Sterling Commerce
An IBM Company

Selling and Fulfillment Foundation: Integration Guide

Release 8.5

October 2009

Copyright Notice

STERLING COMMERCE SOFTWARE

Sterling Commerce, Inc.
4600 Lakehurst Court
Dublin, Ohio 43016-2000

Third-Party Software

Portions of the Sterling Commerce Software may include products, or may be distributed on the same storage media with products, ("Third Party Software") offered by third parties ("Third Party Licensors"). Sterling Commerce Software may include Third Party Software covered by the following copyrights: Copyright © 2006-2008 Andres Almiray. Copyright © 1999-2005 The Apache Software Foundation. Copyright (c) 2008 Azer Koçulu http://azer.kodfabrik.com. Copyright © Einar Lielmanis, einars@gmail.com. Copyright (c) 2006 John Reilly (www.inconspicuous.org) and Copyright (c) 2002 Douglas Crockford (www.crockford.com). Copyright (c) 2009 John Resig, http://jquery.com/. Copyright © 2006-2008 Json-lib. Copyright © 2001 LOOX Software, Inc. Copyright © 2003-2008 Luck Consulting Pty. Ltd. Copyright 2002-2004 © MetaStuff, Ltd. Copyright © 2009 Michael Mathews micmath@gmail.com. Copyright © 1999-2005 Northwoods Software Corporation. Copyright (C) Microsoft Corp. 1981-1998. Purple Technology, Inc. Copyright (c) 2004-2008 QOS.ch. Copyright © 2005 Sabre Airline Solutions. Copyright © 2004 SoftComplex, Inc. Copyright © 2000-2007 Sun Microsystems, Inc. Copyright © 2001 VisualSoft Technologies Limited. Copyright © 2001 Zero G Software, Inc. All rights reserved by all listed parties.

The Sterling Commerce Software is distributed on the same storage media as certain Third Party Software covered by the following copyrights: Copyright © 1999-2006 The Apache Software Foundation. Copyright (c) 2001-2003 Ant-Contrib project. Copyright © 1998-2007 Bela Ban. Copyright © 2005 Eclipse Foundation. Copyright © 2002-2006 Julian Hyde and others. Copyright © 1997 ICE Engineering, Inc./Timothy Gerard Endres. Copyright 2000, 2006 IBM Corporation and others. Copyright © 1987-2006 ILOG, Inc. Copyright © 2000-2006 Infragistics. Copyright © 2002-2005 JBoss, Inc. Copyright LuMriX.net GmbH, Switzerland. Copyright © 1998-2009 Mozilla.org. Copyright © 2003-2009 Mozdev Group, Inc. Copyright © 1999-2002 JBoss.org. Copyright Raghu K, 2003. Copyright © 2004 David Schweinsberg. Copyright © 2005-2006 Darren L. Spurgeon. Copyright © S.E. Morris (FISH) 2003-04. Copyright © 2006 VisualSoft Technologies. Copyright © 2002-2009 Zipwise Software. All rights reserved by all listed parties.

Certain components of the Sterling Commerce Software are distributed on the same storage media as Third Party Software which are not listed above. Additional information for such Third Party Software components of the Sterling Commerce Software is located at: installdir/mesa/studio/plugins/SCI_Studio_License.txt.

Third Party Software which is included, or are distributed on the same storage media with, the Sterling Commerce Software where use, duplication, or disclosure by the United States government or a government contractor or subcontractor, are provided with RESTRICTED RIGHTS under Title 48 CFR 2.101, 12.212, 52.227-19, 227.7201 through 227.7202-4, DFAR 252.227-7013(c) (1) (ii) and (2), DFAR 252.227-7015(b)(6/95), DFAR 227.7202-3(a), FAR 52.227-14(g)(2)(6/87), and FAR 52.227-19(c)(2) and (6/87) as applicable.

Additional information regarding certain Third Party Software is located at installdir/SCI_License.txt.

Some Third Party Licensors also provide license information and/or source code for their software via their respective links set forth below:

http://danadler.com/jacob/

http://www.dom4j.org

This product includes software developed by the Apache Software Foundation (http://www.apache.org). This product includes software developed by the Ant-Contrib project (http://sourceforge.net/projects/ant-contrib). This product includes software developed by the JDOM Project (http://www.jdom.org/). This product includes code licensed from RSA Data Security (via Sun Microsystems, Inc.). Sun, Sun Microsystems, the Sun Logo, Java, JDK, the Java Coffee Cup logo, JavaBeans , JDBC, JMX and all JMX based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. All other trademarks and logos are trademarks of their respective owners.

THE APACHE SOFTWARE FOUNDATION SOFTWARE

The Sterling Commerce Software is distributed with or on the same storage media as the following software products (or components thereof) and java source code files: Xalan version 2.5.2, Cookie.java, Header.java, HeaderElement.java, HttpException.java, HttpState.java, NameValuePair.java, CronTimeTrigger.java, DefaultTimeScheduler.java, PeriodicTimeTrigger.java, Target.java,

TimeScheduledEntry.java, TimeScheduler.java, TimeTrigger.java, Trigger.java, BinaryHeap.java, PriorityQueue.java, SynchronizedPriorityQueue.java, GetOpt.java, GetOptsException.java, IllegalArgumentException.java, MissingOptArgException.java (collectively, "Apache 1.1 Software"). Apache 1.1 Software is free software which is distributed under the terms of the following license:

License Version 1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgement: "This product includes software developed by the Apache Software Foundation (http://www.apache.org)." Alternatively, this acknowledgement may appear in the software itself, if and whenever such third-party acknowledgements normally appear.

4. The names "Commons", "Jakarta", "The Jakarta Project", "HttpClient", "log4j", "Xerces "Xalan", "Avalon", "Apache Avalon", "Avalon Cornerstone", "Avalon Framework", "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without specific prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without the prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMIPLIED WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTIBILITY, AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTIAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. The GetOpt.java, GetOptsException.java, IlligalArgumentException.java and MissingOptArgException.java software was originally based on software copyright (c) 2001, Sun Microsystems., http://www.sun.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

The preceding license only applies to the Apache 1.1 Software and does not apply to the Sterling Commerce Software or to any other Third-Party Software.

The Sterling Commerce Software is also distributed with or on the same storage media as the following software products (or components thereof): Ant, Antinstaller, Apache File Upload Package, Apache Commons Beans, Apache Commons BetWixt, Apache Commons Collection, Apache Commons Digester, Apache Commons IO, Apache Commons Lang., Apache Commons Logging, Apache Commons Net, Apache Jakarta Commons Pool, Apache Jakarta ORO, Lucene, Xerces version 2.7, Apache Log4J, Apache SOAP, Apache Struts and Apache Xalan 2.7.0, (collectively, "Apache 2.0 Software"). Apache 2.0 Software is free software which is distributed under the terms of the Apache License Version 2.0. A copy of License Version 2.0 is found in the following directory files for the individual pieces of the Apache 2.0 Software: installdir/jar/commons_upload/1_0/ CommonsFileUpload_License.txt,
installdir/jar/jetspeed/1_4/RegExp_License.txt,
installdir/ant/Ant_License.txt
<install>/jar/antInstaller/0_8/antinstaller_License.txt,
<install>/jar/commons_beanutils/1_7_0/commons-beanutils.jar (/META-INF/LICENSE.txt),
<install>/jar/commons_betwixt/0_8/commons-betwixt-0.8.jar (/META-INF/LICENSE.txt),

<install>/jar/commons_collections/3_2/LICENSE.txt,
<install>/jar/commons_digester/1_8/commons-digester-1.8.jar (/META-INF/LICENSE.txt),
<install>/jar/commons_io/1_4/LICENSE.txt,
<install>/jar/commons_lang/2_1/Commons_Lang_License.txt,
<install>/jar/commons_logging/1_0_4/commons-logging-1.0.4.jar (/META-INF/LICENSE.txt),
<install>/jar/commons_net/1_4_1/commons-net-1.4.1.jar (/META-INF/LICENSE.txt),
<install>/jar/smcfs/8.5/lucene-core-2.4.0.jar (/META-INF/LICENSE.txt),
<install>/jar/struts/2_0_11/struts2-core-2.0.11.jar (./LICENSE.txt),
<install>/jar/mesa/gisdav/WEB-INF/lib/Slide_License.txt,
<install>/mesa/studio/plugins/xerces_2.7_license.txt,
<install>/jar/commons_pool/1_2/Commons_License.txt,
<install>/jar/jakarta_oro/2_0_8/JakartaOro_License.txt,
<install>/jar/log4j/1_2_15/LOG4J_License.txt,
<install>/jar/xalan/2_7/Xalan_License.txt,
<install>/jar/soap/2_3_1/Apache_SOAP_License.txt

Unless otherwise stated in a specific directory, the Apache 2.0 Software was not modified. Neither the Sterling Commerce Software, modifications, if any, to Apache 2.0 Software, nor other Third Party Code is a Derivative Work or a Contribution as defined in License Version 2.0. License Version 2.0 applies only to the Apache 2.0 Software which is the subject of the specific directory file and does not apply to the Sterling Commerce Software or to any other Third Party Software. License Version 2.0 includes the following provision:

"Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License."

NOTICE file corresponding to the section 4 d of the Apache License, Version 2.0, in this case for the Apache Ant distribution. Apache Ant Copyright 1999-2008 The Apache Software Foundation. This product includes software developed by The Apache Software Foundation (http://www.apache.org/). This product includes also software developed by :

- the W3C consortium (http://www.w3c.org) ,

- the SAX project (http://www.saxproject.org)

The <sync> task is based on code Copyright (c) 2002, Landmark Graphics Corp that has been kindly donated to the Apache Software Foundation.

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.

- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.

- voluntary contributions made by Paul Eng on behalf of the Apache Software Foundation that were originally developed at iClick, Inc., software copyright (c) 1999.

NOTICE file corresponding to the section 4 d of the Apache License, Version 2.0, in this case for the Apache Lucene distribution. Apache Lucene Copyright 2006 The Apache Software Foundation. This product includes software developed by The Apache Software Foundation (http://www.apache.org/). The snowball stemmers in contrib/snowball/src/java/net/sf/snowball were developed by Martin Porter and Richard Boulton. The full snowball package is available from http://snowball.tartarus.org/

Ant-Contrib Software

The Sterling Commerce Software is distributed with or on the same storage media as the Anti-Contrib software (Copyright (c) 2001-2003 Ant-Contrib project. All rights reserved.) (the "Ant-Contrib Software"). The Ant-Contrib Software is free software which is distributed under the terms of the following license:

The Apache Software License, Version 1.1

DOM4J Software

THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The preceding license only applies to the Dom4j Software and does not apply to the Sterling Commerce Software, or any other Third-Party Software.

THE ECLIPSE SOFTWARE FOUNDATION

The Sterling Commerce Software is also distributed with or on the same storage media as the following software:

com.ibm.icu.nl1_3.4.4.v200606220026.jar, org.eclipse.ant.core.nl1_3.1.100.v200606220026.jar,
org.eclipse.ant.ui.nl1_3.2.0.v200606220026.jar, org.eclipse.compare.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.boot.nl1_3.1.100.v200606220026.jar,
org.eclipse.core.commands.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.contenttype.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.expressions.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.filebuffers.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.filesystem.nl1_1.0.0.v200606220026.jar,
org.eclipse.core.jobs.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.resources.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.runtime.compatibility.auth.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.runtime.compatibility.nl1_3.1.100.v200606220026.jar,
org.eclipse.core.runtime.nl1_3.2.0.v200606220026.jar,
org.eclipse.core.variables.nl1_3.1.100.v200606220026.jar,
org.eclipse.debug.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.debug.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.equinox.common.nl1_3.2.0.v200606220026.jar,
org.eclipse.equinox.preferences.nl1_3.2.0.v200606220026.jar,
org.eclipse.equinox.registry.nl1_3.2.0.v200606220026.jar,
org.eclipse.help.appserver.nl1_3.1.100.v200606220026.jar,
org.eclipse.help.base.nl1_3.2.0.v200606220026.jar, org.eclipse.help.nl1_3.2.0.v200606220026.jar,
org.eclipse.help.ui.nl1_3.2.0.v200606220026.jar, org.eclipse.jdt.apt.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.apt.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.core.manipulation.nl1_1.0.0.v200606220026.jar,
org.eclipse.jdt.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.debug.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.doc.isv.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.doc.user.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.junit4.runtime.nl1_1.0.0.v200606220026.jar,
org.eclipse.jdt.launching.nl1_3.2.0.v200606220026.jar, org.eclipse.jdt.nl1_3.2.0.v200606220026.jar,
org.eclipse.jdt.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.jface.databinding.nl1_1.0.0.v200606220026.jar,
org.eclipse.jface.nl1_3.2.0.v200606220026.jar, org.eclipse.jface.text.nl1_3.2.0.v200606220026.jar,
org.eclipse.ltk.core.refactoring.nl1_3.2.0.v200606220026.jar,
org.eclipse.ltk.ui.refactoring.nl1_3.2.0.v200606220026.jar,
org.eclipse.osgi.nl1_3.2.0.v200606220026.jar, org.eclipse.osgi.services.nl1_3.1.100.v200606220026.jar,
org.eclipse.osgi.util.nl1_3.1.100.v200606220026.jar, org.eclipse.pde.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.pde.doc.user.nl1_3.2.0.v200606220026.jar,
org.eclipse.pde.junit.runtime.nl1_3.2.0.v200606220026.jar,
org.eclipse.pde.nl1_3.2.0.v200606220026.jar, org.eclipse.pde.runtime.nl1_3.2.0.v200606220026.jar,
org.eclipse.pde.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.platform.doc.isv.nl1_3.2.0.v200606220026.jar,
org.eclipse.platform.doc.user.nl1_3.2.0.v200606220026.jar,

