LISTEN.
THINK.
SOLVE.SM

INTEGRATED PRODUCTION & PERFORMANCE SUITE

Production Management

FactoryTalk® ProductionCentre

INTEGRATE WEB SERVICES
RELEASE 10.4
SETUP AND CODE EXAMPLES

PUBLICATION PCIWS-IN104A-EN-E

ALLEN-BRADLEY • ROCKWELL SOFTWARE **Rockwell Automation**

Table of Contents

Read Me First7

Audience and Expectations.......... 8
Other Information Sources.......... 8
  Related Documentation.......... 8
  Solutions and Technical Support.......... 9

Chapter 1 Overview and Setting Up the Environment.......... 11

Integrate Web Services Overview.......... 12
Communication Mechanisms.......... 13
  EJB Client Application.......... 13
  Web Services Client Application.......... 13
Choosing Manually-generated or Pre-generated Proxies.......... 14
  Default Values for New Objects.......... 14
  Object Handling.......... 15
  Constants.......... 15
  User Authentication.......... 15
Preparing the Development Environment for Java.......... 15
Creating Proxies from the Web Services.......... 16
Using .NET Clients with WebLogic Platform.......... 17
Using .NET Clients with JBoss Platform.......... 18
Upgrading Plant Operations and Web Services.......... 19
Description of Each Package.......... 19
  Constants.......... 20
  Filtering.......... 20
  Dataobjects.......... 20
  Exceptions.......... 21
  Services.......... 21

Utility 22
Version 23

Chapter 2 Using Integrate Web Services 25
Creating the Required Components of an Application 26
Import the Classes 26
Create an Instance of the ProxyFactory 26
Log in to the Plant Operations Application Server 27
Use Try-Catch Blocks 28
Log off of the Application Server 28
Data Transfer Flag 28
Modification Flag 29
DTrxInfo Class 31
Sample Code for Each Services Package 31
AccessControl 31
BoxHandling 32
CarrierHandling 32
ClientUtility 32
Compatibility 33
EquipmentHandling 33
FlowLotHandling 34
LotHandling 34
ObjectRetrieval 35
ObjectRevisionRetrieval 35
ObjectStorage 36
ProductionLineHandling 36
Security 37
System 37
UnitHandling 38
WorkCenterHandling 38
WorkFlowHandling 39
WorkOrderHandling 39

Chapter 3 Integrate Web Services Code Examples 41
Working with DateTime Values 43
Retrieving Objects from the Database 45
Retrieving Objects by Object Name 45
Retrieving Objects by Key 46
Retrieving Objects Using a Filter 46

Filtering By Date Values 47
Retrieving Objects Using SQL 48
Retrieving Object Attributes 49
Changing Object Attributes 51
Automatically Committed Changes 51
Manually Committed Changes 51
Performing and Updating Data Collection 53
StandAlone Data Collection 56
Modifying Data Collection 56
Starting a Unit at a Route Step 58
Completing a Unit at a Route Step 58
Collecting Test Data 59
Creating a Work Order 61
Creating an Account 62
Linking an Account to a Work Order 63
Tracking Equipment on a Resource Route 64
Creating a Part 65
Creating a BOM 65
Modifying the BOMItem Quantity 66
Linking a BOM to a Part 67
Performing Consumption 67
Managing Application Tables 69
Create a New Row to Store Data 70
Update an Existing Row 72
Creating a Master-Detail AT Relationship 73

Index75

Read Me First

In this chapter

- ❑ **Audience and Expectations 8**
- ❑ **Other Information Sources 8**

  Related Documentation 8
  Solutions and Technical Support 9

This section introduces the reader to the following topics:

- "Audience and Expectations"
- "Other Information Sources"

Audience and Expectations

This book is intended for experienced professionals who understand their company's business needs and the technical terms used in this guide. We expect the user to be experienced with the following:

- Programming using a Web Service
- Writing code in the programming language of choice (e.g. Java)

This guide assumes that the supporting network equipment and software, including Plant Operations, have been installed.

Other Information Sources

In addition to this guide, the following additional information sources are available.

Related Documentation

The following table lists other available documents related to Integrate Web Services.

Table 1 Related Documents

Title	Purpose
Required third party software installation and configuration, such as *Visual Studio.net Installation Guide* or *Microsoft Visual Studio Tool*	Installing development software or developing an application.
Integrate Web Services API Online Help located at*: http://<*serverName*>:<*portName*>/PlantOperations/docs/help/ws/index.html -or- <*ws_download*>\docs\api	Online Help for Integrate Web Services API.
FactoryTalk® ProductionCentre (called FTPC hereafter) Pre-generated Proxies API located at*: <*ws_download*>\docs\proxies	API description for ProxyFactory class.
Integrate Web Services Utility API located at*: <*ws_download*>\docs\utils	API description for the DataConverter Class.

Title	Purpose
Plant Operations Release Notes located at*: http://<*serverName*>:<*portName*>/ PlantOperations/docs/help/pd/index.htm	Release Notes for Plant Operations.
Readme.html located at* <*ws_download*>\	This contains information about compiling and running a Web Services client application or EJB client application when using the Pre-generated Proxies for Java.
* *<ws_download> is the location where you downloaded and installed the Web Services.* <*serverName*>:<*portName*> is the Plant Operations application server name and HTTP port number.	

Solutions and Technical Support

The FTPC online knowledge base contains information and articles related to Integrate Web Services. Contact Rockwell Automation for a username and password to access the knowledge base. To search for solutions related to Integrate Web Services, enter the keyword **wsIntegrate**.

Chapter 1

Overview and Setting Up the Environment

In this chapter

- **Integrate Web Services Overview 12**
- **Communication Mechanisms 13**

  EJB Client Application 13
  Web Services Client Application 13
- **Creating Proxies from the Web Services 16**
- **Using .NET Clients with WebLogic Platform 17**
- **Choosing Manually-generated or Pre-generated Proxies 14**

  Preparing the Development Environment for Java 15
  Upgrading Plant Operations and Web Services 19

Integrate Web Services Overview

There are two components to the Integrate Web Services product. At its core, is a set of XML Web Services provided by the Plant Operations application server to expose the API functionality.

It also includes a second component, a set of pre-generated proxies for Java, that makes it easier to configure and develop applications written in the Java programming language. If you do not use the pre-generated proxies, you must manually generate the proxies from wsdl files on the application server.

Although both the pre-generated and manually-generated proxies accomplish the same purpose, there are slight differences in their capabilities. The pre-generated proxies include some capabilities that make the mechanics of building an application easier. This document will describe how to get started using the pre-generated proxy for Java. If you want to manually generate the proxies, contact Rockwell Automation Technical Support for more information.

Integrate Web Services conforms to the following industry-accepted technology standards:

- Web Services Interoperability (WS-1) Basic Profile 1.0
- Simple Object Access Protocol (SOAP) 1.1
- Web Services Definition Language (WSDL) 1.1
- SOAP with Attachments API for Java (SAAJ) 1.1
- RMI Communication (Remote Method Invocation/Internet Inter-ORB Protocol)
- Apache Axis - Your java client application should use the Web Services jar files that are compatible with the application server version you are running (i.e., jar files provided by IBM, JBoss/Red Hat, or BEA).

  For example, JBoss 4.0.3 SP1 uses axis-ws4ee.jar. Because the Axis source code is pre-packaged in this file, the Apache Axis version may not match what is posted on the Apache Axis web site. Therefore, do not automatically go to the Apache Axis web site and download the latest jar files. Instead, get the necessary client-side jar files from your application server vendor for the specific version/build of the application server that is installed.

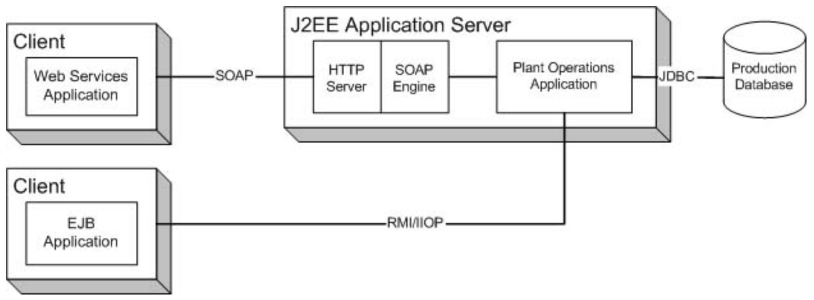
Integrate Web Services is built primarily using these two technologies:

- SOAP (Simple Object Access Protocol): SOAP is the transport protocol used by XML Web Services to expose useful functionality to web users.
- RMI-IIOP provides you with a powerful environment in which to communicate with the Plant Operations EJBs directly. This technology can be used in certain circumstances only (see “Communication Mechanisms” on page 13, for more information).

- WSDL (Web Services Description Language): XML Web Services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is provided in an XML document called a WSDL document.

The following figure shows the Integrate Web Services architecture.

Figure 1-1: Integrate Web Services Architecture



Integrate Web Services is intended to provide you with a generic and standard interface to integrate Plant Operations with third-party applications like ERP and Supply Chain systems. Integrate Web Services can also be used to create custom client applications that can communicate with the Plant Operations middle tier without the overhead of the scripting environment.

Communication Mechanisms

Prior to developing your application, you must decide what communication mechanism your application will use. This choice governs whether you create an Enterprise Java Bean (EJB) client application or a Web Services client application.

EJB Client Application

If you are using Java to build your application and your application will run over a LAN, you have the choice of creating an EJB client application, rather than a Web Services application. If you choose this option, you must use RMI/IIOP to communicate directly to the Plant Operations Enterprise Java Beans.

The advantage of the EJB client application is that it communicates directly to the EJBs on the Plant Operations application server using RMI/IIOP, which results in a faster application.

RMI communication is only supported with the pre-generated proxies.

Web Services Client Application

You must create a Web Services client application if you are connecting over a Wide Area Network (WAN), the client application is written in a language other than Java, or if you are using SOAP as the communication protocol.

A Web Services client application uses the SOAP protocol to communicate to the Web Services in the Plant Operations application server. The Web Services then communicates to the EJBs on the Plant Operations application server.

Choosing Manually-generated or Pre-generated Proxies

If you are programming in Java, you should use the pre-generated proxies for Java instead of manually-generated proxies from the Web Services. The pre-generated proxies for Java come packaged with Plant Operations and can be downloaded from the application server. The Plant Operations application server is installed with a set of pre-generated proxies for Java that can be used to build an application using the Java programming language. This set of pre-generated proxies provides the following benefits:

- when you create and save a new object, certain properties are set to default values if they are not specified.
- allows more seamless object handling compared to using the proxies that were manually generated from WSDL files.
- includes constant definitions for certain object attributes.
- simplifies user authentication.

If you are programming in any other language that supports XML Web Services, you must manually generate the proxies from the Web Services (wsdl files). Contact your Rockwell Automation Technical Support for guidance about implementation. The mechanism by which you implement your application varies slightly depending on which proxy you use.

Default Values for New Objects

Certain Plant Operations objects have properties that require a value. Using the pre-generated proxies for Java, properties that require a value will be set to a default when one is not defined before the object is saved. This behavior is similar to the Process Designer API that contains logic to set these properties to a default value. You do not need to set as many properties prior to saving the object, provided you use the default values.

When using the manually-generated proxies, if you do not specify a value for a required property, the middleware zeros-out the memory for the property and stores the resulting value. For example:

- String properties will be null
- int and long properties will be 0

If the value for that property is invalid, the middleware will throw an exception when the object is saved. In the Integrate Web Services API online help, properties that require a value can be identified by the presence of a ‘Default Value:’ field in the description.

For example, if you are creating a new part and want to know the default value for the bomTrackedMode property, then perform the following in the Integrate Web Services API online help:

1. Under the Classes list, select DPart.
2. In the Field Summary table, select bomTrackedMode.

   A description of the bomTrackedMode property is displayed along with the default value and the possible values. The default value is EXPLODED. When using the pre-generated proxies, if this is the value that you want for the part, then you do not need to set this property. When using the manually-generated proxies, you must set this property to one of either the Accepted or Default Values specified in the Integrate Web Services API online help.

