
starting tasks, 3-23, 4-7, 5-3
stylesheet, 9-3, 10-66
Subscribe-Event webject, 7-48, 10-157
Subscribe-Queue webject, 10-144
Subset-Group webject, 10-61
substituting parameter values, 6-2
success
  processing, 3-19, 8-45
tag, 3-19, 8-45
Summarize-Groups webject, 10-64
syntax
  task, 4-5
task webjects, 3-4

T
tag library, 3-6, 4-10
taglib directive, 8-9
tags
  authenticate, 8-14
  createObjects, 8-17
  deleteObject, 8-19
  displayResource, 7-20, 8-20
  failure, 8-22
  forEach, 7-23, 8-24
  getService, 7-33, 8-26
  getValue, 7-23, 8-28
  Info*Engine, 3-6, 8-10
  init, 8-30
  JSP, 8-2
  listObjects, 8-32
  parallel, 8-34
  param, 8-36
  queryObjects, 8-40
  resetService, 8-42
  success, 8-45
  task, 8-46
  unit, 8-51
  updateObjects, 8-54
  webject, 8-57
task tag, 4-17, 8-46
tasks
  authoring, 4-3
  creating, 4-6
  executing, 3-23, 4-7, 5-3
  extension, 4-6
  introduction, 2-2
  location, 4-6
  nesting, 4-16
  parts, 4-5
  syntax, 4-5
tag syntax, 8-46
  webjects, 3-12, 10-1
  XML output, 4-12
Technical support, xvi
templates, 2-2
Throw-Exception webject, 3-22, 7-46, 10-98
Translate-Group webject, 10-66
type attribute, 10-82
Union-Groups webject, 10-72
unit tag, 3-19, 3-22, 8-51
Unknown-Class-Name, 4-13
Unsubscribe-Queue webject, 10-147
Unsubscribe-Tasks webject, 10-160
updateObjects tag, 8-54
uploading data groups, 2-4
uploading files, 6-13
URI
  absolute, 3-15
  relative, 3-15
  specifying, 3-15
URL
  absolute, 3-15
  E-Mail Broker, 5-9
  LDAP, 3-15
  relative, 3-15
  specifying, 3-15
user
  request, 1-6, 1-8
  validation, 6-24
user authentication
  credentials mapping, 6-25
  servlet-based, 6-22
  types, 6-22
  Web server, 6-22
validating users, 6-24
VDB groups
  changing attributes, 10-9
  concatenating, 10-11
  context, 2-14, 6-5, 8-13
  copying, 10-13
  creating, 10-16
  difference between, 10-23, 10-75
  extracting attributes, 10-27
  formatting, 10-29
  intersecting, 10-32
  joining, 10-35
  manipulating, 2-8, 2-11
  merging, 10-41
  overview, 2-5
  registering metadata, 10-50
  returning, 10-44
  sorting, 10-55
  subset, 10-61
  substitution, 6-8
  summarizing, 10-64
  translating, 10-66
  union, 10-72
  XML output, 4-13
wc:COLLECTION element, 4-12, 4-13
wc:INSTANCE element, 4-12, 4-14
wc:Meta element, 4-15
Web Event Service webjects, 2-4, 10-149
Web server authentication, 6-22
webject
  action, 2-4
  adapter, 3-7, 3-11
  administrative, 2-4, 10-3
  custom, 2-4, 6-31, 7-42
  definition, 2-3
  display, 2-3, 2-7, 2-10, 2-12, 3-13, 9-1
  group, 2-4, 2-13, 10-6
  management, 2-4, 10-78
  message, 2-4, 10-104
  query, 2-4, 2-6, 2-8
  tag, 8-57
  task, 3-12, 10-1
  types, 4-3
  Web Event Service, 2-4, 10-149
  WES type, 3-12, 4-3
  WES webjects, 2-4, 10-149
  Write-Log webject, 10-101
XML
  declaration, 4-12
  formatted page, 9-54
  JSP pages, 2-3
  output, 4-12
  tasks, 2-2
XOR-Groups webject, 10-75
XSL stylesheet, 9-3, 10-66