org.eclipse.rcp.nl1_3.2.0.v200606220026.jar, org.eclipse.search.nl1_3.2.0.v200606220026.jar,
org.eclipse.swt.nl1_3.2.0.v200606220026.jar, org.eclipse.team.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.team.cvs.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.team.cvs.ssh.nl1_3.2.0.v200606220026.jar,
org.eclipse.team.cvs.ssh2.nl1_3.2.0.v200606220026.jar,
org.eclipse.team.cvs.ui.nl1_3.2.0.v200606220026.jar, org.eclipse.team.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.text.nl1_3.2.0.v200606220026.jar, org.eclipse.ui.browser.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.cheatsheets.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.console.nl1_3.1.100.v200606220026.jar,
org.eclipse.ui.editors.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.externaltools.nl1_3.1.100.v200606220026.jar,
org.eclipse.ui.forms.nl1_3.2.0.v200606220026.jar, org.eclipse.ui.ide.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.intro.nl1_3.2.0.v200606220026.jar, org.eclipse.ui.navigator.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.navigator.resources.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.presentations.r21.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.views.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.views.properties.tabbed.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.workbench.nl1_3.2.0.v200606220026.jar,
org.eclipse.ui.workbench.texteditor.nl1_3.2.0.v200606220026.jar,
org.eclipse.update.configurator.nl1_3.2.0.v200606220026.jar,
org.eclipse.update.core.nl1_3.2.0.v200606220026.jar,
org.eclipse.update.scheduler.nl1_3.2.0.v200606220026.jar,
org.eclipse.update.ui.nl1_3.2.0.v200606220026.jar,
com.ibm.icu_3.4.4.1.jar,
org.eclipse.core.commands_3.2.0.I20060605-1400.jar,
org.eclipse.core.contenttype_3.2.0.v20060603.jar,
org.eclipse.core.expressions_3.2.0.v20060605-1400.jar,
org.eclipse.core.filesystem.linux.x86_1.0.0.v20060603.jar,
org.eclipse.core.filesystem_1.0.0.v20060603.jar, org.eclipse.core.jobs_3.2.0.v20060603.jar,
org.eclipse.core.runtime.compatibility.auth_3.2.0.v20060601.jar,
org.eclipse.core.runtime_3.2.0.v20060603.jar, org.eclipse.equinox.common_3.2.0.v20060603.jar,
org.eclipse.equinox.preferences_3.2.0.v20060601.jar, org.eclipse.equinox.registry_3.2.0.v20060601.jar,
org.eclipse.help_3.2.0.v20060602.jar, org.eclipse.jface.text_3.2.0.v20060605-1400.jar,
org.eclipse.jface_3.2.0.I20060605-1400.jar, org.eclipse.osgi_3.2.0.v20060601.jar,
org.eclipse.swt.gtk.linux.x86_3.2.0.v3232m.jar, org.eclipse.swt_3.2.0.v3232o.jar,
org.eclipse.text_3.2.0.v20060605-1400.jar,
org.eclipse.ui.workbench.texteditor_3.2.0.v20060605-1400.jar,
org.eclipse.ui.workbench_3.2.0.I20060605-1400.jar, org.eclipse.ui_3.2.0.I20060605-1400.jar,
runtime_registry_compatibility.jar, eclipse.exe, eclipse.ini, and startup.jar
(collectively, "Eclipse Software").
All Eclipse Software is distributed under the terms and conditions of the Eclipse Foundation Software User Agreement (EFSUA) and/or terms and conditions of the Eclipse Public License Version 1.0 (EPL) or other license agreements, notices or terms and conditions referenced for the individual pieces of the Eclipse Software, including without limitation "Abouts", "Feature Licenses", and "Feature Update Licenses" as defined in the EFSUA .

A copy of the Eclipse Foundation Software User Agreement is found at
<install_dir>/SI/repository/rcp/rcpdependencies/windows/eclipse/notice.html,
<install_dir>/SI/repository/rcp/rcpdependencies/windows/eclipse/plugins/notice.html,
<install_dir>/SI/repository/rcp/rcpdependencies/gtk.linux_x86/eclipse/notice.html, and
<install_dir>/SI/repository/rcp/rcpdependencies/gtk.linux_x86/eclipse/plugins/notice.html.

A copy of the EPL is found at
<install_dir>/SI/repository/rcp/rcpdependencies/windows/eclipse/plugins/epl-v10.htm,
<install_dir>/SI/repository/rcp/rcpdependencies/windows/eclipse/epl-v10.htm,
<install_dir>/SI/repository/rcp/rcpdependencies/gtk.linux.x86/eclipse/plugins/epl-v10.html, and
<install_dir>/SI/repository/rcp/rcpdependencies/gtk.linux.x86/eclipse/epl-v10.html.

The reference to the license agreements, notices or terms and conditions governing each individual piece of the Eclipse Software is found in the directory files for the individual pieces of the Eclipse Software as described in the file identified as installdir/SCI_License.txt.

These licenses only apply to the Eclipse Software and do not apply to the Sterling Commerce Software, or any other Third Party Software.

The Language Pack (NL Pack) piece of the Eclipse Software, is distributed in object code form. Source code is available at http://archive.eclipse.org/eclipse/downloads/drops/L-3.2_Language_Packs-200607121700/index.php. In the event the source code is no longer available from the website referenced above, contact Sterling Commerce at 978-513-6000 and ask for the Release Manager. A copy of this license is located at <install_dir>/SI/repository/rcp/rcpdependencies/windows/eclipse/plugins/epl-v10.htm and

<install_dir>/SI/repository/rcp/rcpdependencies/gtk.linux.x86/eclipse/plugins/epl-v10.html.

The org.eclipse.core.runtime_3.2.0.v20060603.jar piece of the Eclipse Software was modified slightly in order to remove classes containing encryption items. The org.eclipse.core.runtime_3.2.0.v20060603.jar was modified to remove the Cipher, CipherInputStream and CipherOutputStream classes and rebuild the org.eclipse.core.runtime_3.2.0.v20060603.jar.

Ehcache Software

The Sterling Commerce Software is also distributed with or on the same storage media as the ehache v.1.5 software (Copyright © 2003-2008 Luck Consulting Pty. Ltd.) ("Ehache Software"). Ehcache Software is free software which is distributed under the terms of the Apache License Version 2.0. A copy of License Version 2.0 is found in <install>/jar/smcfs/8.5/ehcache-1.5.0.jar (./LICENSE.txt).

The Ehcache Software was not modified. Neither the Sterling Commerce Software, modifications, if any, to the Ehcache Software, nor other Third Party Code is a Derivative Work or a Contribution as defined in License Version 2.0. License Version 2.0 applies only to the Ehcache Software which is the subject of the specific directory file and does not apply to the Sterling Commerce Software or to any other Third Party Software. License Version 2.0 includes the following provision:

"Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License."

EZMorph Software

The Sterling Commerce Software is also distributed with or on the same storage media as the EZMorph v. 1.0.4 software (Copyright © 2006-2008 Andres Almiray) ("EZMorph Software"). EZMorph Software is free software which is distributed under the terms of the Apache License Version 2.0. A copy of License Version 2.0 is found in <install>/jar/ezmorph/1_0_4/ezmorph-1.0.4.jar (./LICENSE.txt).

The EZMorph Software was not modified. Neither the Sterling Commerce Software, modifications, if any, to the EZMorph Software, nor other Third Party Code is a Derivative Work or a Contribution as defined in License Version 2.0. License Version 2.0 applies only to the EZMorph Software which is the subject of the specific directory file and does not apply to the Sterling Commerce Software or to any other Third Party Software. License Version 2.0 includes the following provision:

"Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License."

Firebug Lite Software

The Sterling Commerce Software is distributed with or on the same storage media as the Firebug Lite Software which is free software distributed under the terms of the following license:

Redistribution and use of this software in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Azer Koçulu. nor the names of any other contributors may be used to endorse or promote products derived from this software without specific prior written permission of Parakey Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

ICE SOFTWARE

The Sterling Commerce Software is distributed on the same storage media as the ICE Software (Copyright © 1997 ICE Engineering, Inc./Timothy Gerard Endres.) ("ICE Software"). The ICE Software is independent from and not linked or compiled with the Sterling Commerce Software. The ICE Software is a free software product which can be distributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License or any later version.

A copy of the GNU General Public License is provided at installdir/jar/jniregistry/1_2/ICE_License.txt. This license only applies to the ICE Software and does not apply to the Sterling Commerce Software, or any other Third Party Software.

The ICE Software was modified slightly in order to fix a problem discovered by Sterling Commerce involving the RegistryKey class in the RegistryKey.java in the JNIRegistry.jar. The class was modified to comment out the finalize () method and rebuild of the JNIRegistry.jar file.

Source code for the bug fix completed by Sterling Commerce on January 8, 2003 is located at: installdir/jar/jniregistry/1_2/RegistryKey.java. Source code for all other components of the ICE Software is located at http://www.trustice.com/java/jnireg/index.shtml.

The ICE Software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

JBOSS SOFTWARE

The Sterling Commerce Software is distributed on the same storage media as the JBoss Software (Copyright © 1999-2002 JBoss.org) ("JBoss Software"). The JBoss Software is independent from and not linked or compiled with the Sterling Commerce Software. The JBoss Software is a free software product which can be distributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License or any later version.

A copy of the GNU Lesser General Public License is provided at:
<install_dir>\jar\jboss\4_2_0\LICENSE.html

This license only applies to the JBoss Software and does not apply to the Sterling Commerce Software, or any other Third Party Software.

The JBoss Software is not distributed by Sterling Commerce in its entirety. Rather, the distribution is limited to the following jar files: el-api.jar, jasper-compiler-5.5.15.jar, jasper-el.jar, jasper.jar, jboss-common-client.jar, jboss-j2ee.jar, jboss-jmx.jar, jboss-jsr77-client.jar, jbossmq-client.jar,

jnpserver.jar, jsp-api.jar, servlet-api.jar, tomcat-juli.jar.

The JBoss Software was modified slightly in order to allow the ClientSocketFactory to return a socket connected to a particular host in order to control the host interfaces, regardless of whether the ClientSocket Factory specified was custom or note. Changes were made to org.jnp..server.Main. Details concerning this change can be found at http://sourceforge.net/tracker/?func=detail&aid=1008902&group_id=22866&atid=376687.

Source code for the modifications completed by Sterling Commerce on August 13, 2004 is located at: http://sourceforge.net/tracker/?func=detail&aid=1008902&group_id=22866&atid=376687. Source code for all other components of the JBoss Software is located at http://www.jboss.org.

JGO SOFTWARE

The Sterling Commerce Software is distributed with, or on the same storage media, as certain redistributable portions of the JGo Software provided by Northwoods Software Corporation under a commercial license agreement (the "JGo Software"). The JGo Software is provided subject to the disclaimers set forth above and the following notice:

U.S. Government Restricted Rights

The JGo Software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (C)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (C)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor / manufacturer of the JGo Software is Northwoods Software Corporation, 142 Main St., Nashua, NH 03060.

JSLib Software

The Sterling Commerce Software is distributed with or on the same storage media as the JSLib software product (Copyright (c) 2003-2009 Mozdev Group, Inc.) ("JSLib Software"). The JSLib Software is distributed under the terms of the MOZILLA PUBLIC LICENSE Version 1.1. A copy of this license is located at <install>\repository\eardata\platform_uifwk_ide\war\designer\MPL-1.1.txt. The JSLib Software code is distributed in source form and is located at http://jslib.mozdev.org/installation.html. Neither the Sterling Commerce Software nor any other Third-Party Code is a Modification or Contribution subject to the Mozilla Public License. Pursuant to the terms of the Mozilla Public License, the following notice applies only to the JSLib Software (and not to the Sterling Commerce Software or any other Third-Party Software):

"The contents of the file located at http://www.mozdev.org/source/browse/jslib/ are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/MPL-1.1.html.

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is Mozdev Group, Inc. code. The Initial Developer of the Original Code is Mozdev Group, Inc. Portions created by_Mozdev Group, Inc. are Copyright © 2003 Mozdev Group, Inc. All Rights Reserved. Original Author: Pete Collins <pete@mozdev.org>one Contributor(s):_____none listed_____.

Alternatively, the contents of this file may be used under the terms of the ____ license (the "[___] License"), in which case the provisions of [___] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [___] License and not allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [___] License. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the [___] License."

The preceding license only applies to the JSLib Software and does not apply to the Sterling Commerce Software, or any other Third-Party Software.

Json Software

The Sterling Commerce Software is also distributed with or on the same storage media as the Json 2.2.2 software (Copyright © 2006-2008 Json-lib) ("Json Software"). Json Software is free software which is distributed under the terms of the Apache License Version 2.0. A copy of License Version 2.0 is found in <install>/jar/jsonlib/2_2_2/json-lib-2.2.2-jdk13.jar.