Object Handling

The Integrate Web Services API is separated into multiple services that each are defined in a different XML namespace. The pre-generated proxies for Java merges all services into a single namespace when creating the proxies. This provides better object handling. If you manually generate the proxies, contact your Rockwell Automation Technical Support for guidance about implementation.

Constants

Some API methods take a constant as an argument. Using the pre-generated proxies for Java, you can specify a constant using a textual representation of the constant value. Without this capability, you will need to use the Integrate Web Services online help to identify the numeric representation of the attribute value.

User Authentication

The pre-generated proxies for Java contain a ProxyFactory class that simplifies user authentication. This class does not exist in manually-generated proxies.

Preparing the Development Environment for Java

The following describes how to prepare your development environment when you choose to use the Plant Operations pre-generated proxies for Java and support files. All of the following steps must be performed.

The code examples in this document are written in Java. The steps and syntax may be different for your development environment. If you want to use manually-generated proxies, contact Rockwell Automation Technical Support for additional information.

Step 1. Download the Plant Operations pre-generated proxies for Java and support files. The following steps describe how to download them from your application server:

a. Navigate to http://<*serverName*>:<*portName*>/PlantOperations/index.htm.

b. Click the “Java Web Service Proxy” link.

c. Click Save.

d. In the Save As dialog, select a location to save the file, PlantOpsClientSDK.zip, and then click [Save].

e. Navigate to the .zip file and extract the files.

The extract directory will contain the pre-generated proxies for Java (PlantOpsJavaProxies-<*application_server*>.jar) and the /lib folder, containing other required support files, where <*application_server*> is the name of the J2EE application server you are using to run Plant Operations.

Step 2. Set up the classpath to the PlantOpsJavaProxies-<app_server>.jar file, where <app_server> identifies the J2EE application server type that Plant Operations is running under.

Step 3. Set up the classpath all JAR files in the /lib directory.

Step 4. If you are using JBoss or WebLogic as the J2EE application server, when you run the application, you must set a Java system property value that identifies the application server. You can set this argument when you run the java.exe.

```
java -Dcom.datasweep.plantops.j2eevendor=JBoss
java -Dcom.datasweep.plantops.j2eevendor=WebLogic
```

Creating Proxies from the Web Services

Your application will require proxies that are generated based on the Plant Operations WSDL files. Manually-generated proxies do not have the same benefits as the pre-generated proxies. See the section “Choosing Manually-generated or Pre-generated Proxies” on page 14 for more information about these differences.

To generate the proxies manually:

Step 1. Determine which of the following services you need for your application.

Step 2. Complete the steps specific to your development tool to reference the url to the wsdl file. The url is case-sensitive. The url always begins with the following: http://<*serverName*>:<*portName*>/PlantOperationsCore/services/

Te URL ends with one of the following wsdl file names:

Table 1-1 WSDL names

AccessControl?wsdl	ObjectRevisionRetrieval?wsdl
BoxHandling?wsdl	ObjectStorage?wsdl
CarrierHandling?wsdl	ProductionLineHandling?wsdl
ClientUtility?wsdl	Security?wsdl
Compatibility?wsdl	System?wsdl
EquipmentHandling?wsdl	UnitHandling?wsdl
FlowLotHandling?wsdl	WorkCenterHandling?wsdl
LotHandling?wsdl	WorkFlowHandling?wsdl
ObjectRetrieval?wsdl	WorkOrderHandling?wsdl

For example:

http://prod01:8080/PlantOperationsCore/services/System?wsdl

Step 3. You may be prompted to enter the user name and password for the Default Realm or Discovery Credential. Use a Plant Operations administrative user name and password.

Using .NET Clients with WebLogic Platform

To allow .NET clients to successfully connect to Plant Operations running on the WebLogic application server, you must override the SoapHttpClientProtocol.GetWebRequest method in the client-side stub to insert the calling user name and password credentials into the HTTP request header. The following instructions describe how to do this.

Step 1. Generate the Proxy using Visual Studio to add a web reference to the selected web service. This step creates Reference.cs (C#) or Reference.vb (Visual Basic).

Step 2. Open the Reference.cs (C#) or Reference.vb (Visual Basic) file and locate the SoapHttpClientProtocol class for the client-side stub.

Step 3. Add the new GetWebRequest method override as the first member of the SoapHttpClientProtocol class.

IMPORTANT: **This code example uses C# programming language. The code required for Visual Basic .NET will be different.**

```
protected override System.Net.WebRequest GetWebRequest(Uri
uri)
{
System.Net.HttpWebRequest request =
(System.Net.HttpWebRequest)base.GetWebRequest (uri);
System.Net.NetworkCredential credentials =
(System.Net.NetworkCredential)request.Credentials;
string username = credentials.UserName;
string password = credentials.Password;
string auth = username + ":" + password;
byte[] binaryData = new Byte[auth.Length];
binaryData = System.Text.Encoding.UTF8.GetBytes(auth);
auth = Convert.ToBase64String(binaryData);
auth = "Basic " + auth;
request.Headers["AUTHORIZATION"] = auth;
return request;
}
```

Step 4. Check that

- the DSAuthenticator custom security provider is configured with a Control Flag of REQUIRED. This configration is described in the *Plant Operations Server Installation Guide for WebLogic*.
- all other security providers are configured with a control flag of OPTIONAL.

Using .NET Clients with JBoss Platform

To allow .NET clients to successfully connect to Plant Operations running on the JBoss application server, you must override the HTTP version that is used by default by JBoss.

IMPORTANT: **This procedure is not necessary if the .NET client code is using the provided .Net/COM proxy factory because the override is done on the client-side by including information in the HTTP header.**

The following instructions describe how to do this.

Step 1. Stop your JBoss application server.

Step 2. Open the server.xml file in a text editor.

- ▶ For a JBoss Stand-Alone installation, this file is located at <*JBoss_install_directory*>\jboss\server\datasweepConfig\deploy\jbosswe b-tomcat55.sar.

- For a JBoss Advanced installation, this file is located at <*JBoss_install_directory*>\jboss\server\all\deploy\jbossweb-tomcat55.sar.

Step 3. Add the following line to the HTTP Connector tag:

```
restrictedUserAgents="^.*MS Web Services Client Protocol.*$"
```

The HTTP Connector tag should resemble the following:

```
<!-- A HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" address="${jboss.bind.address}"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" redirectPort="8443" acceptCount="100"
connectionTimeout="20000" disableUploadTimeout="true"
restrictedUserAgents="^.*MS Web Services Client Protocol.*$"
/>
```

Step 4. Restart your JBoss application server.

Step 5. Start your .NET application.

Upgrading Plant Operations and Web Services

When you upgrade Plant Operations you must also upgrade the proxies, regardless of whether you are using manually-generated or pre-generated proxies. To upgrade the pre-generated proxies for Java, archive the current proxies, then download and extract the PlantOpsClientSDK.zip file from the updated instance of the Plant Operations application server. If you download and extract the files to the same location as they were originally located, then you do not need to do anything else. If you download and extract the files to a different location, then you will need to repeat the steps to create the reference to the new library location.

To upgrade the manually-generated proxies, you must either re-generate the proxies or use the update capabilities available in the development environment. See the product documentation for your development tool for instructions about how to do this.

IMPORTANT: **If you are upgrading from a previous release, please see the Plant Operations Release Notes and Plant Operations Upgrade Guide for a list of compatability issues.**

Description of Each Package

The following section describes each of the packages that appear in the pre-generated proxies for Java. You may see classes in the IDE that are not described in the API online help. If no documentation is available for a specific class, that class is used internally only and is not supported for use in a client application. The following section lists each package and the location of the documentation for the package.

Constants

The com.datasweep.plantops.common.constants package defines all constants used for all of the Plant Operations data objects. The constants in this package are only available if you use the pre-generated proxies for Java. If you manually generate the proxies, you will not have these predefined constants available. You must use the online help to determine the integer values when specifying attribute types for objects and arguments.

Documentation can be found in the extract directory at:
<ws_download>\docs\api\index.html

Filtering

The com.datasweep.plantops.common.constants.filtering package defines the class fields that are available when you are using the DFilter class to retrieve objects from the Plant Operations application server. The DFilter object is a class within the com.datasweep.plantops.common.dataobjects package.

The constants in this package are only available if you use the pre-generated proxies for Java. If you manually generate the proxies, you will not have available any predefined filtering constants. You must use the online help to determine the integer value when specifying attribute types for the DFilter object.

Documentation for this package can be found in the extract directory at:
<ws_download>\docs\api\index.html

Dataobjects

The com.datasweep.plantops.common.dataobjects package contains the classes that represent the Plant Operations objects. The methods in these classes are used only to set and get values for object attributes. It does not include methods that allow you to perform transactions on an object, such as saving the object. The classes and methods in this package are available when you use the pre-generated proxies for Java and when you manually generate the proxies. To save an object you must use the save method found in the ObjectStorage service for most classes. The save method for the following classes are in the respective Handling services:

- UnitHandling contains the save(...) method for the DUnit.
- BoxHandling contains the save(...) method for the DBox.
- Security contains the saveCrew(...), saveGroup(...), and saveUser(...) methods.
- FlowLotHandling contains the save(...) method for the DFlowLot.

- LotHandling contains the save(...) method for any DLot other than DFlowLot.
- WorkOrderHandling contains the save(...) method for the DWorkOrder.

Documentation can be found in the extract directory at:
<ws_download>\docs\api\index.html

Exceptions

The com.datasweep.plantops.common.exceptions package contains all Plant Operations-specific exception classes. The API in this package is available when you use the pre-generated proxies for Java and when you manually generate the proxies.

Documentation can be found in the extract directory at:
<ws_download>\docs\api\index.html

Services

The com.datasweep.plantops.common.services package contains the classes and methods that allow you to:

- Get objects from the database.
- Perform transactions on runtime objects.
- Perform the Access Control capabilities available in the Process Designer.
- Perform auxiliary capabilities such as execute SQL statements, send email, and get system information.
- Delete objects from the database.
- Save objects.
- Manage users and crews.

The API in this package is available when you use the pre-generated proxies for Java and when you manually generate the proxies. Documentation can be found in the extract directory at: <ws_download>\docs\api\index.html

Each of the classes in this package is represented by a separate web service (*.wsdl) file that is described as follows:

- AccessControl: this class allows you to perform the access control capabilities that are similarly available in Process Designer.
- BoxHandling: This class allows you perform transactions on a DBox object.
- CarrierHandling: This class allows you to perform transactions on a DCarrier object.

- ClientUtility: Contains methods to send an email, write an entry to the trx_base table, and execute SQL statements or stored procedures on the production and ODS databases.
- Compatibility: Contains methods that support backward compatibility transactions. Do not use this service.
- EquipmentHandling: This class allows you to perform transactions on a DEquipment object.
- FlowLotHandling: This class allows you to perform transactions on a DFlowLot object.
- LotHandling: This class allows you to perform transactions on a DLot object of any type other than a DFlowLot.
- ObjectRetrieval: This class is used to get the current version of objects from the database and to check for the presence of certain object attributes.
- ObjectRevisionRetrieval: Methods in this class are used primarily to get previous versions of objects from the database and to check for the presence of certain object attributes.
- ObjectStorage: Methods in this class are used primarily to save and delete objects from the database.
- ProductionLineHandling: This class allows you to perform transactions on a DProductionLine object.
- Security: Methods in this class allow you to manage DUsers and DCrews.
- System: Methods in this class allow you to change a user’s password and get database information.
- UnitHandling: This class allows you to perform transactions on a DUnit object.
- WorkCenterHandling: This class allows you to perform transactions on a DWorkCenter object.
- WorkFlowHandling: This class allows you to perform transactions on a DWorkFlow object.
- WorkOrderHandling: This class allows you to perform transactions on a DWorkOrder object.

Utility

The com.datasweep.plantops.common.utility package contains the DataConverter class that provides methods to convert data to and from a String. Documentation can be found in the extract directory at: <ws_download>\docs\utils\index.html

Version

The com.datasweep.plantops.proxies.version package stores build and release information about the pre-generated proxies for Java. This package may appear in your development environment, but it is not described in the Integrate Web Services online help.

Chapter 2

Using Integrate Web Services

In this chapter

- ❑ **Creating the Required Components of an Application 26**
- ❑ **Data Transfer Flag 28**
- ❑ **Modification Flag 29**
- ❑ **DTrxInfo Class 31**
- ❑ **Sample Code for Each Services Package 31**

BoxHandling 32
CarrierHandling 32
ClientUtility 32
Compatibility 33
EquipmentHandling 33
FlowLotHandling 34
ObjectRetrieval 35
ObjectRevisionRetrieval 35
ObjectStorage 36
ProductionLineHandling 36
Security 37
System 37
UnitHandling 38
WorkCenterHandling 38
WorkFlowHandling 39
WorkOrderHandling 39

All middle tier functionality available through the Plant Operations API is available through the Integrate Web Services API. This document summarizes some of the API available from Integrate Web Services.

NOTE: This document is release independent. For specific methods, see Integrate Web Services API.

Creating the Required Components of an Application

The following section describes, in general, how to create the required components of your application when you choose to use the pre-generated proxies for Java and support files. All of the following steps must be performed in the application. This section assumes that you have completed the steps in "Preparing the Development Environment for Java" on page 15.

IMPORTANT: **The following code examples were written in Java using the pre-generated proxies for Java. The steps and syntax may be different for your environment.**

Import the Classes

You must import the required proxies and packages to your class. The following code imports the proxies and packages required to run the examples in this document.

```
import com.datasweep.plantops.proxies.ProxyFactory;
import com.datasweep.plantops.common.services.*;
import com.datasweep.plantops.common.constants.*;
import com.datasweep.plantops.common.constants.filtering.*;
import com.datasweep.plantops.common.dataobjects.*;
import com.datasweep.plantops.common.utility.*;
```

Create an Instance of the ProxyFactory

When using the pre-generated proxies for Java, you must create an instance of the ProxyFactory. You should create this instance once and reuse it throughout the application. The following lines of code establish a SOAP/HTTP connection to the application server.

```
String url = “http://testsrvr:9080”
proxyFactory = ProxyFactory.createProxyFactory(url, ProxyFac-
tory.PROTOCOL_HTTP);
```

If you pass the second argument as PROTOCOL_IIOP, your application will communicate directly to the Plant Operations EJBs via RMI/IIOP. A createProxyFactory signature also exists that takes two server URL strings as arguments, createProxyFactory(iiopURL, httpURL), that first attempts to establish an RMI/IIOP connection based on the first argument. If that fails, it then attempts to establish a SOAP/HTTP connection using the second argument. See the Integrate Web Services API Online Help for more detailed information.

If you use IIOP as the communication mechanism and the Plant Operations application server is running JBoss, the iiopURL agrument for createProxyFactory method must use 'jnp' as the protocol and not 'iiop'. for example, 'jnp://prod01' is a valid url string for the machine name 'prod01 running Plant Operations under the JBoss Application Server.

If you use IIOP as the communication mechanism and the Plant Operations application server is running WebSphere Enterprise Service Bus, you must also set the java system property `com.ibm.CORBA.ConfigURL` to

http://<host_name>:<port>/PlantOperationsCore/webStart/sas.client.props

You can do this from the command line while invoking java using the following:

```
-Dcom.ibm.CORBA.ConfigURL=http://<host_name>:<port>/PlantOp-
erationsCore/webStart/sas.client.props <class>
```

Where:

- <*host_name*> is the sytem name of the Plant Operations application server.
- <*port*> is the HTTP listener port in the Plant Operations application server.
- <*class*> is the class in your application that executes the main(..) method.

IMPORTANT: **The ProxyFactory is specific to the pre-generated proxies for Java. With manually-generated proxies, there is no equivalent class to the ProxyFactory.**

Log in to the Plant Operations Application Server

The ProxyFactory contains a login method to log in to the Plant Operations application server. The user name and password must be a valid Plant Operations user and password with the appropriate privileges to perform the capabilities in your application.

```
proxyFactory.login("userName", "password");
```

Some methods can only be executed by certain user groups. If there is a restriction, you will see the following entry in the Integrate Web Services API online help, indicating the minimum group membership required to execute the method:

Required authorization roles:

<GROUP_NAME>

Where <GROUP_NAME> is one of the five Plant Operations groups: PLANTOPS_ADMIN, PLANTOPS_DESIGNER, PLANTOPS_SUPERVISOR, PLANTOPS_OPERATOR, PLANTOPS_GUEST

Use Try-Catch Blocks

As you develop the functionality of your application, always use Try Catch blocks to handle exceptions. For the sake of simplicity, the examples shown in this document do not include error handling. The Integrate Web Services API includes a set of Plant Operations-specific Exception classes. See the Integrate Web Services API online help for more information.

Log off of the Application Server

You must always include the capability to log off of the Plant Operations application server when exiting the application, as this is not performed automatically.

```
securityproxy.logout();
proxyFactory.logout();
```

Data Transfer Flag

Transaction methods for certain classes perform the transaction, then return the object to the client. These methods take a xferType argument that allows you to specify what amount of data you want returned with the object. The argument refers to the IDataTransferTypes constant. Use this flag to specify what data to transfer to or retrieve from the database. Possible values are:

- 0: TRANSFER_NONE

  This flag applies only to runtime transactions. When you pass TRANSFER_NONE, the middleware returns a null object back to the client application when the transaction is complete.

  In the following example, the IDataTransferTypes is set to TRANSFER_NONE because we do not need the unit once the priority has been changed. The uHandler variable is the unit handling proxy.

```
uHandler.changePriority(objUnit.getKey(), 5, null, "",IData-
TransferTypes.TRANSFER_NONE, null);
```

- 1: TRANSFER_NORMAL

  Use this flag when you need values (properties) from an object after it is retrieved or saved.

  In the following example, the IDataTransferTypes is set to TRANSFER_NORMAL because after saving the new work order, we need its key. The wHandler variable is the work order handling proxy.

```
DWorkOrder objOrder1 = woHandler.saveWorkOrder(objOrder, null,
"", IDataTransferTypes.TRANSFER_NORMAL, null);
```

- 2: TRANSFER_EXTENDED

  Units, lots, flow lots, and boxes have NORMAL and EXTENDED data. EXTENDED data refers to attributes considered least commonly used by the applicable object transactions, such as billAccountKey.

  To determine if the attribute that you require is EXTENDED or NORMAL data, refer to the DUnitExtended, DLotExtended, DFlowLotExtended, and DBoxExtended classes in the Integrate Web Services API. If the attribute you require is a field of the extended class, then you must specify the TRANSFER_EXTENDED flag when you retrieve the object.

  If you need to change an object such as a unit after it is retrieved, then you must set IDataTransferTypes to EXTENDED.

  In the following example, the IDataTransferTypes is set to TRANSFER_EXTENDED because we want to set the unit's UDA value after we retrieve the unit. The objRet variable is the Object Retrieval proxy.

```
DUnit objUnit = objRet.getUnitByKey(unitKey,
IDataTransferTypes.TRANSFER_EXTENDED);
```

Modification Flag

All objects that are subclasses of the DDataObject class inherit the method setModificationFlag(ImodificationFlag). The argument refers to the IModificationFlags constant. Use this flag to specify the modification level of an object. This flag identifies the type of change that will occur when you save the object.

IMPORTANT: **Setting the modification flag does not commit the object to the database. You must call the appropriate save method, even for items flagged as DELETED.**

Possible values are:

- 0: NEW

  When you create a new object, set the flag to NEW. The object will be saved to the database when you call the appropriate save method.

  In the following example, a new part is created and flagged as NEW. The part must be flagged as NEW or it will not get saved in the database:

```
objPartNew.setModificationFlag(IModificationFlags.NEW);
long key = os.savePart(objPartNew, null, "", null);
```

- 1: UNCHANGED

When an object is retrieved from the database, it is flagged as UNCHANGED. There is no scenario where you need to explicitly set the flag to UNCHANGED.

- 2: MODIFIED

  If you retrieve an object from the database, modify it, then want to save the modified version, you must set the flag to MODIFIED. The modified object will be saved to the database when you call the appropriate save method.

- 3: DELETED

  If you want to delete an object, set the flag to DELETED, and then call the appropriate save method.

There are some situations where the object flagged is different from the object the save method is performed on. This occurs when

- there is a parent-child relationship between objects, for example, when you update a DTestResult (child), but you must the save the DTestInstance that it belongs to. There is no save method for the DTestResult.
- when you save a set of objects in bulk. Some methods have signatures that take an array of objects, such as saveDCSInstances(...). If you use this method, you must flag each DDCSInstance in the DDCSInstances array separately.

The following table shows which objects you can set the IModificationFlag to and then which object to save:

Set IModificationFlags on:	**Save the:**
DATRow	DATRow array
DDCSInstance	DDCSInstance array
DTestInstance	DTestInstance array
DTestResult	DTestInstance
DDefectRepairEntry	DTestInstance
DAbstractChecklistItem	DAbstractChecklist
DConsumedPart	DConsumptionSet
DRuntimeBomItem	DConsumptionSet
DMessagePack	DLocalePack
DMessageID	DMessagePack

The save methods described in the Integrate Web Services API online help provide more information on this flag. For example, if you look at the saveDCSInstances method, it provides details on what will happen to the objects you pass to it based on their modification flags.

DTrxInfo Class

Some methods take the trxInfo argument. This argument refers to the DTrxInfo class. Use this flag to specify what users should be recorded as responsible for a transaction when you are using a crew, approving user, or electronic signature. If you only want the current user to be recorded as responsible for a transaction, then you can set trxInfo to null. For more information on the DTrxInfo class, refer to the Integrate Web Services API online help.

Sample Code for Each Services Package

The com.datasweep.plantops.common.services package contains interfaces that you will use to build your Web Services application. To use the methods under each interface, you must create an instance of the service to use in your application.

NOTE: In the following code examples, the **`proxyFactory`** variable is an instance of ProxyFactory.

The following is an alphabetized list of all the interfaces. Under each interface name, you will find a brief description of the interface and a code example that shows how to use a method from that interface.

AccessControl

This service contains access control methods. Use this service to change an object's access level or check in, check out, and undo checkout on an object.

```
changeAccessLevelRoute(...)
```

The following example checks in a list object:

```
// Create required proxies for the method calls

ObjectRetrieval objRet =
proxyFactory.getObjectRetrievalProxy();

AccessControl accCont = proxyFactory.getAccessControlProxy();

// Retrieve list object

DList objList = objRet.getListByName("WS_DefectCodes");

// check in list

accCont.checkinList(objList, null);
```

BoxHandling

This service contains box handling methods. Use this service to perform transactions on a box. Methods include:

```
addUnitToBox(...)
saveBox(...)
```

The following example adds a unit to a box:

```
// Create required proxy for the method call
BoxHandling boxHand = proxyFactory.getBoxHandlingProxy();
// Set the box key and unit key to use in this example
long boxKey = 9633003;
long unitKey = 9633002;
// Add unit to Box
boxHand.addUnitToBox(boxKey, unitKey, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

CarrierHandling

This service contains carrier handling methods. Use this service to perform transactions on a carrier. Methods include:

```
addBoxToCarrier(...)
changeLocation(...)
```

The following example adds a unit to a carrier:

```
// Create required proxy for the method call
CarrierHandling carrHand =
proxyFactory.getCarrierHandlingProxy();
// Set the carrier key and unit key to use in this example
long carrierKey = 4050042;
long unitKey = 9231005;
// Add unit to a Carrier
carrHand.addUnitToCarrier(carrierKey, unitKey, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

ClientUtility

This service contains client miscellaneous utility methods required for the client application. Methods include:

```
getArrayDataFromActive(...)
sendEmail(...)
```

The following example sends an email message:

```
// Create required proxy for the method call
ClientUtility cltUtil = proxyFactory.getClientUtilityProxy();

// Set variables for email information
String strFrom = "user1@company.com";
String strTo = "user2@company.com";
String strCC = "";
String strBCC = "";
String strSubj = "subject";
String strMsg = "message";
boolean boolHTML = false;

// Send the email
cltUtil.sendEmail(strFrom, strTo, strCC, strBCC, strSubj,
strMsg, boolHTML);
```

Compatibility

This service contains methods that support backward compatibility transactions. Do not use this service.