This product includes software developed by Douglas Crockford (http://www.crockford.com).

The Json Software was not modified. Neither the Sterling Commerce Software, modifications, if any, to the Json Software, nor other Third Party Code is a Derivative Work or a Contribution as defined in License Version 2.0. License Version 2.0 applies only to the Json Software which is the subject of the specific directory file and does not apply to the Sterling Commerce Software or to any other Third Party Software. License Version 2.0 includes the following provision:

"Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License."

Purple Technology

The Sterling Commerce Software is distributed with or on the same storage media as the Purple Technology Software (Copyright (c) 1995-1999 Purple Technology, Inc.) ("Purple Technology Software"), which is subject to the following license:

Copyright (c) 1995-1999 Purple Technology, Inc. All rights reserved.

PLAIN LANGUAGE LICENSE: Do whatever you like with this code, free of charge, just give credit where credit is due. If you improve it, please send your improvements to alex@purpletech.com. Check http://www.purpletech.com/code/ for the latest version and news.

LEGAL LANGUAGE LICENSE: Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of the authors and the names "Purple Technology," "Purple Server" and "Purple Chat" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact server@purpletech.com.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS AND PURPLE TECHNOLOGY "AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR PURPLE TECHNOLOGY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The preceding license only applies to the Purple Technology Software and does not apply to the Sterling Commerce Software, or any other Third Party Software.

Rico Software

The Sterling Commerce Software is also distributed with or on the same storage media as the Rico.js software (Copyright © 2005 Sabre Airline Solutions) ("Rico Software"). Rico Software is free software

which is distributed under the terms of the Apache License Version 2.0. A copy of License Version 2.0 is found in <install>/repository/eardata/platform/war/ajax/scripts/Rico_License.txt.

The Rico Software was not modified. Neither the Sterling Commerce Software, modifications, if any, to the Rico Software, nor other Third-Party Code is a Derivative Work or a Contribution as defined in License Version 2.0. License Version 2.0 applies only to the Rico Software which is the subject of the specific directory file and does not apply to the Sterling Commerce Software or to any other Third-Party Software. License Version 2.0 includes the following provision:

"Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License."

Rhino Software

The Sterling Commerce Software is distributed with or on the same storage media as the Rhino js.jar (Copyright (c) 1998-2009 Mozilla.org.) ("Rhino Software"). A majority of the source code for the Rhino Software is dual licensed under the terms of the MOZILLA PUBLIC LICENSE Version 1.1. or the GPL v. 2.0. Additionally, some files (at a minimum the contents of toolsrc/org/Mozilla/javascript/toolsdebugger/treetable) are available under another license as set forth in the directory file for the Rhino Software.

Sterling Commerce's use and distribution of the Rhino Software is under the Mozilla Public License. A copy of this license is located at <install>/3rdParty/rico license.doc. The Rhino Software code is distributed in source form and is located at http://mxr.mozilla.org/mozilla/source/js/rhino/src/. Neither the Sterling Commerce Software nor any other Third-Party Code is a Modification or Contribution subject to the Mozilla Public License. Pursuant to the terms of the Mozilla Public License, the following notice applies only to the Rhino Software (and not to the Sterling Commerce Software or any other Third-Party Software):

"The contents of the file located at <install>/jar/rhino/1_7R1/js.jar are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/.

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is Rhino code, released May 6, 1999. The Initial Developer is Netscape Communications Corporation. Portions created by the Initial Developer are Copyright © 1997-1999. All Rights Reserved. Contributor(s):_____none listed.

The preceding license only applies to the Rico Software and does not apply to the Sterling Commerce Software, or any other Third-Party Software.

Sun Microsystems

The Sterling Commerce Software is distributed with or on the same storage media

as the following software products (or components thereof): Sun JMX, and Sun JavaMail (collectively, "Sun Software"). Sun Software is free software which is distributed under the terms of the licenses issued by Sun which are included in the directory files located at:

SUN COMM JAR - <install>/Applications/Foundation/lib

SUN ACTIVATION JAR - <install>/ Applications/Foundation/lib

SUN JavaMail - <install>/jar/javamail/1_4/LICENSE.txt

The Sterling Commerce Software is also distributed with or on the same storage media as the Web-app_2_3.dtd software (Copyright © 2007 Sun Microsystems, Inc.) ("Web-App Software"). Web-App Software is free software which is distributed under the terms of the Common Development

and Distribution License ("CDDL"). A copy of the CDDL is found in http://kenai.com/projects/javamail/sources/mercurial/show.

The source code for the Web-App Software may be found at: <install>/3rdParty/sun/javamail-1.3.2/docs/JavaMail-1.2.pdf

Such licenses only apply to the Sun product which is the subject of such directory and does not apply to the Sterling Commerce Software or to any other Third Party Software.

The Sterling Commerce Software is also distributed with or on the same storage media as the Sun Microsystems, Inc. Java (TM) look and feel Graphics Repository ("Sun Graphics Artwork"), subject to the following terms and conditions:

Copyright 2000 by Sun Microsystems, Inc. All Rights Reserved.

Sun grants you ("Licensee") a non-exclusive, royalty free, license to use, and redistribute this software graphics artwork, as individual graphics or as a collection, as part of software code or programs that you develop, provided that i) this copyright notice and license accompany the software graphics artwork; and ii) you do not utilize the software graphics artwork in a manner which is disparaging to Sun. Unless enforcement is prohibited by applicable law, you may not modify the graphics, and must use them true to color and unmodified in every way.

This software graphics artwork is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE GRAPHICS ARTWORK.

IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE GRAPHICS ARTWORK, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

If any of the above provisions are held to be in violation of applicable law, void, or unenforceable in any jurisdiction, then such provisions are waived to the extent necessary for this Disclaimer to be otherwise enforceable in such jurisdiction.

The preceding license only applies to the Sun Graphics Artwork and does not apply to the Sterling Commerce Software, or any other Third Party Software.

WARRANTY DISCLAIMER

This documentation and the Sterling Commerce Software which it describes are licensed either "AS IS" or with a limited warranty, as set forth in the Sterling Commerce license agreement. Other than any limited warranties provided, NO OTHER WARRANTY IS EXPRESSED AND NONE SHALL BE IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR USE OR FOR A PARTICULAR PURPOSE. The applicable Sterling Commerce entity reserves the right to revise this publication from time to time and to make changes in the content hereof without the obligation to notify any person or entity of such revisions or changes.

The Third Party Software is provided "AS IS" WITHOUT ANY WARRANTY AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. FURTHER, IF YOU ARE LOCATED OR ACCESSING THIS SOFTWARE IN THE UNITED STATES, ANY EXPRESS OR IMPLIED WARRANTY REGARDING TITLE OR NON-INFRINGEMENT ARE DISCLAIMED.

Without limiting the foregoing, the ICE Software and JBoss Software are distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Contents

3 Integrating with Stand-Alone Sterling WMS

12 Integrating with Financial Systems

13 Rapid Deployment Features

Index

Preface

This manual describes how Selling and Fulfillment Foundation integrates with other Sterling Commerce offerings, such as Distributed Center Solution and third-party applications.

Intended Audience

This manual is intended for use by those who are responsible for integrating Selling and Fulfillment Foundation with other applications.

Structure

This manual contains the following sections:

Chapter 1, "Introduction"

This chapter discusses integration in general terms and provides an overview of the application integration architecture.

Chapter 2, "Integrating with the Distribution Center System"

This chapter describes how to integrate Selling and Fulfillment Foundation with the Distribution Center System (DCS), Sterling Commerce's previously released Distribution Center Solution.

Chapter 3, "Integrating with Stand-Alone Sterling WMS"

This chapter describes how to integrate Selling and Fulfillment Foundation with a stand-alone Sterling Warehouse Management System.

Chapter 4, "Integrating with Third-Party Warehouse Management Systems"

This chapter describes how to integrate Selling and Fulfillment Foundation with third-party warehouse management systems.

Chapter 5, "Integrating with the Loftware Print Server and Label Manager"

This chapter explains how to integrate the Sterling Warehouse Management System with the Loftware Print Server and Loftware Label Manager.

Chapter 6, "Integrating with the Parcel Carrier Adapters"

This chapter explains how to integrate the Sterling Warehouse Management System with the Parcel Carrier Adapters (Carrier Adapter).

Chapter 7, "Integrating with Material Handling Equipment"

This chapter explains how to integrate the Sterling Warehouse Management System with various material handling equipment (MHE), including the Mettler Toledo Weighing Scale.

Chapter 8, "Integrating with Enterprise Resource Planning Systems"

This chapter explains how to integrate the Sterling Warehouse Management System with Enterprise Resource Planning (ERP) systems to utilize any additional functions that are available in the existing environment.

Chapter 9, "Integrating with Point of Sale Systems"

This chapter explains how to integrate the Sterling Warehouse Management System with point-of-sale systems in stores.

Chapter 10, "Integrating User and Item Data with External Systems"

This chapter explains how Selling and Fulfillment Foundation enables you to integrate with external systems used to sell products, through multiple channels.

Chapter 11, "Integrating with JMS Systems"

This chapter explains how to configure third-party message queueing applications for BEA WebLogic JMS and IBM WebSphere MQ JMS.

Chapter 12, "Integrating with Financial Systems"

This chapter explains how to integrate the Selling and Fulfillment Foundation inventory cost management interfaces with your financial system.

Chapter 13, "Rapid Deployment Features"

This chapter explains the rapid deployment features in the Sterling Warehouse Management System and how to utilize these for rapid deployment of Selling and Fulfillment Foundation.

Selling and Fulfillment Foundation Documentation

For more information about the Selling and Fulfillment Foundation components, see the following manuals:

- *Selling and Fulfillment Foundation: Release Notes*
- *Selling and Fulfillment Foundation: Installation Guide*
- *Selling and Fulfillment Foundation: Upgrade Guide*
- *Selling and Fulfillment Foundation: Configuration Deployment Tool Guide*
- *Selling and Fulfillment Foundation: Performance Management Guide*
- *Selling and Fulfillment Foundation: High Availability Guide*
- *Selling and Fulfillment Foundation: System Management Guide*
- *Selling and Fulfillment Foundation: Localization Guide*
- *Selling and Fulfillment Foundation: Customization Basics Guide*
- *Selling and Fulfillment Foundation: Customizing APIs Guide*
- *Selling and Fulfillment Foundation: Customizing Console JSP Interface for End User Guide*
- *Selling and Fulfillment Foundation: Customizing the RCP Interface Guide*

- *Selling and Fulfillment Foundation: Customizing User Interfaces for Mobile Devices Guide*
- *Selling and Fulfillment Foundation: Customizing Web UI Framework Guide*
- *Selling and Fulfillment Foundation: Customizing Swing Interface Guide*
- *Selling and Fulfillment Foundation: Extending the Condition Builder Guide*
- *Selling and Fulfillment Foundation: Extending the Database Guide*
- *Selling and Fulfillment Foundation: Extending Transactions Guide*
- *Selling and Fulfillment Foundation: Using Sterling RCP Extensibility Tool Guide*
- *Selling and Fulfillment Foundation: Integration Guide*
- *Selling and Fulfillment Foundation: Product Concepts Guide*
- *Sterling Warehouse Management™ System: Concepts Guide*
- *Selling and Fulfillment Foundation: Application Platform Configuration Guide*
- *Sterling Distributed Order Management™: Configuration Guide*
- *Sterling Supply Collaboration: Configuration Guide*
- *Sterling Global Inventory Visibility™: Configuration Guide*
- *Catalog Management™: Configuration Guide*
- *Sterling Logistics Management: Configuration Guide*
- *Sterling Reverse Logistics™: Configuration Guide*
- *Sterling Warehouse Management System: Configuration Guide*
- *Selling and Fulfillment Foundation: Application Platform User Guide*
- *Sterling Distributed Order Management: User Guide*
- *Sterling Supply Collaboration: User Guide*
- *Sterling Global Inventory Visibility: User Guide*
- *Sterling Logistics Management: User Guide*

- *Sterling Reverse Logistics: User Guide*
- *Sterling Warehouse Management System: User Guide*
- *Selling and Fulfillment Foundation: Mobile Application User Guide*
- *Selling and Fulfillment Foundation: Business Intelligence Guide*
- *Selling and Fulfillment Foundation: Javadocs*
- *Sterling Selling and Fulfillment Suite™: Glossary*
- *Parcel Carrier: Adapter Guide*
- *Selling and Fulfillment Foundation: Multitenant Enterprise Guide*
- *Selling and Fulfillment Foundation: Password Policy Management Guide*
- *Selling and Fulfillment Foundation: Properties Guide*
- *Selling and Fulfillment Foundation: Catalog Management Concepts Guide*
- *Selling and Fulfillment Foundation: Pricing Concepts Guide*
- *Business Center: Item Administration Guide*
- *Business Center: Pricing Administration Guide*
- *Business Center: Customization Guide*
- *Business Center: Localization Guide*

Conventions

The following conventions may be used in this manual:

Convention	Meaning
. . .	Ellipsis represents information that has been omitted.
< >	Angle brackets indicate user-supplied input.
`mono-spaced text`	Mono-spaced text indicates a file name, directory path, attribute name, or an inline code example or command.