EquipmentHandling

This service contains equipment handling methods. Use this service to perform transactions on equipment. Methods include:

```
changeResourceCondition(...)
pause(...)
```

The following example changes the resource condition of a piece of equipment:

```
// Create required proxy for the method call
EquipmentHandling eqHand =
proxyFactory.getEquipmentHandlingProxy();

// Set the equipment key to use in this example
long equipKey = 9593853;

// Change the resource condition
eqHand.changeResourceCondition(equipKey, "Down",
"Shutting Down For Maintenance", "Monthly Maintenance", true,
null, "", IDataTransferTypes.TRANSFER_NONE, null);
```

FlowLotHandling

This service contains flow lot handling methods. Use this service to perform transactions on flow lots. Methods include:

```
decrementQuantityConsumed(...)

saveFlowLot(...)
```

The following example saves a flow lot object:

```
// Create required proxies for the method calls

FlowLotHandling flHand =
proxyFactory.getFlowLotHandlingProxy();

ObjectRetrieval objRet =
proxyFactory.getObjectRetrievalProxy();

// Get flow lot by name

DFlowLot objFlowLot = objRet.getFlowLotByName("FL_1");

// Change the flow lot as required

...

// Save the modified flow lot

flHand.saveFlowLot(objFlowLot, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

LotHandling

This service contains lot handling methods. Use this service to perform transactions on lots. Methods include:

```
mergeLot(...)

saveLot(...)
```

The following example adds a unit to a lot:

```
// Create the required proxy for the method call

LotHandling lHand = proxyFactory.getLotHandlingProxy();

// Set the lot key to use in this example

long lotKey = 3208695;

// Add one unit to the lot

DLotReturnData returnData = lHand.addOneUnit(lotKey, "SN_890",
null, "", IDataTransferTypes.TRANSFER_EXTENDED,
IDataTransferTypes.TRANSFER_NONE, null);

// Retrieve the lot object from DLotReturnData

DLot retObjLot = returnData.getDLot();
```

ObjectRetrieval

This service contains object retrieval methods. Use this service to retrieve objects or object counts from the database. Methods include:

```
getProductionLineByKey(...)

getConsumedParts(...)
```

The following example retrieves a list object and then prints the list items to the console:

```
// Create the required proxy for the method call

ObjectRetrieval objRet =
proxyFactory.getObjectRetrievalProxy();

// Retrieve the list object by name

DList objList = objRet.getListByName("BarCodeToPart");

// Loop through the list items and display in console

DListItem[] listItems = objList.getListItems();

for (int p = 0; p < listItems.length; p++) {

    DListItem listItem = listItems[p];

    java.lang.System.out.println("List Item: " +

    listItem.getItemValue());

}
```

ObjectRevisionRetrieval

This service contains object revision retrieval methods. Use this service to retrieve object revisions, object counts, object revision counts, and object summaries. When you retrieve object summaries, you retrieve the keys and names of all objects that either exist in the current tables or were deleted and now exist in the audit tables. Methods include:

```
getObjectRevisionCount(...)

getPartRevisions(...)
```

The following example retrieves all revisions that exist for a part:

```
// Create the required proxy for the method call

ObjectRevisionRetrieval objRevRet = proxyFactory.getObjectRe-
visionRetrievalProxy();

// longKey is the part object key

// intMax is the maximum number of revisions to retrieve

long longKey = 4047096;

int intMax = 100;

// Retrieve parts
```

```
DPart[] arrParts = objRevRet.getPartRevisions(longKey, IOb-
jectSource.ACTIVE, intMax);
```

ObjectStorage

This service contains object storage methods. The box, lot, flow lot, unit, and work order object handling services have their own save methods. The crew, group, and user objects are saved using the Security service. You must use the object storage service to save all other objects. For example:

```
saveTestInstance(...)
```

Box and work order handling services have their own remove methods. You must use the object storage service to remove all other objects that can be removed from the database. For example:

```
removeTestDefinition(...)
```

The object storage service also contains the following methods:

```
changeUserSequenceTag(...)

createOrGetTemplateBomByBomKey(...)

createOrGetTemplateBomByBomName(...)

createOrGetTemplateBomByPartNumber(...)

resetUserSequence(...)
```

The following example creates and saves a new route operation:

```
// Create the required proxy for the method call

ObjectStorage objStor = proxyFactory.getObjectStorageProxy();

// Create the new route operation and set its properties

DRouteOperation newOp = new DRouteOperation();

newOp.setName("Rework_Op");

newOp.setModificationFlag(IModificationFlags.NEW);

// Save the new route operation

long newKey = objStor.saveRouteOperation(newOp, null, "",
null);
```

ProductionLineHandling

This service contains production line handling methods. Use this service to perform transactions on production lines. Methods include:

```
changeResourceCondition(...)

restart(...)
```

The following example pauses a production line:

```
// Create required proxy for the method call
```

```
ProductionLineHandling plHandling =
proxyFactory.getProductionLineHandlingProxy();

// Set the production line key to use in this example

long plKey = 8475483;

// Pause the production line

plHandling.pause(plKey, "", null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Security

This service contains security methods. Use this service to authenticate, retrieve, and save users; login and logout; remove or save groups; save crews; and, retrieve the security realm. Methods include:

```
isCallerInRole(...)

saveCrew(...)
```

The following example retrieves the current user and then displays the user's first and last name in the console:

```
// Create required proxy for the method call

Security sec = proxyFactory.getSecurityProxy();

// Retrieve the current user

DUser currentUser = sec.getCurrentUser();

// Display the user’s first and last name in the console

String strFirstName = currentUser.getFirstName();

String strLastName = currentUser.getLastName();

java.lang.System.out.println("Current User: " + strFirstName +
" " + strLastName);
```

System

This service contains the following system methods:

```
changePassword(...)

getDBInfo(...)
```

TIP: To access this service, you do not have to log in to the application server.

The following example retrieves database information and then displays the Active database name and type (e.g. Oracle) in the console:

```
// Create required proxy for the method call

com.datasweep.plantops.common.services.System syst =
proxyFactory.getSystemProxy();
```

```
// Retrieve database information
DDBInfo dbInfo = syst.getDBInfo();

// Display the Active database name and type
String strActiveName = dbInfo.getActiveDBDatabaseName();
String strActiveType =
dbInfo.getActiveDBDatabaseProductName();
java.lang.System.out.println("Active DB Name: " +
strActiveName + ", Active DB Type: " + strActiveType);
```

UnitHandling

This service contains unit handling methods. Use this service to perform transactions on units. Methods include:

```
applyECOFromBom(...)
save(...)
```

The following example changes a unit's on hand status to bad:

```
// Create the required proxy for the method call
UnitHandling uHand = proxyFactory.getUnitHandlingProxy();
// Set the unit key to use in this example
long unitKey = 3287564;
// Change the on hand status to bad
uHand.changeOnHandStatus(unitKey,
ITrackedObjectConstants.ON_HAND_STATUS_BAD, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

WorkCenterHandling

This service contains work center handling methods. Use this service to perform transactions on work centers. Methods include:

```
addTools(...)
pause(...)
```

The following example adds three pieces of equipment (tools) to a work center:

```
// Create the required proxy for the method call
WorkCenterHandling wcHand =
proxyFactory.getWorkCenterHandlingProxy();
// Set the work center key to use in this example
long longWCKey = 4053047;
```

```
// Set the equipment keys to use in this example
long[] longEquipKeys = new long[3];
longEquipKeys[0] = 4053041;
longEquipKeys[1] = 4053043;
longEquipKeys[2] = 4053045;
// Add the equipment (tools) to the work center
wcHand.addTools(longWCKey, longEquipKeys, true, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

WorkFlowHandling

This service contains work flow handling methods. Use this service to perform transactions on work flows. Methods include:

```
changeResourceCondition(...)
removeObjects(...)
```

The following example closes a work flow:

```
// Create the required proxies for the method calls
WorkFlowHandling wfHand =
proxyFactory.getWorkFlowHandlingProxy();
// Set the work flow key to use in this example
long wfKey = 6743545;
// Close the work flow
wfHand.close(wfKey, "", null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

WorkOrderHandling

This service contains work order handling methods. Use this service to perform transactions on work orders. Methods include:

```
addWorkOrderItem(...)
saveWorkOrder(...)
```

The following example ships a work order:

```
// Create the required proxy for the method call
WorkOrderHandling woHand =
proxyFactory.getWorkOrderHandlingProxy();
// Set the work order key to use in this example
long woKey = 9396239;
// Ship the work order
woHand.ship(woKey, "", false, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Chapter 3

Integrate Web Services Code Examples

In this chapter

- ❑ **Working with DateTime Values 43**
- ❑ **Retrieving Objects from the Database 45**

  Retrieving Objects by Object Name 45
  Retrieving Objects by Key 46
  Retrieving Objects Using a Filter 46
  Filtering By Date Values 47
  Retrieving Objects Using SQL 48
  Retrieving Object Attributes 49
- ❑ **Changing Object Attributes 51**

  Automatically Committed Changes 51
  Manually Committed Changes 51
- ❑ **Performing and Updating Data Collection 53**
- ❑ **Starting a Unit at a Route Step 58**
- ❑ **Completing a Unit at a Route Step 58**
- ❑ **Collecting Test Data 59**
- ❑ **Creating a Work Order 61**
- ❑ **Creating an Account 62**
- ❑ **Linking an Account to a Work Order 63**
- ❑ **Tracking Equipment on a Resource Route 64**
- ❑ **Creating a Part 65**
- ❑ **Creating a BOM 65**
- ❑ **Modifying the BOMItem Quantity 66**
- ❑ **Linking a BOM to a Part 67**
- ❑ **Performing Consumption 67**

❑ **Managing Application Tables 69**

This chapter provides code examples that you can use to build your application. The code examples are written in Java. The steps and syntax may be different for your environment.

Working with DateTime Values

Anytime you are storing or filtering a DateTime value (for example, using a DDCSDefinition to collect DateTime values, using a date as a search constraint for a filter, etc), the Date value must be formatted as a String and follow the ISO 8601 format for dates and times when you save to the database. The ISO 8601 format is:

CCYY-MM-DDThh:mm:ss.SSS

Table 3-1 defines each value in the ISO 8601 representation.

Table 3-1 ISO 8601 Definitions

Variable	Definition
CC	Thousands and hundreds of years (century).
YY	Tens and individual years.
MM	Month
DD	Day, preceded by an optional leading sign to indicate a negative number. If the sign is omitted, then + is the default value.
T	Date and time separator
hh	Hour
mm	Minute
ss	Second
SSS	Millisecond

The following rules apply:

- Do not use a.m. or p.m. You must use 24-hour clock notation.
- Leading zeroes are required.
- If necessary, you can specify the DateTime value in relation to Coordinated Universal Time (UTC).

TIP: UTC is the local time at the prime meridian (zero longitude) and was previously known as Greenwich Mean Time. It is the international standard of time.

If you specify the DateTime value in relation to UTC, then the data is converted to local database time before it is saved. There are two ways you can specify the DateTime value in relation to UTC.

- If you want to specify that the value is expressed in UTC, then add a “Z” to the end. When the data is saved to the database, it will be converted to the local database time in relation to UTC.
- If you want to specify that the value is expressed in a different local time zone and that time zone is *x* number of hours ahead or behind UTC, then add a “+” or “-” sign immediately after the time, followed by hh:mm to indicate the time difference (offset) from UTC. When the data is saved to the database, it will be converted to the local database time in relation to the different local time zone with the offset.

Your DateTime value must be formatted like one of the following three examples:

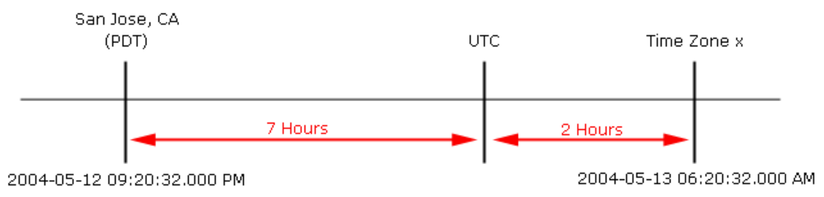
- 2004-05-13T06:20:32.000

  The above value is saved to the database as expressed. No relation to UTC was specified.
- 2004-05-13T06:20:32.000Z

  The above says that the DateTime value is expressed in UTC. If the database is in San Jose, CA and California is currently in Daylight Saving Time, then this corresponds to 2004-05-12 11:20:32.000 PM United States Pacific Daylight Time, which is seven hours behind UTC. The value is saved to the database as 2004-05-12 11:20:32.000 PM.
- 2004-05-13T06:20:32.000+02:00

  The above says that the DateTime value is expressed in a local time in a time zone that is two hours ahead of UTC. If the database is in San Jose, CA and California is currently in Daylight Saving Time, then this corresponds to 2004-05-12 09:20:32.000 PM United States Pacific Daylight Time. The nine hour difference includes the seven hours that PDT is behind UTC plus the two hours that the second time zone is ahead of UTC (Fig. 3-1). The value is saved to the database as 2004-05-12 09:20:32.000 PM.

Figure 3-1: Relative DateTime Value



The DataConverter class is packaged as part of PlantOpsClientSDK.zip (see “Choosing Manually-generated or Pre-generated Proxies” on page 14) and includes methods that convert values to the ISO 8601 format. The steps for DateTime data collection are the same as those outlined in “Performing and Updating Data Collection” on page 53, except as follows:

- include an additional import statement so you can access the DataConverter class.

  ```
  import com.datasweep.plantops.common.utility.*;
  ```

- convert the DateTime value when you use the setValue method

  In the above example, change the setValue line to the following:

  ```
  dcInstItems[x].setValue(DataConverter.fromDate(new
  GregorianCalendar()));
  ```

Retrieving Objects from the Database

You can retrieve objects from the database by:

- Specifying the object name
- Specifying the object key
- Using the DFilter
- Executing SQL statements

You can also retrieve object attributes from the database without retrieving the object.

Retrieving Objects by Object Name

If an object's name uniquely identifies it, then you will be able to retrieve the object using only its name. For example, a part's name is not distinct enough for it to be used to retrieve a unique part from the database. Multiple parts can have the same name but different revisions, and the part name and revision together uniquely identify the part.

Part, unit, and BOM objects are the only objects that cannot be retrieved by name. For all other objects, you will find a get<*objectName*>ByName(...) method in the Integrate Web Services API.

```
ObjectRetrieval objectRetrievalService = proxyFactory.getOb-
jectRetrievalProxy();

DBox objBox = objectRetrievalService.getBoxByName("lnNormal-
Box", IDataTransferTypes.TRANSFER_NORMAL);
```

In Web Services, there is no method equivalent to the Plant Operations getUnitBySerialNumber(...) method. To retrieve a unit by serial number, you must use a filter as shown in “Retrieving Objects Using a Filter” on page 46. In the examples in this document, when a unit is retrieved from the database, it is retrieved by a key.

Retrieving Objects by Key

You can use an object’s key to retrieve the object from the database.

```
ObjectRetrieval objectRetrievalService = proxyFactory.getOb-
jectRetrievalProxy();

DBox objBox = objectRetrievalService.getBoxByKey(9261002,
IDataTransferTypes.TRANSFER_EXTENDED);
```

Retrieving Objects Using a Filter

If an object’s name does not uniquely identify it, then you must use a filter to retrieve the object from the database. You must use a filter to retrieve a single part, unit, or BOM from the database. For example, multiple parts can have the same name but different revisions, and the part name and revision together uniquely identify the part. You can also use filters to retrieve multiple objects from the database that match the filter criteria.

After you have created the filter object, you must set its object type (e.g. unit) and object source (e.g. Production database). Next, you create and set your search constraints. Finally, you assign the search constraints to the filter, and then pass the filter to the appropriate method.

When using the DFilter.setSearchConstraints(array dfiltersearchconstraint) and you have more than one dfiltersearchconstraint in the array, the middleware will join the constraints using the AND keyword. If you want the constraints combined using an OR, you can use the setOrFilters(DFilter[]) method on two filters each containing one search criteria.

```
ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();


// Create the filter, set the object type and object source

DFilter filt = new DFilter();

filt.setObjectType(IObjectTypes.TYPE_UNIT);

filt.setObjectSource(IObjectSource.ACTIVE);


// Create the search constraint

DFilterSearchConstraint cons = new DFilterSearchConstraint();


// Filter by serial number

cons.setAttribute(IUnitFilterAttributes.SERIAL_NUMBER);

cons.setComparisonOperator(IFilterComparisonOpera-
tors.EQUAL_TO);

String[] strSNs = new String[1];
```

```
strSNs[0] = "DS290";
cons.setValues(strSNs);

// constraints must be in an array
DFilterSearchConstraint[] arrCons = new DFilterSearchCon-
straint[1];
arrCons[0] = cons;

// assign the constraints to the filter
filt.setSearchConstraints(arrCons);
// use the filter with the getUnits method
DUnit[] objs = objRet.getUnits(filt, IDataTransfer-
Types.TRANSFER_EXTENDED);
DUnit singleObj = objs[0];
```

Filtering By Date Values

If you want to create a filter with a search constraint for date value, you must format the Date value as a String and follow the ISO 8601 format for dates and times when you save to the database. See “Working with DateTime Values” on page 43 for more information about this.

The following example shows how to create filter that retrieves DCInstances that were created before a certain date.

```
//create the DFilter
// Set the ObjectIdentifier to a specific DCS object
// set the object type to be retrieved
// identify source database
DFilter atFilt = new DFilter();
atFilt.setObjectIdentifier("DCS_Definition");
atFilt.setObjectType(IObjectTypes.TYPE_DCSINSTANCE);
atFilt.setObjectSource(IObjectSource.ACTIVE);

// create the search constraint object
DFilterSearchConstraint atCons = new DFilterSearchCon-
straint();

//filter by “creation time less than”
atCons.setAttribute(IDCSInstanceFilterAttributes.CREATION-
TIME);
```

```
atCons.setComparisonOperator(IFilterComparisonOpera-
tors.LESS_THAN);

// the comparison values must be in an array
// IMPORTANT: note here that strDate is a string formatted
// using the ISO 8601 format
String strDate = "2005-02-01T06:20:32.000";
String[] strNames = new String[1];
strNames[0] = strDate;
atCons.setValues(strNames);

// all search constraints must be in an array
DFilterSearchConstraint[] arrATCons = new DFilterSearchCon-
straint[1];
arrATCons[0] = atCons;

// assign the constraints to the filter and pass in the
// comparison values
atFilt.setSearchConstraints(arrATCons);

//get the getDCSInstance objects that meet the criteria
// uisng the filter
ObjectRetrieval objRetrieve = proxyFactory.getObjectRetrieval-
Proxy();
DDCSInstance[] arrDCInst = objRetrieve.getDCSInstances(at-
Filt);
```

Retrieving Objects Using SQL

You can use methods that take a SQL statement as an argument to retrieve one or more objects from the database.We do not recommend this approach because it creates a dependency on the current table schema and suggest you use other retrieval methods whenever possible. For example, you can retrieve object attributes using the getObjectAttribute(...) method. See “Retrieving Object Attributes” on page 49.

```
ClientUtility cu = proxyFactory.getClientUtilityProxy();

// Define the SQL statement, pass it to getArrayDataFromActive
String strSql = "SELECT BOX_NAME, BOX_TYPE FROM BOX WHERE
BOX_NAME LIKE 'ln%'";
```

```
DDataSet dataSet = cu.getArrayDataFromActive(strSql);

// Retrieve the data from dataSet
DRow[] dataRows = dataSet.getDataRows();
int x = 0;
int y = 0;
for (x=0; x<dataRows.length; x++) {
    DRow dataRow = dataRows[x];
    // Retrieve the box name and type values from dataRow
    String[] strValues = dataRow.getValues();
    for (y=0; y<strValues.length; y++) {
     String strValue = strValues[y];
     java.lang.System.out.println(strValue);
    }
}
```

When you run this example, the println output window displays the box name and box type for every box in the database, for example:

```
lnGroupBox185
1
.
.
.
lnGroupBox130
1
lnNormalBox213
0
```

Retrieving Object Attributes

You can fetch object attributes from the database without retrieving the object itself. If you do not need the actual object but only need a few attributes, it is faster to use the getObjectAttributes(...) method than to retrieve the object and then retrieve its attributes. For example, to start a unit at a route step, you need its key but you do not need the unit object. Use the getObjectAttributes(...) method to retrieve the key as follows:

```
ObjectRetrieval or = proxyFactory.getObjectRetrievalProxy();

// Create the filter
DFilter filt = new DFilter();
filt.setObjectType(IObjectTypes.TYPE_UNIT);
filt.setObjectSource(IObjectSource.ACTIVE);

// Create the search constraint
DFilterSearchConstraint cons = new DFilterSearchConstraint();

// Filter by serial number
cons.setAttribute(IUnitFilterAttributes.SERIAL_NUMBER);
cons.setComparisonOperator(IFilterComparisonOpera-
tors.EQUAL_TO);
String[] strSNs = new String[1];
strSNs[0] = "DS290";
cons.setValues(strSNs);

// Constraints must be in an array
DFilterSearchConstraint[] arrCons = new DFilterSearchCon-
straint[1];
arrCons[0] = cons;

// Assign constraints to the filter
filt.setSearchConstraints(arrCons);

// Create the attribute array
DAttribute[] arrAttr = new DAttribute[1];

// Define what attribute you want back
DAttribute attr = new DAttribute();
attr.setAttribute(IUnitFilterAttributes.KEY);

// Assign desired attributes to array
arrAttr[0] = attr;

// Call the getObjectAttributes method
DDataSet dataSet = or.getObjectAttributes(arrAttr, filt);

// Retrieve data from dataSet
DRow[] dataRows = dataSet.getDataRows();
DRow dataRow = dataRows[0];
```

```
String[] strValues = dataRow.getValues();
long longKey = Long.parseLong(strValues[0]);
```

Changing Object Attributes

The following examples show how to retrieve a unit and change its attributes. The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

In Plant Operations, an automatically committed change is a transaction and a response object is returned, while a manually committed change sets a property and requires a save. Similarly, in Web Services, an automatically committed change is a transaction, but no response object is returned. A manually committed change sets a property and requires a save.

Automatically Committed Changes

In this example, a unit is retrieved and its priority is changed. The *changePriority(...)* method saves the new priority to the database without the need to perform a separate save transaction on the unit (similar to the corresponding method in Plant Operations).

```
// Create required proxies for the method calls
ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();
UnitHandling uh = proxyFactory.getUnitHandlingProxy();


// Retrieve the unit key using getObjectAttributes(...) as shown
// in "Retrieving Object Attributes" on page 49
// For simplicity, unit key is hard-coded
long unitKey = 9324002;


// Change the priority
// IDataTransferType NONE because I don't need the unit
// object returned
uh.changePriority(unitKey, 5, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Manually Committed Changes

In this example, a unit is retrieved and its named UDA is changed. Unlike the example above, the unit must then be saved for these changes to take effect (similar to the corresponding method in the Plant Operations API).