Convention	Meaning
/ or \	Slashes and backslashes are file separators for Windows, UNIX, and Linux operating systems. The file separator for the Windows operating system is "\" and the file separator for UNIX and Linux systems is "/". The UNIX convention is used unless otherwise mentioned.
<INSTALL_DIR>	User-supplied location of the Selling and Fulfillment Foundation installation directory. This is only applicable for Release 8.0 or later.
<INSTALL_DIR_OLD>	User-supplied location of the Selling and Fulfillment Foundation installation directory (for Release 8.0 or later). **Note:** This is applicable only for users upgrading from Release 8.0 or later.
<YANTRA_HOME>	User-supplied location of the Sterling Supply Chain Applications installation directory. This is only applicable for Releases 7.7, 7.9, and 7.11.
<YANTRA_HOME_OLD>	User-supplied location of the Sterling Supply Chain Applications installation directory (for Releases 7.7, 7.9, or 7.11). **Note:** This is applicable only for users upgrading from Releases 7.7, 7.9, or 7.11.
<YFS_HOME>	For Releases 7.3, 7.5, and 7.5 SP1, this is the user-supplied location of the Sterling Supply Chain Applications installation directory. For Releases 7.7, 7.9, and 7.11, this is the user-supplied location of the <YANTRA_HOME>/Runtime directory. For Release 8.0 or above, the <YANTRA_HOME>/Runtime directory is no longer used and this is the same location as <INSTALL_DIR>.
<YFS_HOME_OLD>	This is the <YANTRA_HOME>/Runtime directory for Releases 7.7, 7.9, or 7.11. **Note:** This is only applicable for users upgrading from Releases 7.7, 7.9, or 7.11.

Convention	Meaning
<ANALYTICS_HOME>	User-supplied location of the Sterling Analytics installation directory. **Note:** This convention is used only in the *Selling and Fulfillment Foundation: Business Intelligence Guide*.
<COGNOS_HOME>	User-supplied location of the IBM Cognos 8 Business Intelligence installation directory. **Note:** This convention is used only in the *Selling and Fulfillment Foundation: Business Intelligence Guide*.
<MQ_JAVA_INSTALL_PATH>	User-supplied location of the IBM WebSphere® MQ Java components installation directory. **Note:** This convention is used only in the *Selling and Fulfillment Foundation: System Manangement and Administration Guide*.
<DB>	Refers to Oracle®, IBM DB2®, or Microsoft SQL Server® depending on the database server.
<DB_TYPE>	Depending on the database used, considers the value oracle, db2, or sqlserver.

Note: The Selling and Fulfillment Foundation documentation set uses the following conventions in the context of the product name:

- Yantra is used for Release 7.7 and earlier.
- Sterling Supply Chain Applications is used for Releases 7.9 and 7.11.
- Sterling Multi-Channel Fulfillment Solution is used for Releases 8.0 and 8.2.
- Selling and Fulfillment Foundation is used for Release 8.5.

1

Introduction

This guide describes how to integrate Selling and Fulfillment Foundation with other Sterling Commerce offerings, such as the Distributed Center Solution and third-party applications through the services defined using the Service Definition Framework. For more information about defining specific services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Selling and Fulfillment Foundation provides integration with:

- The Distribution Center System
- Third-Party Warehouse Management System
- The Parcel Carrier Adapters
- Loftware Print Server and Label Manager
- Material Handling Equipment
- Enterprise Resource Planning Systems
- Point of Sale Systems
- JMS Systems
- Financial Systems
- Interface Field Mapping Documents

Note: If you try to configure more than one action serially using the Service Definition Framework, the Applications Manager throws an error message, "A continue link must be attached to the next condition or action."

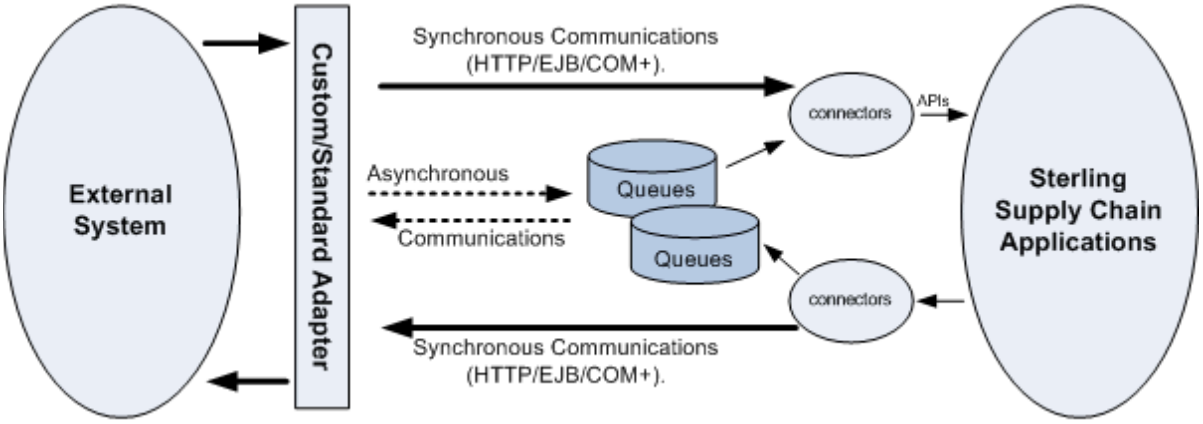
Therefore, Sterling Commerce recommends you to group these actions and replace them with one service.

1.1 Application Integration Architecture

Adapters connect to external systems through the Service Definition Framework for data transformation.

Figure 1–1, "Integration Architecture" shows how the Service Definition Framework fits into the applications integration architecture of Selling and Fulfillment Foundation, the various adapters that perform data transformation, and the goals of the transformations. For more information about the adapter used within Selling and Fulfillment Foundation, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Figure 1–1 Integration Architecture



1.2 Integration with Warehouse Management Systems

Selling and Fulfillment Foundation provides real-time integration with the Sterling Warehouse Management System (Sterling WMS).

Selling and Fulfillment Foundation supports integration with the Distribution Center System (DCS). The DCS application supports warehouse inventory, distribution, returns and activities. Typically, it is used in distribution centers for fulfilling large numbers of orders, with items required in quantities of a case or less. For information about integrating Selling and Fulfillment Foundation with the Distribution Center System (DCS), see Chapter 2, "Integrating with the Distribution Center System".

Selling and Fulfillment Foundation also supports integration with third-party warehouse management systems. For information about integrating Selling and Fulfillment Foundation with third-party warehouse management systems, see Chapter 4, "Integrating with Third-Party Warehouse Management Systems".

1.3 Integration with the Parcel Carrier Adapters

Selling and Fulfillment Foundation provides integration with the Parcel Carrier Adapters (Carrier Adapter), which manages all the carrier-integration related functions of Selling and Fulfillment Foundation. Selling and Fulfillment Foundation interfaces with the Carrier Adapter to use its carrier-integration functions.

The Carrier Adapter is regularly updated with the latest carrier data, such as rates and special services, and hence can act as a centralized carrier-integration database and business rules manager. The Carrier Adapter helps companies to quickly meet the changing requirements initiated by both carriers and customers, in the most efficient way.

The Carrier Adapter has a data driven design. The functionality is defined in terms of the relation between data elements stored in the database. Carriers having similar functionality can be incorporated into an installation with minimal engineering effort.

The Carrier Adapter is now integrated into Selling and Fulfillment Foundation. For more information about the Carrier Adapter and how to configure it, see the *Parcel Carrier: Adapter Guide*.

1.4 Integration with the Loftware Print Server and Label Manager

Selling and Fulfillment Foundation provides integration with the Loftware Print Server and Loftware Label Manager for printing reports and designing custom labels. You can also design custom print services using the Service Definition Framework. For more information about the print server and label manager, see Chapter 5, "Integrating with the Loftware Print Server and Label Manager".

1.5 Integration with Material Handling Equipment

Selling and Fulfillment Foundation provides integration with various material handling equipment (MHE). The automation enabled through the integration enables increased efficiency in various processes of a warehouse. For information about integrating Selling and Fulfillment

Foundation with MHE, see Chapter 7, "Integrating with Material Handling Equipment".

1.6 Integration with Enterprise Resource Planning Systems

Selling and Fulfillment Foundation provides integration with the Enterprise Resource Planning (ERP) systems. An ERP system is a packaged business software system that allows a company to automate and integrate the majority of its business processes. For information about integrating Selling and Fulfillment Foundation with ERP Systems, see Chapter 8, "Integrating with Enterprise Resource Planning Systems".

1.7 Integration with Point of Sale Systems

Selling and Fulfillment Foundation provides integration with the point-of-sale systems used in stores for product check-outs and returns from customers. For information about integrating Selling and Fulfillment Foundation with point-of-sale systems, see Chapter 9, "Integrating with Point of Sale Systems".

1.8 Integration with JMS Systems

In order for some service nodes to communicate with external applications, external message queueing software must be configured. For information about configuring the third-party message queueing applications, see Chapter 11, "Integrating with JMS Systems".

1.9 Integration with Financial Systems

To use the data captured using the Selling and Fulfillment Foundation Inventory Cost Management feature with your financial system, you must load the Initial Inventory Cost Data and configure process-specific events.

For information about integrating Selling and Fulfillment Foundation with financial systems, see Chapter 12, "Integrating with Financial Systems".

1.10 Rapid Deployment Features

This chapter explains the Selling and Fulfillment Foundation Rapid Deployment Features, and how to utilize these for the rapid deployment of Selling and Fulfillment Foundation. For information about Rapid Deployment Features, see Chapter 13, "Rapid Deployment Features".

2

Integrating with the Distribution Center System

Note: For the Selling and Fulfillment Foundation Release 8.5, the integration described in this chapter has been deprecated.

The Distribution Center System (DCS) is a previously released product that supports warehouse activities such as the inventory of items and the distribution of packages. Typically, DCS operates in distribution centers fulfilling large numbers of orders for items required in quantities of a case or less. It supports both real-time radio frequency (RF) transactions and paper-based transactions.

Selling and Fulfillment Foundation provides an interface-based integration with DCS Release 6.2 SP2 for the following operations:

- DCS Purchase Order Interface
- DCS Order Release Interface
- DCS Inventory Interface

Important: Selling and Fulfillment Foundation and Distribution Center System integration requires that the DCS interface format conforms to the field size and start positions at *each* of the integration points as detailed in the tables in this chapter. For information about configuring DCS, see the DCS 6.2 documentation. In addition, you must configure Selling and Fulfillment Foundation as described in this chapter.

Note: Selling and Fulfillment Foundation is certified for DCS 6.2 Service Pack 3 Hot Fix 13 and above.

2.1 DCS Purchase Order Interface

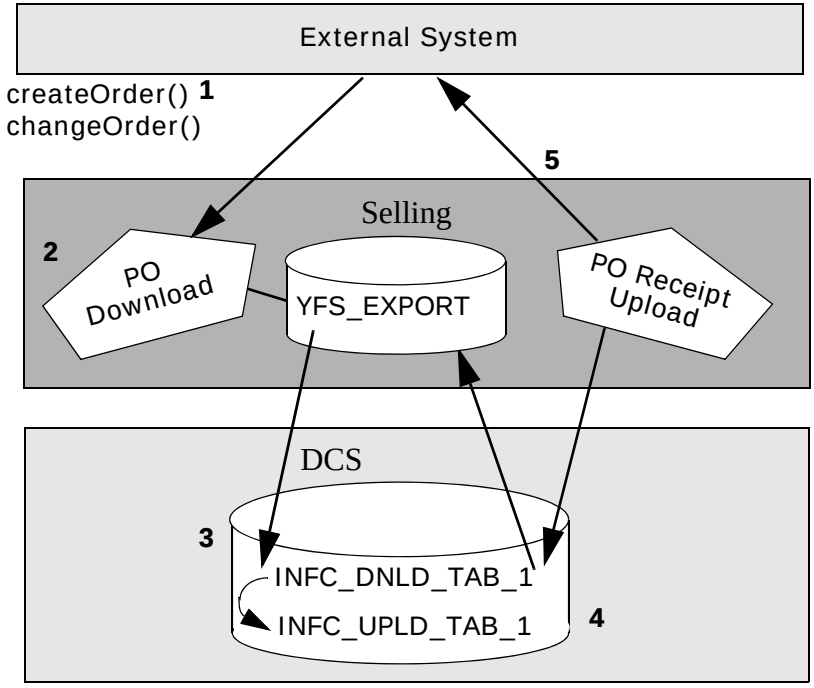
When a Purchase Order is created on Selling and Fulfillment Foundation (either by importing Purchase Orders created by external order management systems or by using the Application Console), DCS integration enables you to publish that data to the DCS. The integration interface uses the Purchase Order Download and Upload time-triggered transactions. For more information about these transactions, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

2.1.1 Purchase Order Workflow

Figure 2–1 illustrates the workflow for the Purchase Order Download and Purchase Order Upload time-triggered transactions that send Purchase Order data between an external system and the DCS using the Selling and Fulfillment Foundation.

For step-by-step procedures, see "Configuring the Purchase Order Time-Triggered Transactions".

Figure 2–1 Workflow for Purchase Order Transactions



1. An external Purchase Order system invokes Selling and Fulfillment Foundation `createOrder()` API to create a Purchase Order for a DCS receiving node. A Purchase Order is created and the order status becomes `Created (1100)`.