```
// Create required proxies for the method calls
ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();
UnitHandling uh = proxyFactory.getUnitHandlingProxy();

// Retrieve the unit by key
// IDataTransferType EXTENDED because I want to change the unit
long unitKey = 9540005;
DUnit objUnit = objRet.getUnitByKey(unitKey, IDataTransfer
Types.TRANSFER_EXTENDED);

// Create and populate the UDA instance item
// IModificationFlags NEW to flag as new UDA instance item
DUDAInstanceItem udaInstItem = new DUDAInstanceItem();
udaInstItem.setName("color");
udaInstItem.setValue("Green");
udaInstItem.setModificationFlag(IModificationFlags.NEW);

// UDA instance item must be in an array
DUDAInstanceItem[] udaInstItems = new DUDAInstanceItem[1];
udaInstItems[0] = udaInstItem;

// Create and populate the UDA instance
// IModificationFlags NEW to flag as new UDA instance
DUDAInstance udaInst = new DUDAInstance();
udaInst.setModificationFlag(IModificationFlags.NEW);
udaInst.setDataItems(udaInstItems);

// Assign the UDA instance to the object and save
objUnit.setUserDefinedAttributes(udaInst);
uh.save(objUnit, null, "", IDataTransferTypes.NONE, null);
```

To set uda0 - uda9 or udt0 - udt2, use the above code but change the name assigned to the UDA Instance Item. The uda and udt names must be prefixed by a hash character (#). For example,

- to set uda0, use the following for setName():

```
udaInstItem.setName("#uda0");
```

- to set udt1, use the following:

```
udaInstItem.setName("#udt1");
```

For named UDAs that are DateTime type and for udt0 - udt2 values, you must follow the ISO 8601 format for dates and times when you save to the database. For more information, see "Working with DateTime Values" on page 43.

Performing and Updating Data Collection

The following example shows how to collect data for a unit using a data collection set (DCS). The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

The mechanics of creating and saving new DC Instances using Integrate Web Services differ from using script in Process Designer. You need four DCS objects:

- DDCSDefinition
- DDCSDefinitionItem
- DDCSInstance
- DDCSInstanceItem

DDCSDefinitionItem objects contain information about each item in the DDCSDefinition (the template). DDCSInstanceItem objects contain the name and value information. DDCSInstanceItems are assigned to DDCSInstance objects (the runtime instance) and then saved to the database.

1. Retrieve the DDCSDefinition and DDCSDefinitionItem objects from the database. The following retrieves this information for the DCS named DCS1:

   ```
   DDCSDefinition dcsDef = objRet.getDCSDefinitionByName("DCS1");
   ```

   ```
   DDCSDefinitionItem[] arrDefItems = dcsDef.getDefinition-
   Items();
   ```

2. The DDCSInstanceItems objects must be in a DDCSInstanceItems array. When you declare the array variable, specify how many DDCSInstanceItems you want to create by declaring the size of the array to be the size of the DDCSDefinitionItem array. The following does this for the DCS retrieved in step 1:

   ```
   DDCSInstanceItem[] dcInstItems = new DDCSInstanceItem[arrDe-
   fItems.length];
   ```

3. Loop through the DDCSDefinitionItem array. Each time you loop through, create a new DDCSInstanceItem based on the DDCSDefinitionItem. Use DCSDefinitionItem.getName() to find the items you wish to set. This is shown in the detailed code example following these steps.

4. To begin collecting data for a particular DCS, the DCS key must be assigned to the DDCSInstance object. The following shows how to assign the DCS

object key (DCS already retrieved from the database) to the DDCSInstance object:

```
dcInst.setDcsDefinitionKey(dcsDef.getKey());
```

5. The DDCSInstanceItems objects must be assigned to the DDCSInstance object. The following shows how to assign an array of DDCSInstanceItems to a DDCSInstance object:

```
dcInst.setDataItems(dcInstItems)
```

The following example sets values for two DDCSInstanceItem objects in one DCInstance.

```
// Create required proxies for the method calls
ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();
ObjectStorage objStor = proxyFactory.getObjectStorageProxy();

// Retrieve the unit key using getObjectAttributes(...) as shown
// in “Retrieving Object Attributes” on page 49
// For simplicity, unit key is hard-coded
long unitKey = 9540005;

// Get the DCS
DDCSDefinition dcsDef = objRet.getDCSDefinitionByName("UnitIn-
fo");

// Get DCS Definition items from dcsDef
DDCSDefinitionItem[] arrDefItems = dcsDef.getDefinition-
Items();

// Create new DDCSInstance
DDCSInstance dcInst = new DDCSInstance();

// Create new dcInstanceItems set to the length of arrDefItems
DDCSInstanceItem[] dcInstItems = new DDCSInstanceItem[arrDe-
fItems.length];

// Loop through arrDefItems and set itemOne and itemTwo
for (int x = 0; x < arrDefItems.length; x++) {
    DDCSDefinitionItem defItem = arrDefItems[x];
    dcInstItems[x] = new DDCSInstanceItem();
    dcInstItems[x].setName(defItem.getName());
    dcInstItems[x].setModificationFlag(
    IModificationFlags.NEW);
    dcInstItems[x].setType(defItem.getType());

    // Set values for dcInstanceItem
    if (defItem.getName().equalsIgnoreCase("itemOne")){
     dcInstItems[x].setValue("Blue");
    } else if (defItem.getName().equalsIgnoreCase("itemTwo"))
{
     dcInstItems[x].setValue("XHG-09765");
    }
}

// assign unit key and dcInstanceItems array to the dcInstance
dcInst.setObjectKey(unitKey);
dcInst.setObjectType(IClassTypes.UNIT_CLASS);
dcInst.setDataItems(dcInstItems);
dcInst.setDcsDefinitionKey(dcsDef.getKey());
dcInst.setModificationFlag(IModificationFlags.NEW);

// save the dcInstance
// an array of keys for the dc instances is returned
long[] arrKeys = objStor.saveDCSInstances(new
    DDCSInstance[]{dcInst}, "", null, "", null);
```

StandAlone Data Collection

The steps for standalone data collection are the same as above, except:

- do not assign an object key.

  In the above example, omit the following line:
  `dcInst.setObjectKey(objUnit.getKey());`
- set the object type to standalone.

  In the above example, change the object type line to the following:
  `dcInst.setObjectType(“StandAlone”);`

Modifying Data Collection

The following example shows how to:

- Get a DDCSInstance (a DCInstance in Plant Operations)
- Look through the data itmes
- Modify values in that DDCSInstance
- save the DDCSInstance

```
// create ObjectRetrieval object
// and assign to variable objRetreive
// create ObjectStorage object and assign to variable objStor
// create a DFilter object and assign to variable arrDCInst
...
// get the array of DDCSInstance objects
DDCSInstance[] arrDCInst = objRetrieve.getDCSInstances(at-
Filt);

// loop through the array of DDCSInstance objects and
// get the DDCSInstanceItem array for each
for (int i = 0 ; i < arrDCInst.length; i++)
{
    DDCSInstance objDCInst = arrDCInst[i];
    DDCSInstanceItem[] arrDIs = objDCInst.getDataItems();

// loop through the DDCSInstanceItem array
//
for (int j = 0 ; j < arrDIs.length; j++)
{
DDCSInstanceItem objDI = arrDIs[j];
// if the item is called ‘Item_1’ and the value is null
// change the value
if (objDI.getName().equals("Item_1") && (objDI.getValue()==
null) )
    {
    objDI.setValue("Added Test");
    // **IMPORTANT**
    // if the value is change, set the ModificationFlags
    //for the DCInstance to MODIFIED.
    arrDCInst[i].setModificationFlag(IModificationFlags.MODI-
FIED);
    } //end if
}//end for: arrDIs
}//end for: arrDCInst
//using ObjectStorage object save the array of DDCSInstance
// objects. Those that were marked MODIFIED are saved.
long[] arrkeys = objStor.saveDCSInstances(arrDCInst, "update",
null, "", null);
```

Starting a Unit at a Route Step

The following example shows how to start a unit at a route step. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

To start the unit, retrieve the unit object’s key from the database. Next, perform the start transaction.

```
// Create required proxy for the method call
UnitHandling uh = proxyFactory.getUnitHandlingProxy();

// Retrieve the unit key using getObjectAttributes(...) as shown
// in “Retrieving Object Attributes” on page 49
// For simplicity, unit key is hard-coded
long unitKey = 9324002;

// Start at Route Step 010
uh.startAtRouteStep(unitKey, "010", true, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Completing a Unit at a Route Step

The following example shows how to complete a unit at a route step. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

To complete the unit, retrieve the unit object’s key from the database. Next, perform the complete transaction.

```
// Create required proxy for the method call
UnitHandling uh = proxyFactory.getUnitHandlingProxy();

// Retrieve the unit key using getObjectAttributes(...) as shown
// in “Retrieving Object Attributes” on page 49
// For simplicity, unit key is hard-coded
long unitKey = 9324002;

// Complete at Route Step 010
uh.completeAtRouteStep(unitKey, "010", "Pass", true, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Collecting Test Data

The following example shows how to create a test instance and collect defect and repair information against that test instance. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

In this example we are collecting test data against a unit. You can also collect test data against a route step, lot, or equipment. The mechanics of creating and saving test data using Integrate Web Services differ from using script in Process Designer.

1. The DTestResult objects must be in an array. When you declare the array variable, you must specify how many test results you want to create by declaring the size of the array. Once populated with test result data, this array is assigned to the test instance. The following shows how to create an array that will hold one test result:

   ```
   DTestResult[] arrTestRes = new DTestResult[1];
   ```

2. The DDefectRepairEntry objects must be in an array. When you declare the array variable, you must specify how many defect repair entries you want to create (per test instance) by declaring the size of the array. Once populated with defect repair entry data, this array is assigned to the test instance. The following shows how to create an array that will hold three defect repair entries:

   ```
   DDefectRepairEntry[] arrDre = new DDefectRepairEntry[3];
   ```

The following example collects one test result and one defect repair entry for a unit against a test instance. In this example, the defect is repaired at the same time it is recorded.

```
// Create required proxies for the method calls

ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();

ObjectStorage objStor = proxyFactory.getObjectStorageProxy();

UnitHandling uh = proxyFactory.getUnitHandlingProxy();


// Retrieve the Unit

long unitKey = 9384002;

DUnit objUnit = objRet.getUnitByKey(unitKey, IDataTransfer-
Types.TRANSFER_NORMAL);


// Get the Test Definition

DTestDefinition testDef = objRet.getTestDefinition-
```

```
ByName("TD_1");

// Create the Test Instance and set its properties
// set IModificationFlags to NEW
DTestInstance testInst = new DTestInstance();
testInst.setTestDefinitionKey(testDef.getKey());
testInst.setTestPassed(true);
testInst.setTestValid(true);
testInst.setModificationFlag(IModificationFlags.NEW);
testInst.setObjectKey(objUnit.getKey());
testInst.setObjectName(objUnit.getSerialNumber());
testInst.setObjectType(IClassTypes.UNIT_CLASS);

// Create the Test Result and set its properties
// set IModificationFlags to NEW
DTestResult testRes = new DTestResult();
testRes.setTestResultCode("Temp out of range 2");
testRes.setModificationFlag(IModificationFlags.NEW);

// Test Result needs to be in a Test Result array
DTestResult[] arrTestRes = new DTestResult[1];
arrTestRes[0] = testRes;

// Create the Defect Repair Entry and set its properties
// set IModificationFlags to NEW
DDefectRepairEntry dre = new DDefectRepairEntry();
dre.setDefectCode("48779");
dre.setDefectUserName("ppark");
dre.setRepairCode("Rep90210");
dre.setRepaired(true);
dre.setModificationFlag(IModificationFlags.NEW);

// Defect Repair Entry needs to be in a Defect Repair Entry array
DDefectRepairEntry[] arrDre = new DDefectRepairEntry[1];
arrDre[0] = dre;
```

```
// Assign the Test Results and Defect Repair Entry to the
// Test Instance
testInst.setTestResults(arrTestRes);
testInst.setDefectRepairEntries(arrDre);

// Save the Test Instance
long key = objStor.saveTestInstance(testInst, null, "", null);
```

Creating a Work Order

The following example shows how to create a work order, work order item, lot, and units. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

If you create and save the new Work Order Item at the same time you save the new Work Order, then you can use the DWorkOrder.setOrderItems(...) method. If you want to add the Work Order Item after you have already saved the Work Order, then you must use the WorkOrderHandling.addWorkOrderItem(...) method. You cannot use setOrderItems(...) once the Work Order has been saved.

```
// Create required proxies for the method calls
WorkOrderHandling woh = proxyFactory.getWorkOrderHandling-
Proxy();
LotHandling lh = proxyFactory.getLotHandlingProxy();

// Create Work Order
DWorkOrder objOrder = new DWorkOrder();
objOrder.setModificationFlag(IModificationFlags.NEW);
objOrder.setName("WO_876966");

// Create Work Order Item
DWorkOrderItem objItem = new DWorkOrderItem();
objItem.setModificationFlag(IModificationFlags.NEW);
objItem.setPartNumber("Part1a");
objItem.setPartRevision("1");
objItem.setName("OrderItem1");

// Assign Work Order Item to Work Order Item array
DWorkOrderItem[] items = new DWorkOrderItem[1];
items[0] = objItem;
```

```
// Create Lot & Units
// Required attributes from Integrate Web Services API
// Units are automatically created based on lot quantity
DLot objLot = new DLot();
objLot.setName("WO_876966_LOT");
objLot.setModificationFlag(IModificationFlags.NEW);
objLot.setOrderNumber(objOrder.getName());
objLot.setOrderItem(objItem.getName());
objLot.setQuantity(new java.math.BigDecimal("5"));
objLot.setType(ILotTypes.LOT_TYPE_NORMAL);

// Assign Work Order Item array to Work Order and
// save the Work Order
objOrder.setOrderItems(items);
DWorkOrder objOrder1 = woh.saveWorkOrder(objOrder, null, "",
IDataTransferTypes.TRANSFER_NONE, null);

// Save the lot
DLotReturnData objLotRetDat = lh.saveLot(objLot, null, "",
IDataTransferTypes.TRANSFER_NORMAL, IDataTransfer-
Types.TRANSFER_NONE, null);
```

Creating an Account

The following example shows how to create an account object. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

In order to successfully create and save an account object, it is not necessary to provide data for all three possible contact addresses. However, you must declare the array variable that contains the contact addresses to size 3. If you do not, then the account creation will fail when you try and save it.

```
// Create required proxy for the method call
ObjectStorage objectStorageService = proxyFactory.getObject-
StorageProxy();

// Create Account
DAccount objAccount = new DAccount();
objAccount.setName("Account DEF");
objAccount.setId("8643457");

// Create Contact Address
DContactAddress objAddress = new DContactAddress();
objAddress.setName("Mr. President");
objAddress.setPhone("408-555-9898");
objAddress.setPostalCode("90210");

// Assign Contact Address to Contact Address array
DContactAddress[] addresses = new DContactAddress[3];
addresses[0] = objAddress;

// Assign Contact Address array to Account and save the Account
objAccount.setContactAddresses(addresses);
objAccount.setModificationFlag(IModificationFlags.NEW);
long key = objectStorageService.saveAccount(objAccount, null,
"", null);
```

Linking an Account to a Work Order

The following example shows how to link an account object to a work order. The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

In this example, the work order and account objects are already created and can be retrieved from the database. If you create the account object at the same time you assign it to the work order, then you must remember to capture the newly created object's key that is returned. The account key is needed to associate with the work order. See "Creating a Work Order" on page 61 and "Creating an Account" on page 62 for more information on creating work orders and accounts.

Once you have the work order and account objects, set one or more of the account key properties of the work order object to the account key. You must save the work order for the changes to take effect.

```
// Create required proxies for the method calls
ObjectRetrieval or = proxyFactory.getObjectRetrievalProxy();
WorkOrderHandling woh = proxyFactory.getWorkOrderHandling-
Proxy();
// Retrieve Work Order and Account objects
// IDataTransferTypes EXTENDED because changing the work order
DWorkOrder objOrder = or.getWorkOrderByName("WO_0203_1630",
IDataTransferTypes.TRANSFER_EXTENDED);
DAccount objAcct = or.getAccountByName("Account_A");

// Assign the Account Key to one of the Work Order Account
// properties
objOrder.setBillAccountKey(objAcct.getKey());

// Save the Work Order
// IDataTransferTypes NONE because do not need the object
woh.saveWorkOrder(objOrder, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Tracking Equipment on a Resource Route

The following example shows how to change the resource condition of a piece of equipment. The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

In order to use the method that changes the resource condition, you must first retrieve the equipment and resource route objects from the database. They are required parameters in the method. Alternatively, you can create the equipment and resource route using Web Services, save them, and then use the newly created objects in the following script.

```
// Create required proxies for the method calls
ObjectRetrieval or = proxyFactory.getObjectRetrievalProxy();
EquipmentHandling eh = proxyFactory.getEquipmentHandling-
Proxy();
ObjectStorage os = proxyFactory.getObjectStorageProxy();

// Retrieve Equipment
DEquipment objEquip = or.getEquipmentByName("200-009");

// Set the Resource Route
objEquip.setResourceRouteName("EquipRoute");
long key = os.saveEquipment(objEquip, null, "", null);

// Change the Resource Condition
eh.changeResourceCondition(objEquip.getKey(), "Calibration",
"", "Calibrating", true, null, "", IDataTransfer-
Types.TRANSFER_NONE, null);
```

Creating a Part

The following example shows how to create a part. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

You must set at least the part number, part revision, and consumption type properties or the part will not save.

```
// Create required proxy for the method call
ObjectStorage os = proxyFactory.getObjectStorageProxy();

// Create Part
DPart objPartNew = new DPart();
objPartNew.setPartNumber("NewPart1");
objPartNew.setPartRevision("1");
objPartNew.setConsumptionType("SerialNumber");
objPartNew.setDescription("Read Head Assy");
objPartNew.setModificationFlag(IModificationFlags.NEW);

// Save the part
long key = os.savePart(objPartNew, null, "", null);
```

Creating a BOM

The following example shows how to create a BOM. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

You must set at least the BOM name, BOM revision, and BOM type properties or the BOM will not save.

```
// Create required proxy for the method call
ObjectStorage os2 = proxyFactory.getObjectStorageProxy();


// Create BOM
DBom objBomNew = new DBom();
objBomNew.setBomName("RH-0816");
objBomNew.setRevision("1");
objBomNew.setType(IBomTypes.STANDARD);
objBomNew.setModificationFlag(IModificationFlags.NEW);
```

```
// Create BOM Item
DBomItem objBomItemNew = new DBomItem();
objBomItemNew.setPartNumber("0316-0908");
objBomItemNew.setPartRevision("1");
objBomItemNew.setModificationFlag(IModificationFlags.NEW);

// Assign BOM Item to a BOM Item array
DBomItem[] bomItems = new DBomItem[1];
bomItems[0] = objBomItemNew;

// Assign BOM Item array to BOM and save the BOM
objBomNew.setBomItems(bomItems);
long key2 = os2.saveBOM(objBomNew, null, "", null);
```

Modifying the BOMItem Quantity

The following example shows how to modify the BOMItem quantity. The assumption is that you have completed the steps in “Creating the Required Components of an Application” on page 26.

When changing a BOMItem property, you must also call the following methods setBomItemsChanged(true) method on the BOM. This method call is required in addition to setting the Modification Flag on the BOM and all the modified BOMItems, explained in “Modification Flag” on page 29.

```
// Create required proxies for the method calls
ObjectRetrieval or = proxyFactory.getObjectRetrievalProxy();
ObjectStorage os = proxyFactory.getObjectStorageProxy();

// Get the BOM by specifying the object key
DBom bom = or.getBOMByKey(3002);

// Get an array containing all BOMItems in the BOM
DBomItem[] bomitems = bom.getBomItems();

// Change the quanitity for the first BOMItem in the array
// Flag the BOMItem to be changed
bomitems[0].setQuantity(new BigDecimal(5555));
```

```
bomitems[0].setModificationFlag(IModificationFlags.MODIFIED);

//set the new BOMItems to the BOM and
//flag the BOM to be changed
bom.setBomItems(bomitems);
bom.setModificationFlag(IModificationFlags.MODIFIED);
bom.setBomItemsChanged(true);
os.saveBOM(bom,null,"",null);
```

Linking a BOM to a Part

The following example shows how to link a BOM object to a part. The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

In this example, the BOM and the part objects are already created and can be retrieved from the database. Linking a BOM to a part is accomplished by setting BOM properties of the part object and then saving the part.

```
// Create required proxies for the method calls
ObjectRetrieval or = proxyFactory.getObjectRetrievalProxy();
ObjectStorage os = proxyFactory.getObjectStorageProxy();

// Retrieve Part by Key
DPart objPart = or.getPartByKey(486029);

// Set BOM Name and BOM Revision properties
objPart.setBomName("RH-0816");
objPart.setBomRevision("1");

// Save the part
os.savePart(objPart, null, "", null);
```

Performing Consumption

The following example shows how to consume a part into a unit. The assumption is that you have completed the steps in "Creating the Required Components of an Application" on page 26.

The mechanics of consumption using Integrate Web Services differ from using script in Plant Operations. To consume one part into a unit in Web Services:

1. Retrieve the unit from the database.
2. Create and set the properties for the consumed part.
3. Retrieve the unit’s runtime BOM and get its runtime BOM items.
4. Loop through the runtime BOM items until you find the item you want and assign the consumed part to that runtime BOM item.
5. Set any other runtime BOM item properties.
6. Create a consumption set, assign BOM items to it, and then save the consumption set.

The following example consumes one part into a unit.

```
// Create required proxies for the method calls
ObjectRetrieval objRet = proxyFactory.getObjectRetrieval-
Proxy();
ObjectStorage objStor = proxyFactory.getObjectStorageProxy();

// Retrieve the Unit
long unitKey = 9615002;
DUnit objUnit = objRet.getUnitByKey(unitKey, IDataTransfer-
Types.TRANSFER_NORMAL);

// Create consumed part and set its values
DConsumedPart cp = new DConsumedPart();
cp.setPartSerial("CP897");
cp.setPartNumber("CP11");
cp.setPartRevision("1");
cp.setModificationFlag(IModificationFlags.NEW);

// Get runtime BOM
long bomKey = objUnit.getTBomKey();
DRuntimeBom tbom = objRet.getRuntimeBomByKey(bomKey);

// Get runtime BOM items
DRuntimeBomItem[] bomItems = tbom.getBomItems();

// Find the BOM item we want
DRuntimeBomItem bomItem = null;
for (int i=0; i<bomItems.length; i++) {
    if (bomItems[i].getPartNumber().equalsIgnoreCase("CP11"))
    {
     bomItem = bomItems[i];
     break;
    }
}

// Assign consumed part to BOM item and set BOM item properties
// Consumed part must be in a consumed part array
DConsumedPart[] arrCp = new DConsumedPart[1];
arrCp[0] = cp;
bomItem.setTrackedObjectKey(objUnit.getKey());
bomItem.setTrackedObjectType(IClassTypes.UNIT_CLASS);
bomItem.setConsumedParts(arrCp);
bomItem.setModificationFlag(IModificationFlags.MODIFIED);

// Create consumption set
DConsumptionSet consSet = new DConsumptionSet();

// Assign BOM items to consumption set
// BOM items must be in a BOM items array
DRuntimeBomItem[] arrBomItems = new DRuntimeBomItem[1];
arrBomItems[0] = bomItem;
consSet.setBomItems(arrBomItems);
consSet.setModificationFlag(IModificationFlags.NEW);

// Save the consumption set
objStor.saveConsumptionSet(consSet, null, "",
IDataTransferTypes.TRANSFER_NONE, null);
```

Managing Application Tables

The following section provides code examples and explanations about managing application table data using the Integrate Web Services pre-generated proxies for Java.

Create a New Row to Store Data

This example shows you how to populate an application table by adding rows and defining the column values. The ATDefinition used in this example, has the following columns:

- column name: Column_unit, data type: object (unit)
- column name: Column_string, data type: string
- column name: Column_unitKey, data type: long

Note in the following code example that the column called 'Column_unit' is defined as an 'object' data type, but the actually stores the unit_key. In the database, the column name is Column_unit_47 ('47' represents the unit class id). The master-detail relationship between the DATRow and unit object is created by storing the object key for the unit in the Column_unit_47 field. In the Process Designer scripting environment, when you pass the unit object as an argument, the middleware automatically retrieves the unit_key and stores it in this field. Using Integrate Web Services, you must do this manually.

When referencing another object, the middleware will not check that the value stored is that of a valid object. Also, although an object key is stored in the database as a numeric datatype, you must convert it to a string before passing it as an argument to the setValue(...) method.