   Any future modifications to the original Purchase Order by an external system are made by invoking the `changeOrder()` API.

2. The ON_SUCCESS event of the `createOrder()` or `changeOrder()` API invokes an action, which in turn invokes a service called `YantraWMSPODownloadService`. This service publishes data into the YFS_EXPORT table with `YantraWMSPODownloadService` as the flow name.

3. The Purchase Order Download time-triggered transaction takes the record from the YFS_EXPORT table and inserts it into the DCS interface INFC_DNLD_TAB_1 table. Before downloading to the DCS, the transaction verifies that the ship node assigned to the Purchase

Order line is a DCS ship node. If the ship node is not a DCS ship node, the transaction marks the record as processed and takes no further action.

4. The vendor sends an advance shipment notice (ASN) to the DCS for shipping the items on the Purchase Order. When items are received at the receiving node, the DCS uploads Purchase Order Receipt records to the DCS interface table.

   The Purchase Order Receipt Upload time-triggered transaction picks up the Purchase Order Receipt records from the DCS upload interface table and calls the `receiveOrder()` API with the Receive Purchase Order transaction. The status of items received is changed to `Received` (3900).

   If the Purchase Order is to be downloaded to Selling and Fulfillment Foundation from an external system, the ON_SUCCESS event of the Receive Purchase Order transaction can be configured to invoke an action to publish the Purchase Order Receipt data to the YFS_EXPORT table.

   The data is then uploaded back to the external Purchase Order system.

2.1.2 Understanding Purchase Order Transactions

When deciding how to implement the DCS Purchase Order functionality, keep in mind the expected behaviors associated with the Purchase Order transactions when used in the following situations:

Supply Type Behavior

When the Purchase Order Status is `Created` (1100), the quantity in the YFS_INVENTORY_SUPPLY table is added to the `PO_PLACED` supply type.

When the Purchase Order Status is moved to `Order Received` (3900), the quantity in the YFS_INVENTORY_SUPPLY table moves from supply type `PLANNED_PO` to supply type `ONHAND`. This is the default behavior and can be reconfigured as needed.

Creating a Purchase Order

Selling and Fulfillment Foundation requires the Purchase Order number it passes to the DCS to be unique across all Enterprises. While Selling and Fulfillment Foundation permits the length of the Order number to be up

to 40 characters, the DCS limits the length of both the Order and the Purchase Order number to a maximum of 13 characters. In addition, to comply with the DCS requirements, Purchase Order numbers may contain any combination of numbers and upper-case alphabetic characters; lower-case alphabetic characters are not permitted.

All Purchase Order lines must use consecutive prime line numbers, with all subline numbers as = 1. The PODTL Record Type does not take in subline numbers. For more information see Section 2.1.5.2, "PODTL - Purchase Order Download Detail".

When integrating with the DCS, all the advance shipment notifications (Purchase Order Receipt) created and uploaded to the DCS interface table are only for the Purchase Orders that were initially downloaded from Selling and Fulfillment Foundation.

When passing parameters to the DCS interface table, be sure that the length does not exceed that which is enabled by the DCS Purchase Order header and detail records.

Parameters are passed to the DCS when Selling and Fulfillment Foundation downloads Purchase Orders from an external system.

Note that the date for the Estimated Time of Arrival in the DCS is the Requested Delivery Date at the time of the Purchase Order creation on Selling and Fulfillment Foundation.

Modifying a Purchase Order

Only the following modifications to a Purchase Order are permitted:

- Changing the quantity
- Changing the requested delivery date
- Adding one or more lines

Splitting a Purchase Order

A Purchase Order cannot be split across receiving nodes, even for the same DCS. One Purchase Order is created for only one installation of the DCS and only one of its receiving nodes. All Purchase Order lines must have the same receiving node.

Canceling a Purchase Order or Line

While it is not possible to explicitly cancel a Purchase Order or Purchase Order line, if the quantity zero (0) is passed from Selling and Fulfillment Foundation, the Purchase Order modification time-triggered transaction interprets it as closing the order line on the DCS. For the DCS, the results of canceling a line is the same as closing a line. If the ordered quantity becomes zero, Selling and Fulfillment Foundation does not permit any further changes to the line.

If Selling and Fulfillment Foundation receives a Purchase Order receipt from the DCS on a line that has been cancelled by the external Purchase OrderPurchase Order system (due to interface timing issues), it raises an exception in Selling and Fulfillment Foundation.

Receiving Goods into Inventory

The warehouse receiving the goods is identified as the Receiving Ship Node on the Purchase Order.

The specific goods that a node receives must match the description of the line items on the original Purchase Order.

Receipt overage is controlled by the DCS by setting up an overage receipt percentage based on your receiving preferences for each line type downloaded. Configure the overage receipt percentage in the Applications Manager by navigating to Applications > Supply Collaboration > Document Specific > Purchase Order > Receipt > Receiving Preference. On the Search Results panel choose  .

The overage percentage is controlled in the DCS. The Selling and Fulfillment Foundation percentage is applied during receipt. This means that the receiving node for the DCS cannot receive quantity in excess of the overage percentage specified. Also, by the same logic, Selling and Fulfillment Foundation does not permit new order quantities to be modified to be below the quantity already received for that Purchase Order line.

Be sure to configure the received quantity so that Selling and Fulfillment Foundation and all the DCS work together. For example, if received quantity is configured as `ONHAND` in Selling and Fulfillment Foundation, it should be configured as `Allocatable` in all the DCS installations.

Table 2–1 Selling and Fulfillment Foundation and DCS Received Quantity Mapping

Quantity Description	Selling and Fulfillment Foundation	DCS
Available items	ONHAND	Allocatable
Items kept in reserve	HELD	Non Allocatable

In addition, a node cannot receive goods against a cancelled or closed line.

Inventory is increased in the onhand supply when Selling and Fulfillment Foundation receives and processes the Purchase Order Receipt Upload transaction from the DCS, which must not be configured to upload separate inventory transactions for receipts.

For more information about configuring DCS Inventory Updates, see the DCS documentation.

2.1.3 Configuring the Purchase Order Time-Triggered Transactions

Setting up a Purchase Order involves configuring and scheduling time-triggered transactions and configuring the pipeline that the Purchase Order should use. You also should check your Oracle database configuration.

To configure the Purchase Order time-triggered transactions:

1. Check that Oracle database links are created for each DCS receiving node for which you want to create a Purchase Order. Selling and Fulfillment Foundation maintains the links and views to the DCS interface table for each receiving node in the DCS system.
2. Configure the Purchase Order Download and Purchase Order Receipt Upload time-triggered transactions. For detailed information about configuring these transactions, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Note: While the WMS Purchase Order Download time-triggered transaction does not require a ship node to be configured for downloading, you do need to configure agent criteria for each ship node from which a WMS Purchase Order Receipt Upload is to be processed.

3. Configure the pipeline using the directions in Section 2.1.4, "Configuring the Purchase Order Pipeline".
4. Schedule the time intervals for running the Purchase Order time-triggered transactions, as described in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

   The Purchase Order Download transaction writes the POHDR and PODTL records into the DCS download interface table.

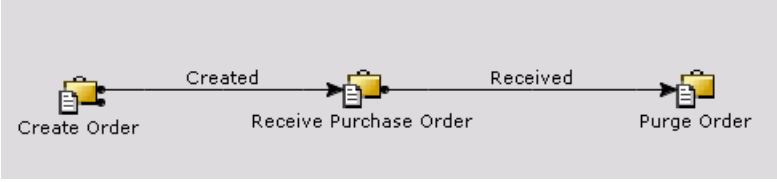
   The Purchase Order Receipt Upload transaction reads the RCPHDR and RCPDTL records from the DCS upload interface table.

2.1.4 Configuring the Purchase Order Pipeline

The Purchase Order time-triggered transactions require a Purchase Order pipeline. If you need additional information about configuring pipelines, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

To configure the Purchase Order pipeline:

1. From the Applications Manager menu, choose Business Process > Process Modeling.
2. Verify that the Purchase Order pipeline is configured with the following transactions:



3. At the bottom of the left pane, click the Services tab to open the Services tree.
4. Create a new service named `YantraWMSPODownloadService` that is invoked synchronously, does not provide real time response, and contains the following sequence of nodes:
   a. Start node
   b. Database node: specify the table name property as YFS_EXPORT
   c. End node
5. Create an action. Click the Invoked Services tab and add the service `YantraWMSPODownloadService` you created in Step 4.
6. Attach this action to the ON_SUCCESS events of the `Create Order` and Change `Order` transactions in the Purchase Order Execution repository. If necessary, add a condition to call this action only if the receiving node is the WMS Node.

2.1.5 DCS Purchase Order Interface

This section provides the lists of header information for purchase order download header, download detail, receipt header, receive order sample output example and input XML mapping with order header, shipment and order line records.

2.1.5.1 POHDR - Purchase Order Download Header

Table 2–2, "POHDR Record Type - Purchase Order Download Header Interface Format" lists the header information required by the Purchase Order Download time-triggered transaction.

Table 2–2 POHDR Record Type - Purchase Order Download Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	OrderLine.ReceivingNode in CreateOrder XML	5	1
record_type	'POHDR'	6	6
action_code	Always 'CH'	2	12
recv_order_type	'VN'	2	14

Table 2–2 POHDR Record Type - Purchase Order Download Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
recv_order_no	Order.OrderNo in CreateOrder XML (Alphabetic characters must be upper-case)	13	16
recv_order_release_no	'1'	3	29
source	Order.SellerOrganizationCode in CreateOrder XML	10	32

2.1.5.2 PODTL - Purchase Order Download Detail

Table 2–3, "PODTL Record Type - Purchase Order Download Detail Interface Format" lists the detail, or line information, required by the Purchase Order Download time-triggered transaction.

Table 2–3 PODTL Record Type - Purchase Order Download Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	OrderLine.ReceivingNode in CreateOrder XML	5	1
record_type	'PODTL'	6	6
action_code	'CL' Only for PO Line Close. (This happens when the line ordered quantity is reduced to zero.) 'CH' for all other modifications, such as changing the quantity (to nonzero), ETA, or adding lines.	2	12
recv_order_type	'VN'	2	14
recv_order_no	Order.OrderNo in CreateOrder XML (Alphabetic characters must be upper-case)	13	16
recv_order_release_no	'1'	3	29
recv_order_line_no	OrderLine.PrimeLineNo in CreateOrder XML	5	32
item_id	OrderLine.Item.ItemID in CreateOrder XML	24	37
product_class	OrderLine.Item.ProductClass in CreateOrder XML	6	61
pack_type	Always blank	4	67
order_qty	OrderLine.OrderedQty in CreateOrder XML	9	71
pre_production	Always blank	1	80

Table 2–3 PODTL Record Type - Purchase Order Download Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
x_doc_recv_order	Always blank	1	81
eta_date	OrderLine.ReqShipDate in CreateOrder XML	8	82
unit_price	OrderLine.LinePriceInfo.UnitPrice in CreateOrder XML	11	90
country_of_origin	OrderLine.Item.CountryOfOrigin in CreateOrder XML	5	101
reference_1	Always blank	20	106
reference_2	Always blank	20	126
reference_3	Always blank	20	146

2.1.5.3 Sample Receive Order Output XML

Example 2–1, "Sample Receive Order Output XML" shows a sample of the XML published by the ON_SUCCESS event of the Receive Order transaction.

Example 2–1 Sample Receive Order Output XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Receipt EnterpriseCode="E1" OrderNo="BB_11" ReceiptNo="AMAR88891">
    <ReceiptLines>
        <ReceiptLine PrimeLineNo="2" Quantity="1.0" ReceiptHeaderKey=""
         SubLineNo="1"/>
    </ReceiptLines>
</Receipt>
```

2.1.5.4 RCPHDR - Purchase Order Receipt Header

Table 2–4, "RCPHDR Record Type - Purchase Order Receipt Header Interface Format" lists the header information required by the Purchase Order Receipt time-triggered transaction.

Table 2–4 RCPHDR Record Type - Purchase Order Receipt Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	OrderLine.ReceivingNode in CreateOrder XML	5	1
record_type	'RCPHDR'	6	6
action_code	Always 'AD'	2	12
asn_no	Advance Shipment Notice number	20	14
asn_type	Advance Shipment Notice type	2	34
reference_1	Reference number	30	191

2.1.5.5 RCPDTL - Purchase Order Receipt Detail

Table 2–5, "RCPDTL Record Type - Purchase Order Receipt Detail Interface Format" lists the detail, or line information, required by the Purchase Order Receipt time-triggered transaction.

Table 2–5 RCPDTL Record Type - Purchase Order Receipt Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	OrderLine.ReceivingNode in CreateOrder XML	5	1
record_type	'RCPDTL'	6	6
action_code	Always 'AD'	2	12
asn_no	Advance Shipment Notice number	20	14
asn_type	Advance Shipment Notice type	2	34
recv_order_no	Order.OrderNo in CreateOrder XML (Alphabetic characters must be upper-case)	13	66
recv_order_line_no	OrderLine.PrimeLineNo in CreateOrder XML	5	82
received_qty	Quantity received in ASN against order line number	7	119

2.1.5.6 Receive Order Input XML Mapping

The `receiveOrder()` API input XML maps to DCS tables at the order header level and at the order line level as described in this section.

Order Header Records

The `receiveOrder()` API input XML and the DCS Order Header map as shown in Table 2–6, "Selling and Fulfillment Foundation and DCS Order Header Mapping".

Table 2–6 Selling and Fulfillment Foundation and DCS Order Header Mapping

Selling and Fulfillment Foundation XML Parameter	DCS Parameter
orderheaderkey	Always blank
orderreleasekey	Always blank
receiptheaderkey	Always blank
receiptno	HEADER.ASN_NO
releaseno	Always blank

Shipment Records

The `receiveOrder()` API input XML and the DCS Order map as shown in Table 2–7, "Selling and Fulfillment Foundation Shipment and DCS Order Mapping".

Table 2–7 Selling and Fulfillment Foundation Shipment and DCS Order Mapping

Selling and Fulfillment Foundation XML Parameter	DCS Parameter
enterprisecode	EnterpriseCode
orderno	DETAIL.RECV_ORDER_NO

Order Line Records

The `receiveOrder()` API input XML and the DCS Order Line map as shown in Table 2–8, "Selling and Fulfillment Foundation and DCS Order Line Mapping".

Table 2–8 Selling and Fulfillment Foundation and DCS Order Line Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter
BreakIntoComponents	Always blank
DispositionCode	Always blank
InspectedBy	Always blank
InspectionComments	Always blank
InspectionDate	Always blank
LotNumber	Always blank
OrderLineKey	Always blank
PrimeLineNo	DETAIL.RECV_ORDER_LINE_NO
SubLineNo	1
Quantity	DETAIL.RECEIVED_QTY
ReceiptLineNo	Always blank
SerialNo	Always blank
ShipByDate	Always blank
<KitLines>	Not used

2.2 DCS Shipment Interface

The DCS integrates with the Sterling Distributed Order Management interface of Selling and Fulfillment Foundation. This integration enables shipment-related information to be passed between applications.

2.2.1 Understanding the Order Transactions

Before implementing the upload and download functionality, you should understand the following default behaviors:

- Modifications to an Order or Order Release in Selling and Fulfillment Foundation after download to DCS are not transmitted to DCS.

- Inventory is reduced from the onhand supply when Selling and Fulfillment Foundation receives and processes the shipment confirmation transaction from DCS. DCS must not be configured to upload separate inventory transactions for shipments.
- The SCAC and Service Code used by the Selling and Fulfillment Foundation input XML corresponds to the SCAC field in the DCS interface. Map each carrier defined in DCS to those in Selling and Fulfillment Foundation by creating an identical configuration in the Applications Manager > Application Platform > Participant Modeling. For example, if DCS uses *UPSG* as the SCAC Code for United Parcel Ground Service, in Selling and Fulfillment Foundation for the participant called *UPS*, set the SCAC and Service Code as UPSG, and specify the *Service* as Ground.
- DCS should disable cancellation from transaction 02012 (Order Release list). Selling and Fulfillment Foundation only recognizes cancellations with return ownership = Y when done from DCS transaction 02013 (load/shipper list).
- The Order No for Shipment Advice can be a maximum length of 13 bytes and must be upper-case characters and numbers or just numbers (lower-case characters are not allowed).

2.2.2 Configuring DCS Shipment Time-Triggered Transactions

Setting up a sales order involves configuring and scheduling the Send Release and WMS Shipment Confirmation time-triggered transactions and configuring the pipeline a sales order should use. You also should check your Oracle database configuration.

To configure the Send Release and WMS Shipment Confirmation time-triggered transactions:

1. Check your Oracle database to ensure that links are created for each DCS ship node for which you create a Release. Selling and Fulfillment Foundation maintains links and views to the DCS interface tables for each node.

2. Configure the Send Release and Ship Confirm time-triggered transactions:

   - If you want to configure the Send Release transaction, from the Applications Manager Applications menu, choose Application Platform > Process Modelling > Order > Sales Order > Order Fulfillment > Transaction Repository Send Release.
   - If you want to configure the Ship Confirm transaction, from the Applications Manager Applications menu, choose Application Platform > Process Modelling > General > General > Transaction Repository > Ship Confirm.

   Note: While the Send Release time-triggered transaction does not require a ship node to be configured for downloading, you do need to configure agent criteria for each ship node from which a WMS shipment confirmation is to be processed.

3. Configure the Sales Order Fulfillment Pipeline to download ship advice to DCS and receive shipment confirmation from DCS.

   The repository has a default pipeline configured to download shipment advice to DCS and receive shipment confirmation. When modifying the pipeline, first copy the default pipeline and then modify that copy to suit your needs. For more information about configuring a pipeline, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

   While configuring the pipeline, keep in mind the following characteristics of the DCS shipment-related integration:

   - Order Releases to be downloaded to DCS are staged with the status `Awaiting WMS Interface` (3200.02). The Send Release transaction in the pipeline is configured to pick up these Order Releases and download them to DCS.
   - After the download completes, the Order Release status moves to `Sent to Node` (3300).
   - The Shipment Confirmation transaction uploads the shipment from the DCS interface table and moves the status of the Order to `Shipped` (3700).

4. Schedule the time intervals for running the Send Release and WMS Shipment Confirmation time-triggered transactions from DCS, as described in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

2.2.3 DCS Order Release Interface

When Order Releases are going to the DCS interface, the Send Release transaction dispatches the Order Release information to DCS in open interface format if the ship node's interface is set to DCS in Selling and Fulfillment Foundation.

To set the ship node as a WMS ship node:

From the Applications Manager Applications menu, choose Application Platform > Participant Modeling > Organization Details > Roles & Participation Tab > Node Attributes/Primary Info Tab (on the right) > Execution In Node Using and choose the WMS 6.2.

This section details only those records and attributes that are supported by Selling and Fulfillment Foundation. These record types are written by the Send Release transaction into the DCS download interface table. Selling and Fulfillment Foundation supports the following record types:

- ORDHDR – Order Release Order Header
- ORDDTL – Order Release Order Detail
- ORDADR – Order Release Order Address
- ORDINS – Order Release Order Instruction
- ORDBOM – Order Release Order Bill of Materials
- ORDNAM – Order Release Order Name

Only the action code Add (AD) is supported by Selling and Fulfillment Foundation.

The following tables list the field information for each record type that the Send Release time-triggered transaction can output.

2.2.3.1 ORDHDR – Order Release Order Header

Table 2–9, "ORDHDR Record Type - Order Header Interface Format" lists the Order Release header information mapped to DCS.

Table 2–9 ORDHDR Record Type - Order Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always ‘ORDHDR’	6	6
action_code	Always ‘AD’	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
ship_to_cust_id	PersonInfoShipTo PersonID in CreateOrder XML	35	30
bill_cust_id	PersonInfoBillTo PersonID in CreateOrder XML	35	65
forward_to_cust_id	PersonInfoMarkFor PersonID in CreateOrder XML	35	100
pack_hold_flag	Always ‘N’	1	135
order_type	OrderType in CreateOrder XML	1	136
order_cancel_date	ReqCancelDate in Order Release being downloaded	8	147
order_due_date	ReqDeliveryDate in CreateOrder XML	8	155
terms_code	TermsCode in CreateOrder XML	8	163
carrier_code	SCAC	4	173
priority_code	PriorityCode in CreateOrder XML	2	177
consol_rule	Always blank	2	179
cartonization_rule	Always blank	2	181
cust_order	CustomerPONo in CreateOrder XML	25	183
pack_list_type	PackList Type in ShipAdvice XML	2	208
spc_ticket_req	PersonalizeCode in CreateOrder XML	2	210
asn_flag	NotifyAfterShipmentFlag in CreateOrder XML	1	212
delivery_date	ReqDeliveryDate in Order Release being downloaded	8	216
orig_ship_date	ReqShipDate in Order Release being downloaded	8	224
samples_flag	Always blank	1	234
ship_to_customer_ name	PersonInfoShipTo FirstName and LastName in CreateOrder XML	35	235

Table 2–9 ORDHDR Record Type - Order Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
ship_to_addr1	PersonInfoShipTo AddressLine1 in CreateOrder XML	35	270
ship_to_addr2	PersonInfoShipTo AddressLine2 in CreateOrder XML	35	305
ship_to_addr3	PersonInfoShipTo AddressLine3 in CreateOrder XML	35	340
ship_to_add4	PersonInfoShipTo AddressLine4 in CreateOrder XML	35	375
ship_to_city	PersonInfoShipTo City in CreateOrder XML	30	410
ship_to_state	PersonInfoShipTo State in CreateOrder XML	2	440
ship_to_zip_code	PersonInfoShipTo Zip Code in CreateOrder XML	9	442
ship_to_country_ code	PersonInfoShipTo Country in CreateOrder XML	5	451
cross_dock_flag	Always blank	2	456
split_flag	ShipCompleteFlag in CreateOrder XML	1	488
consol_flag	Always blank	1	489
shippable_order	Always ‘Y’	1	490
delivery_code	DeliveryCode in CreateOrder XML	1	517
back_order_ authorized_ ind	Always ‘01’	2	526
cal_check_req_ind	Always ‘N’	1	541
inbound_flag	Always ‘N’	1	550
order_create_date	OrderDate in CreateOrder XML	8	564
carrier_service	Carrier Service Code in CreateOrder XML	10	572
cust_carrier_charge_ account_no	Carrier Account Number in CreateOrder XML	35	582
enterprise_code	Enterprise Code	24	639

2.2.3.2 ORDDTL – Order Release Order Detail

Table 2–10, "ORDDTL Record Type - Order Detail Interface Format" lists the Order Release detail information mapped to DCS.

Table 2–10 ORDDTL Record Type - Order Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always ‘ORDDTL’	6	6
action_code	Always ‘AD’	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
order_prime_line	Order Prime Line Number	5	30
order_sub_line	Order Sub Line Number	5	35
mark_for	PersonInfoMarkFor PersonID in CreateOrder XML	35	40
item_id	ItemID in CreateOrder XML	24	75
product_class	ProductClass in CreateOrder XML	6	99
quality_status	Always blank	2	105
department_code	DepartmentCode in CreateOrder XML	6	107
hazard_flag	Always ‘N’	1	119
qty_ordered	OrderedQty in CreateOrder XML	9	120
shippable_qty	Total Quantity to be shipped	9	129
nonshippable_qty	Always ‘0’	9	138
pack_type	Always ‘EACH’	4	147
ship_together_code	ShipTogetherNo in CreateOrder XML	5	151
line_price	Unit Price from LinePriceInfo in CreateOrder XML	11	156
spl_processing_code1	Always blank	4	167
orig_req_ship_date	ReqShipDate in CreateOrder XML	8	249
act_req_ship_date	ReqShipDate in CreateOrder XML	8	257
customer_po_no	CustomerPONo in CreateOrder XML	25	269
ship_sure_model_ind	Always ‘Y’	1	294
order_line_point	Always blank	5	295

Table 2–10 ORDDTL Record Type - Order Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
line_type	Always blank	4	300
carrier_code	Always blank	4	304
samples_flag	Always 'N'	1	308
customer_po_line_no	CustomerLinePONo in CreateOrder XML	13	335
customer_sku	CustomerItem in CreateOrder XML	40	386
kit_code	KitCode in CreateOrder XML	2	466

2.2.3.3 ORDADR – Order Release Order Address

Table 2–11, "ORDADR Record Type - Order Address Interface Format" lists the Order Release address information mapped to DCS.

Table 2–11 ORDADR Record Type - Order Address Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'ORDADR'	6	6
action_code	Always 'AD'	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
address_type	'FT' or 'BT'	2	30
customer_name	FirstName and LastName in CreateOrder XML	35	32
addr1	AddressLine1 in CreateOrder XML	35	67
addr2	AddressLine2 in CreateOrder XML	35	102
addr3	AddressLine3 in CreateOrder XML	35	137
addr4	AddressLine4 in CreateOrder XML	35	172
city	City in CreateOrder XML	30	207
state	State in CreateOrder XML	2	237

Table 2–11 ORDADR Record Type - Order Address Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
zip_code	Zip Code in CreateOrder XML	9	239
country_code	Country in CreateOrder XML	5	248
wms_buffer	Always blank	30	253

2.2.3.4 ORDINS – Order Release Order Instruction

Table 2–12, "ORDINS Record Type - Order Instruction Interface Format" lists the Order Release instruction information mapped to DCS.

Table 2–12 ORDINS Record Type - Order Instruction Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'ORDINS'	6	6
action_code	Always 'AD'	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
order_prime_line	Order Prime Line Number	5	30
order_sub_line	Order Sub Line Number	5	35
instruction_type	InstructionType in CreateOrder XML	3	40
seq_no	Sequence Number of instructions	3	43
usage_type	Instruction usage	2	46
instructions_text	InstructionText in CreateOrder XML	80	48
wms_buffer	Always blank	30	128

2.2.3.5 ORDBOM – Order Release Order Bill of Materials

Table 2–13, "ORDBOM Record Type - Order Bill of Materials Interface Format" lists the Order Release Bill of Materials information mapped to DCS.

Table 2–13 ORDBOM Record Type - Order Bill of Materials Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'ORDBOM'	6	6
action_code	Always 'AD'	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
order_prime_line	Order Prime Line Number	5	30
order_sub_line	Order Sub Line Number	5	35
item_id	ItemID in CreateOrder XML	24	40
product_class	ProductClass in CreateOrder XML	6	64
quality_status	Always blank	2	70
pack_type	Always 'EACH'	4	72
bom_qty	KitQty in CreateOrder XML	9	76
pick_slip_number	Always blank	13	85