The general steps are:

1. Create your ObjectRetrieval and ObjectStorage proxies.

```
//create the ObjectRetrieval Proxy as variable objRetrieve
...
//create the ObjectStorage Proxy as variable objStore
...
```

2. Get the DUnit object using the DFilter or some other mechanism.

```
//create a Dfilter to get the DUnit called ‘DS1’
//assign the DUnit to the unit variable
...
```

3. Create a new DATRow using a DATDefinition, set the DATRow IModificationFlag to NEW, and then set the DATDefinition key and DATRow name.

```
DATDefinition datDef =
    objRetrieve.getATDefinitionByName("ATDefName");
DATRow row = new DATRow();
row.setModificationFlag(IModificationFlags.NEW);
row.setAtDefinitionKey(datDef.getKey());
row.setName("2");
```

4. Initialize each of the DATCell names and data types for The characteristics of each field is defined in a DATColumnDefinition. Each CATCell represents a field in a database record. A DATColumnDefinition is a single column defined in the ATDefinition in Process Designer.

   While initializing the column names, also store the appropriate values to the fields.

```
//initialize all the column definitions in the DATRow
//this is a required step for all new DATRows
DATColumnDefinition[] atColDefs = datDef.getAtColumnDefini-
tions();
DATCell[] cells = new DATCell[atColDefs.length];
for (int i = 0; i < atColDefs.length; i++)
{
    DATColumnDefinition atColDef = atColDefs[i];
    cells[i] = new DATCell();
    cells[i].setName(atColDef.getName());
    cells[i].setDataType(atColDef.getDataType());
    String cellName = cells[i].getName();
    if (cellName.equals("Column_unit"))
    {
          Long unitKey = new Long(unit.getKey());
          cells[i].setValue(unitKey.toString());
    } else if (cellName.equals("Column_string"))
    {
          cells[i].setValue(unit.getSerialNumber());
    //store the unit key
    else if (cellName.equals("Column_unitKey"))
    {
          Long unitKey = new Long(unit.getKey());
          cells[i].setValue(unitKey.toString());
    }
}
```

5. Set the DATCells to the DATRow.

```
row.setDataCells(cells);
```

6. Save the DATRow.

```
objStore.saveATRows(new DATRow[] {row}, null, null,null, null
);
```

Update an Existing Row

The following code example will update the DATRow that has already been saved. If the value being changed is an object key that references another object, the middleware will not check that the value is that of a valid object.

1. Create your ObjectRetrieval and ObjectStorage proxies.

```
//create the ObjectRetrieval Proxy as variable objRetrieve
...
//create the ObjectStorage Proxy as variable objStore
...
```

2. Get the DATRow using a DFilter or some other mechanism.

```
//create a DFilter to get the DATRow with the name ‘2’
//assign the DATRow to the variable ‘row’
...
```

3. Once you have the DATRow, set the Modification Flag to MODIFIED.

```
row.setModificationFlag(IModificationFlags.MODIFIED);
```

4. Get the DATCells from the DATRow, then loop through them to find the cell of interest. Once the cell is found, change the value, and then set the DATCells back to the DATRow.

```
//get the DATCells for the DATRows
DATCell[] cells = row.getDataCells();
//loop through the DATCells and find the cell of interest, then
//change the value of the field
for (int i = 0; i < cells.length; i++)
{
    String cellName = cells[i].getName();
    if (cellName.equals("Column_string"))
    {
    cells[i].setValue("changed");
    }
}
row.setDataCells(cells);
```

5. Save the DATRow.

```
objStore.saveATRows(new DATRow[] {row}, null, null,null, null);
```

Creating a Master-Detail AT Relationship

This example shows how to create a master/detail relationship between two application table rows. For information about ATDefinitions and their related objects, see the Process Designer Online Help. The following example involves two AT Definitions that were created in Process Designer called:

- Master: this contains any number of columns to store data. No specific columns are required.
- Detail: this ATDefinition must have at least one column defined which is where the object key for the Master row will be stored.

Creating a Master / Detail row relationship requires that you:

1. Create your ObjectRetrieval and ObjectStorage proxies.

```
//create the ObjectRetrieval Proxy as variable objRetrieve
//create the ObjectStorage Proxy as variable objStore
```

2. Create the Master DATRow first, and then save it. The Master DATRow must be saved before creating the Detail DATRow. Make sure you retain the object key of the Master DATRow. In this example, it is variable masterKey.

```
//create a new DATRow from a DATDefinition for Master table
DATDefinition datMasterDef = objRetrieve.getATDefinition-
ByName("Master");
DATRow masterRow = new DATRow();
masterRow.setModificationFlag(IModificationFlags.NEW);
masterRow.setAtDefinitionKey(datMasterDef.getKey());
masterRow.setName("2");

//initialize all cells in the Master ATRow
//this is where you would also store any data to the columns
DATColumnDefinition[] atMasterColDefs = datMasterDef.getAtCol-
umnDefinitions();
DATCell[] masterCells = new DATCell[atMasterColDefs.length];
for (int i = 0; i < atMasterColDefs.length; i++)
    {
    DATColumnDefinition atMasterColDef = atMasterColDefs[i];
    masterCells[i] = new DATCell();
    masterCells[i].setName(atMasterColDef.getName());
    masterCells[i].setDataType(atMasterColDef.getData-
Type());
    String cellName = masterCells[i].getName();
    } //end for
```

```
masterRow.setDataCells(masterCells);

//save the Master ATRow

long[] masterKey =
objStore.saveATRows(new DATRow[] {masterRow}, null, null,null,
null);
```

3. Create the Detail DATRow, store the object key from the Master DATRow in the specified column, and then save the Detail DATRow.

```
//create a new DATRow from a DATDefinition for Detail table

DATDefinition datDetDef = objRetrieve.getATDefinition-
ByName("Detail");

DATRow detRow = new DATRow();

detRow.setModificationFlag(IModificationFlags.NEW);

detRow.setAtDefinitionKey(datDetDef.getKey());

detRow.setName("2");


//initialize all cells in the Detail DATRow

//store the Master DATRow object key

DATColumnDefinition[] atDetColDefs = datDetDef.getAtColumnDef-
initions();

DATCell[] detCells = new DATCell[atDetColDefs.length];

for (int i = 0; i < atDetColDefs.length; i++)

{

    DATColumnDefinition atDetColDef = atDetColDefs[i];

    detCells[i] = new DATCell();

    detCells[i].setName(atDetColDef.getName());

    detCells[i].setDataType(atDetColDef.getDataType());

    String detCellName = detCells[i].getName();

    if (detCellName.equals("Master"))

    {

          int myInteger = (int) masterKey[0];

          Integer intKey = new Integer(myInteger);

          String strKey = intKey.toString();

          detCells[i].setValue(strKey);

    }

} //end for

detRow.setDataCells(detCells);

objStore.saveATRows(new DATRow[] {detRow}, null, null,null,
null);
```

Index

A
AccessControl
  code example 31
  overview 21
Accounts
  creating 62
  linking to Work Orders 63
Apache Axis 12
Application
  creating required components 26
Application Tables
  checking for a valid object 70
  link two ATRows 73
  managing 69
  new row 70
  storing object key 70
  update a row 72
Architecture
  Integrate Web Services 13
as-built. See Consumption
ATRow
  relationship with objects 70
authentication 15

B
BOMs
  creating 65, 66
  linking to Parts 67
BoxHandling
  code example 32
  overview 21

C
CarrierHandling
  code example 32
  overview 21
changing object attributes 51
classpath
  set up 16
ClientUtility
  code example 32
  overview 22
collecting data 53
communication
  directly to EJBs 13
  via web services 13
Compatibility 33
  overview 22
Completing objects at a route step 58
constants
  overview 20
  using numeric 15
  using textual 15
Consumption 67
Coordinated Universal Time 43
Creating
  accounts 62
  BOMs 65, 66
  DC Instances 53
  parts 65
  proxies 16
  work order items 61
  work order lots 61
  work order units 61
  work orders 61

D
DAbstractChecklist 30
DAbstractChecklistItem 30

Data Collection 53, 56
Data Transfer Flag 28
DataConverter
  documentation 8
Dataobjects 20
DATCell 71
DATDefinition 70
DATRow 30, 70
DConsumedPart 30
DConsumptionSet 30
DCS
  assigning DC instances to 53
  creating DC instances 53
  StandAlone 56
DDCSDefinition 53
DDCSDefinitionItem 53
DDCSInstance 30, 53
DDCSInstanceItem 53
DDefectRepairEntry 30, 59
default values for new objects 14
DELETED IModificationFlags 30
Detail Row 73
development environment 15
DFilter
  for DUnit 70
DLocalePack 30
DMessageID 30
DMessagePack 30
Documentation
  DataConverter 8
  ProxyFactory 8
download proxies 16
DTestInstance 30
DTestResult 30, 59
DTrxInfo 31

E
EJB Client Application 13
environment
  development 15
Equipment
  changing Resource Condition of 64
equipment tracking 64
EquipmentHandling
  code example 33
  overview 22
exception handling 28
Exceptions 21

F
Filtering 20
FlowLotHandling
  code example 34
  overview 22

G
geneology. See Consumption
getting object attributes 49
getting objects
  by key 46
  using a DFilter 46
  using sql 48
getting objects by name 45

I
IDataTransferTypes 28
IIOP communication
  JBoss 27
initializing DATCells 71
Integrate Web Services
  architecture 13
Interoperability 12
ISO8601 43

J
J2EE vendor 16
JBoss
  IIOP communication 27
jnp 27

L
Logging in 27
Logging out 28
LotHandling
  code example 34
  overview 22

M
managing application table data 69
manually-generated proxies 14
Master Row 73
Modification Flag overview 29
MODIFIED IModificationFlags 30

N
Named UDAs 51
NEW IModificationFlags 29
new objects

default values 14

O
object handling 15
ObjectRetrieval
code example 35
overview 22
ObjectRevisionRetrieval
code example 35
overview 22
objects
automatically committing changes 51
changing attributes 51
completing at a route step 58
getting attributes 49
getting by name 45
manually committing changes 51
retrieving using a DFilter 46
retrieving using SQL 48
retrieving using the key 46
saving 20
starting at a route step 58
ObjectStorage
code example 36
overview 22

P
package
descriptions 19
Parts
creating 65
linking a BOM to 67
pre-generated proxies 14
ProductionLineHandling
code example 36
overview 22
Proxies
choosing the 14
creating 16
creating manually 16
using pre-generated 16
ProxyFactory
documentation 8

R
Readme.html 9
Related Documents 8
Release Notes 9
Required Components 26
RMI 12, 13

S
saveDCSInstances method 30
saving objects
overview 20
relationship to IModificationFlag 30
Security
code example 37
overview 22
Services Package 21
AccessControl 31
BoxHandling 32
CarrierHandling 32
ClientUtility 32
Compatibility 33
EquipmentHandling 33
FlowLotHandling 34
LotHandling 34
ObjectRetrieval 35
ObjectRevisionRetrieval 35
ProductionLineHandling 36
Security 37
System 37
UnitHandling 38
WorkCenterHandling 38
WorkFlowHandling 39
WorkOrderHandling 39
SOAP 12, 13
stand alone data collection 56
Starting objects at a route step 58
System
code example 37
overview 22

T
Technical Support 9
Test Definitions 59
tracking a unit 58
TRANSFER_EXTENDED 29
TRANSFER_NONE 28
TRANSFER_NORMAL 28
trxInfo 31
try-catch blocks 28

U
UDAs
Named UDAs 51
uda0 - uda9 52
udt0 - udt2 52
UNCHANGED IModificationFlags 29

unit tracking 58
UnitHandling
  code example 38
  overview 22
Units
  completing 58
  starting 58
Upgrading an application 19
user authentication 15
UTC 43
Utility
  overview 22

V
vendor
  J2EE Application Server 16
Version
  overview 23

W
Web Services Client Application 13
Work Orders
  creating 61
  linking an Account to 63
WorkCenterHandling
  code example 38
  overview 22
WorkFlowHandling
  code example 39
  overview 22
WorkOrderHandling
  code example 39
  overview 22
WSDL 12

X
xfrType argument 28