picking_line_detail_id	Always blank	13	98
scrap_factor	Always '0000000'	7	111
reference_field1	Always blank	40	118
reference_field2	Always blank	40	158
reference_field3	Always blank	40	198
reference_field4	Always blank	40	238
reference_field5	Always blank	40	278
wms_buffer	Always blank	30	318

2.2.3.6 ORDNAM – Order Release Order Name

This interface format is used to send orders having the following information:

- COD - This record is sent for orders having PaymentType as COD.

- Customer Phone Number - This record is sent only if the ShipTo Customer Day Phone Number is not blank.
- Importer information - This record is sent for international shipments only. This information is not sent if country code in any address (ship node or ship-to address) is blank.
- YFS accepts Import License ID and Import License Expiration Date at Order line level, whereas DCS accepts it at Order header level.
- Exporter Information - This record is sent for international shipments only.

The ship node address country code and ship-to address country code should not be blank.

Table 2–14, "ORDNAM Record Type - Order Name Interface Format" lists the Order Release name information mapped to DCS.

Table 2–14 ORDNAM Record Type - Order Name Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'ORDNAM'	6	6
action_code	Always 'AD'	2	12
order_no	Order Number (Alphabetic characters must be upper-case)	13	14
order_rel_no	Order Release Number	3	27
name	For COD- '100' For Customer Phone Number - '300' For Importer Information - '400'[1] For Exporter Information - '400'	3	30
value	For COD- '103' For Customer Phone Number - '301' For Importer Information - '402'[1] For Exporter Information - '401'	3	33

Table 2–14 ORDNAM Record Type - Order Name Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
reference_field1	For COD - 'COD' For Customer Phone Number - PersonInfoShipTo DayPhone in CreateOrder XML For Importer information - TaxPayerId in CreateOrder XML[1] For Exporter Information - ExportTaxPayerId of the ShipNode	40	36
reference_field2	For COD - Always blank For Customer Phone Number - Always blank For Importer Information - ImportLicenseNo in CreateOrder XML[1] For Exporter Information - ExportLicenseNo of the ShipNode	40	76
reference_field3	For COD - Always blank For Customer Phone Number - Always blank For Importer information - ImportLicenseExpDate in CreateOrder XML[1] For Exporter Information - ExportLicenseExpDate of the ShipNode	40	116
reference_field4	Always blank	40	156
reference_field5	Always blank	40	196
wms_buffer	Always blank	30	236

Note: When Selling and Fulfillment Foundation sends Order Release information to DCS, it sends only the Import License ID and Import License Expiration Date from the first order line and ignores information from the other lines. As a result, if you need to track all license information, group items by license type in separate orders. For example, put all materials that require the same type of license for hazardous material on one order and items that require the same type of license for nonhazardous chemicals on another.

2.2.4 DCS Shipment Confirmation

Selling and Fulfillment Foundation picks up the shipment confirmations posted by DCS in the open interface tables. The WMS Shipment Confirmation time-triggered transaction performs shipment confirmation.

This section details only those records and attributes that are supported by Selling and Fulfillment Foundation. The WMS Shipment Confirmation transaction reads only the following record types from the DCS upload interface table:

- PCKHDR – Shipment Confirmation Pickticket Header
- CARHDR – Shipment Confirmation Carton Header
- PCKINF – Shipment Confirmation Pick Information
- CNCDTL – Shipment Confirmation Cancel Detail
- SRLDTL - Pick Ticket Serial Record

Only action codes Cancel (CA) and Ship (SH) are picked up by Selling and Fulfillment Foundation.

This section describes the interface formats for the different shipment confirmation record types that Selling and Fulfillment Foundation supports.

2.2.4.1 PCKHDR – Shipment Confirmation Pickticket Header

Table 2–15, "PCKHDR Record Type - Pickticket Header Interface Format" lists the shipment confirmation pickticket header information mapped to DCS.

Table 2–15 PCKHDR Record Type - Pickticket Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'PCKHDR'	6	6
action_code	Always 'CA' or 'SH'	2	12
pickticket_no	PickTicketNo in confirmShipment XML	20	14
ship_type	Ship Mode in confirmShipment XML	4	80
actual_ship_date	ShipDate in confirmShipment XML	8	92

Table 2–15 PCKHDR Record Type - Pickticket Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
carrier_code	SCAC and Service Code in confirmShipment XML	4	107
trailer_no	TrailerNo in confirmShipment XML	20	111
freight_charges	FreightCharge in confirmShipment XML	13	131
manifest_no	ManifestNo in confirmShipment XML	20	144
bol_no	BOL Number	20	164
pro_no	ProNo in confirmShipment XML	20	184
master_bol_no	Parent Shipment Key	20	204
total_weight	TotalWeight in confirmShipment XML	13	224
seal_no	Seal Number	20	250
total_volume	TotalVolume in confirmShipment XML	7	296
it_number	IT number	20	303
it_date	IT Date	8	323
from_appointment_date	From appointment date	8	331
to_appointment_date	To appointment date	8	339
appointment_number	Appointment number	40	363
ship_to_addr1	ToAddress AddressLine1 in confirmShipment XML	35	483
ship_to_addr2	ToAddress AddressLine2 in confirmShipment XML	35	518
ship_to_addr3	ToAddress AddressLine3 in confirmShipment XML	35	553
ship_to_addr4	ToAddress AddressLine4 in confirmShipment XML	35	588
ship_to_city	ToAddress City in confirmShipment XML	30	623
ship_to_state	ToAddress State in confirmShipment XML	2	653
ship_to_zip	ToAddress Zip Code in confirmShipment XML	9	655
ship_to_country	ToAddress Country in confirmShipment XML	5	664

2.2.4.2 CARHDR – Shipment Confirmation Carton Header

Table 2–16, "CARHDR Record Type - Carton Header Interface Format" lists the carton header information mapped to DCS.

Table 2–16 CARHDR Record Type - Carton Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'CARHDR'	'6	6
action_code	'CA' or 'SH'	'2	12
pickticket_no	Not used	20	14
carton_no	Container number	11	34
weight	Used for containers other than pallets. Used as Container Gross Weight and Container Net Weight.	13	59
tracking_number	Tracking number	20	72
ucc128_code	If the third character is not '1', this is used as Container SCM.	30	92
manifest_no	Manifest number	10	122
pallet_scm	If the third character of ucc128_code is '1', this is used as Container SCM.	30	132
package_type	Container type	2	162
pallet_length	Used if the container is pallet. Specifies the pallet length.	9	164
pallet_width	Used if the container is pallet. Specifies the pallet length.	9	173
pallet_height	Used if the container is pallet. Specifies the pallet height.	9	182
pallet_gross_weight	Used if the container is pallet. Specifies the pallet gross weight.	9	191
pallet_net_weight	Used if the container is pallet. Specifies the pallet net weight.	9	200
carton_length	Used for containers other than pallet. Specifies the container length.	9	209

Table 2–16 CARHDR Record Type - Carton Header Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
carton_width	Used for containers other than pallet. Specifies the container width.	9	218
carton_height	Used for containers other than pallet. Specifies the container height.	9	227
freight_charge	Freight Charge	7	238

2.2.4.3 PCKINF – Shipment Confirmation Pick Information

Table 2–17, "PCKINF Record Type-Pick Information Interface Format" lists the shipment confirmation pick information mapped to DCS.

Table 2–17 PCKINF Record Type-Pick Information Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'PCKINF'	6	6
action_code	Always 'CA' or 'SH'	2	12
pickticket_no	PickTicketNo in confirmShipment XML	20	14
carton_no	Container No	11	34
item_id	Item ID	24	45
product_class	Product Class	2	69
picked_qty	Shipped Qty	9	80
order_no	Order No	13	89
order_release_no	Release Number	3	102
order_line_no	Prime Line No	4	105
sub_line_no	Sub Line No	5	109

2.2.4.4 CNCDTL – Shipment Confirmation Cancel Detail

Table 2–18, "CNCDTL Record Type-Cancel Detail Interface Format" lists the cancel detail information mapped to DCS.

The CNCDTL record is created only when Orders or Shipments are cancelled or backordered from the DCS Load/Shipper screen (02013), not the Order Release List screen (02012).

Table 2–18 CNCDTL Record Type-Cancel Detail Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'CNCDTL'	6	6
action_code	Always 'CA' or 'SH'	2	12
order_no	Order No	13	34
order_release_no	Order Release No	3	47
order_line_no	Prime Line No	5	50
sub_line_no	Sub Line No	5	55
item_id	Item ID	24	60
product_class	Product Class	2	84
cancel_quantity	BackOrder Qty	9	92

2.2.4.5 SRLDTL - Pick Ticket Serial Record

Selling and Fulfillment Foundation can accept serial numbers when an item has been configured in DCS as Serialized and the Selling and Fulfillment Foundation WMS Ship Confirmation agent is used. When an item is configured as Serialized in DCS and is shipped from DCS, DCS publishes SRLDTL records into the interface tables.

The WMS Ship Confirm Upload agent reads the interface records published by DCS and forms an input XML for the `confirmShipment()` API.

The SRLDTL records published by DCS are across order lines. These records do not contain line information. Selling and Fulfillment Foundation retrieves the serial records corresponding to each shipment line by matching the following attributes from the SRLDTL record with the shipment line, and making a subset of serial records for each shipment line:

- Item ID of SRLDTL with item id of Shipment line,
- Product Class of SRLDTL with product class of Shipment line,
- Pallet SCM of SRLDTL with pallet SCM on the container for the shipment line.
- Carton SCM: Based on setup in DCS, this field can have either carton SCM or container number. If the attribute length is 20, it is mapped to the Carton SCM of the shipment line. Otherwise, it is mapped to the Container Number of the shipment line.

Once a subset of the SRLDTL records is formed, Selling and Fulfillment Foundation adds a ShipmentLine element for each SRLDTL record in the XML and reduces the quantity from the already existing ShipmentLine element.

For example, Not Used if a shipment line has five units and there are five SRLDTL records for each unit, Selling and Fulfillment Foundation adds five ShipmentLine elements to the input XML and reduces the quantity of the original element to zero (0).

Note that the YFS_Container_Details table should have a serial number for each unit shipped.

Table 2–19, "SRLDTL Record Type - PickTicket Serial Record Interface Format" lists the pickticket serial record information mapped to DCS.

Table 2–19 SRLDTL Record Type - PickTicket Serial Record Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode (Not used)	5	1
record_type	SRLDTL	6	6
action_code	Not used	2	12
pickticket_no	Not used	20	14
item_id	Shipment line's item ID	24	34
product_class	Shipment line's product class	2	58
item_pseudo_no	Not used	12	60
item_serial_no	Serial number (Passed to the API)	20	72
component_item_id	Not used	24	92

Table 2–19 SRLDTL Record Type - PickTicket Serial Record Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
component_product_class	Not used	2	116
component_pseudo_no	Not used	12	118
component_serial_no	Not used	20	130
quantity	Quantity	9	150
country_of_origin	Not used	5	159
customer_po_number	Not used	25	164
pallet_scm	CARHDR's pallet SCM.	20	189
carton_scm	If the attribute length is 20, it is mapped to the Carton SCM of the shipment line. Otherwise, it is mapped to the Container Number of the shipment line.	20	209
upc_code	Not used	12	229
upc_case_code_scanned	Not used	14	241
upc_case_code_number_of_boxes	Not used	7	255

2.3 DCS Inventory Interface

The DCS inventory interface can download inventory changes due to Returns in Selling and Fulfillment Foundation to DCS. It can also read the uploads of inventory changes from DCS to Selling and Fulfillment Foundation.

2.3.1 DCS Inventory Upload

The Selling and Fulfillment Foundation inventory upload picks up inventory change information from DCS and uploads the information to Selling and Fulfillment Foundation. The WMS Inventory Upload time-triggered transaction, scheduled through `yfs.wms.inventory`, performs inventory change uploading which is read by the WMS Inventory Upload transaction.

DCS passes only one record type, TRNDTL, for an item and product class combination.

2.3.1.1 TRNDTL – Inventory Change Upload Record

Table 2–20, "TRNDTL Record Type-Inventory Change Upload Interface Format" lists the inventory change upload information mapped to DCS.

Table 2–20 TRNDTL Record Type-Inventory Change Upload Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	ShipNode	5	1
record_type	Always 'TRNDTL'	6	6
action_code	Always 'AD'	2	12
tran_code	ReferenceField4	5	31
tran_reason_code	ReasonCode	4	41
item_id	Item ID	24	45
product_class	Product Class	2	69
pack_type	UOM	4	71
unavailable_quantity	HeldQty	8	77
available_quantity	OnHandQty	8	85
held_quantity	HeldQty	8	93
order_release_no	ReferenceField2	3	203
order_line_no	ReferenceField3	5	206
to_location	ReferenceField5	12	284
reference_field_1	ReferenceField1	30	296
reference_field_2	ReasonText	30	326

Note: Reference Fields 1-5 map to the reference field in the YFS_Inventory_Audit table.

Note: For Work Orders, DCS sets the value of the Reference_Field4 Selling and Fulfillment Foundation Parameter to KITD for use by inventory costing.

For more information about the Inventory Upload transaction, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

2.3.2 DCS Inventory Download

When a Return is recorded in the Selling and Fulfillment Foundation, inventory adjustments may take place depending on the configuration of Status and Supply Type. When inventory adjustments take place on ship nodes specified as InterfaceType DCS, the inventory changes are published to the WMS interface tables, if a service is configured to do so.

Caution: Do not configure multiple Supply Types to be downloaded to DCS. Doing so downloads duplicate records to the INFC_DNLD_TAB_1 interface table.

To configure an inventory download service:

1. From the Applications Manager menu, choose Business Process > Process Modeling. Open the General tab and choose the Details of the General process type.
2. Create a new service that is invoked synchronously, does not provide real time response, and contains the following sequence of nodes:
   a. Start node.
   b. API node. Choose Extended API node and configure it as follows:
      * Specify any name for API name.
      * Specify Class Name as `com.yantra.inv.business.inventory.YFSInventoryDownload`
      * Specify Method Name as `downloadInventory`
   c. End node.

3. Create an action. Choose the Invoked Services tab and add the service you created in Step 2.
4. Enable the INVENTORY_CHANGE event raised by the INVENTORY_CHANGE transaction.
5. Attach the action created in Step 3 to the INVENTORY_CHANGE event of the INVENTORY_CHANGE transaction.
6. If necessary, add a Condition node to call the action only if AdjustmentType is RETURN. The AdjustmentType is RETURN when inventory adjustments take place due to Returns.

> **Note:** Even if a service is configured unconditionally, the ship node must be specified as InterfaceType DCS and AdjustmentType is RETURN in order for the data to be written to the interface tables.

Input XML Format

The following input XML is passed to the service by the event:

```
<?xml version="1.0" encoding="UTF-8"?>
<YantraXML>     <XML AccountNo="" AdjustmentType=" "
        CostCurrency="" EnterpriseCode=" " ItemID=" "
        ItemKey="" Organization=" "
        ProductClass="" Quantity="" ReasonCode="" ReasonText=""
        Reference_1=""
        Reference_2="" Reference_3="" Reference_4=""
        Reference_5="" ShipByDate="" ShipNode=""
        SupplyReference=" " SupplyReferenceType=""
        SupplyType=" " UnitCost="" UnitOfMeasure=" "/> </YantraXML>
```

The `downloadInventory()` method publishes inventory to WMS only if the 'AdjustmentType' in the XML is 'RETURN' and the ship node's interface type is 'WMS_YANTRA'. This method converts the XML into a WMS format string. A record is inserted into the 'Infc_Dnld_Tab_1' table in the WMS database with interface type as 'INVD'.

Adding Location and Reference Fields

The default XML (published by the event) does not contain location. Either Sterling WMS can be configured to have a default location or this XML can be modified (to add the 'WarehouseLocation' attribute) in the

service before passing it to this method. If the XML contains the 'WarehouseLocation' attribute, it is passed to Sterling WMS as the location. Similarly, the 'WMSReferenceField1' and 'WMSReferenceField2' attributes can be added to the XML for Sterling WMS fields 'ReferenceField1' and 'ReferenceField2'.

2.3.2.1 INVCHG - Inventory Change Download Record

Table 2–21, "INVCHG Record Type - Inventory Change Download Interface Format" lists the inventory change download information mapped to DCS.

Table 2–21 INVCHG Record Type - Inventory Change Download Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
whse	/YantraXML/XML/ShipNode	5	1
record_type	INVCHG	6	6
action_code	AD	2	12
tran_date	Transaction date in 'CCYYMMDD' format	8	14
tran_time	Transaction time in 'HHMMSS' format	6	22
tran_seq_no	001	3	28
tran_code	WIMT	5	31
tran_type	Always blank	5	36
tran_reason_code	Always blank	4	41
item_id	/YantraXML/XML/ItemID	40	45
product_class	/YantraXML/XML/ProductClass	6	85
pack_type	EACH	4	91
quality_status	Always blank	2	95
unavailable_quantity	0	7	97
available_quantity	/YantraXML/XML/Quantity	7	104
held_quantity	0	7	111
location	/YantraXML/XML/WarehouseLocation	20	118
user_id	User ID from the context	8	138

Table 2–21 INVCHG Record Type - Inventory Change Download Interface Format

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
reference_field_1	Data maps to /YantraXML/XML/WMSReferenceField1. No data is passed, it maps to /YantraXML/XML/SupplyReference. Note: OrderNo is passed in /YantraXML/XML/SupplyReference by the event.	30	146
reference_field_2	Data maps to /YantraXML/XML/WMSReferenceField2. No data is passed, it maps to /YantraXML/XML/SupplyType.	30	176
wms_buffer	Always blank	30	206

2.4 DCS Returns Interface

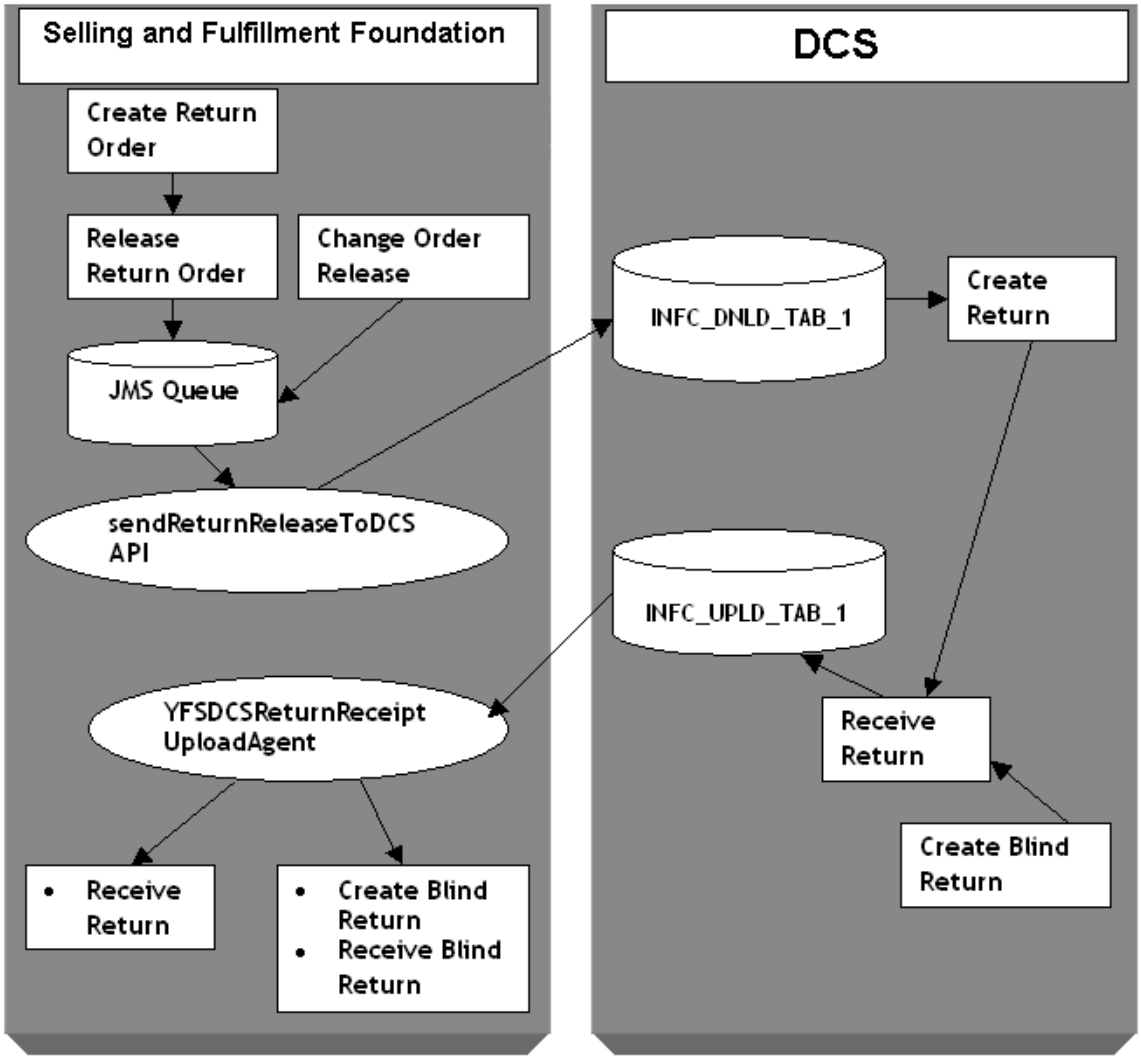
The integration of Selling and Fulfillment Foundation with DCS enables information related to Return Order release to pass between the two applications.

The integration provides an API (sendReturnReleaseToDCS) to send the Return Order release to DCS, and a time triggered transaction (DCS Return Receipt Upload Agent) to get the return release receipt information from DCS. Additionally, this integration supports receipts against blind returns that were created on DCS.

2.4.1 Return Order Integration Workflow

Figure 2–2 illustrates the workflow for the Return Order integration.

Figure 2–2 Workflow for Return Order Integration



1. An external Return Order system invokes the createOrder() API on Selling and Fulfillment Foundation to create a Return Order for a DCS

receiving node. A Return Order is created and the order status becomes Created (1100).

2. When the Return Order is released, the ON_RELEASE_CREATION_OR_CHANGE event of the releaseOrder API can be configured to invoke the service YantraSendReturnReleaseToDCSService, which inserts a message containing the return release key into the JMS Queue. For more information about configuring Return Order integration with DCS, see Section 2.4.3, "Configuring Return Order Integration with DCS".

   The return release is modified by invoking the changeRelease API. After the return release modification, if the ON_SUCCESS event is configured to invoke YantraSendReturnReleaseToDCSService, a message containing the release key is inserted into the JMS queue.

3. The sendReturnReleaseToDCS API picks up the return release key from the JMS Queue, fetches the release details, and inserts a message containing the release details into the DCS interface table INFC_DNLD_TAB_1.

4. An agent on DCS picks up the return release data from INFC_DNLD_TAB_1 and creates a return on DCS.

5. Alternatively, a blind return can be directly created on DCS using the DCS user interface.

6. Once the return is received, DCS agents insert the receipt details into the interface table INFC_UPLD_TAB_1.

7. The DCS Return Receipt Upload Agent picks up the receipt details from the interface table INFC_UPLD_TAB_1 and calls the receiveOrder API to mark the Return Order as received.

   For blind returns, before calling the receiveOrder API, the DCS Return Receipt Upload Agent first calls the createOrder API to create a Return Order, or the changeOrder API to change the order that already exists for this blind return on Selling and Fulfillment Foundation.

2.4.2 Determining the Enterprise Code for Blind Return during Upload

For blind RMA the system determines the enterprise code as follows:

1. If the value of RARHDR.REFERENCE-1 is blank, the primary organization of the owner of the ship node is taken as the enterprise code.
2. If the value of RARHDR.REFERENCE-1 is not blank, the system checks the value of RARHDR.REFERENCE-1.
   - If the value of RARHDR.REFERENCE-1 is a valid organization with an Enterprise role, the system uses the value of RARHDR.REFERENCE-1 as the enterprise Code.
   - If the value of RARHDR.REFERENCE-1 is not a valid organization with an Enterprise role, the system throws an error.

2.4.3 Configuring Return Order Integration with DCS

This section describes the various configurations for Return Order Integration with DCS.

Note1: The setup for the Disposition Code should be identical in both Selling and Fulfillment Foundation and DCS.

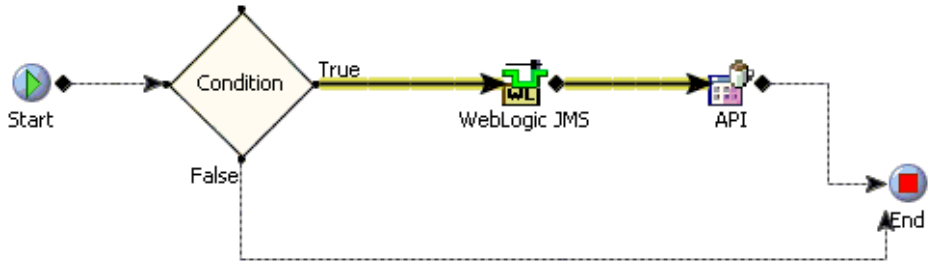
Note2: Inventory updates during return receipt upload should be turned off. Inventory adjustments for return receipts should be done through the inventory adjustment interface with DCS. Whenever inventory is updated in DCS, inventory is updated in Selling and Fulfillment Foundation too through this interface.

2.4.3.1 Configuring return release download to DCS

Configuring return release download to DCS involves creating a new JMS Queue, service, and action.

To configure return release download to DCS:

1. Create a synchronous service, say `YantraSendReturnReleaseToDCSService` under Reverse Logistics Services. This service puts the Return Order release key into a JMS queue, say `RMADownloadQueue`, if the ship node is a DCS node.



Note: The "Condition" mentioned in the figure should be configured with ship node interface type = 'WMS_YANTRA'.

For the API component in the service,

- Choose the General tab.
- Select the Selling and Fulfillment Foundation Standard API option button.
- From the API Name drop-down list, select sendReturnReleaseToDCS.

When the integration server configured in the JMS receiver runs, the `sendReturnReleaseToDCS` API picks up the order release key from the JMS queue and inserts the return release details in the DCS interface table.

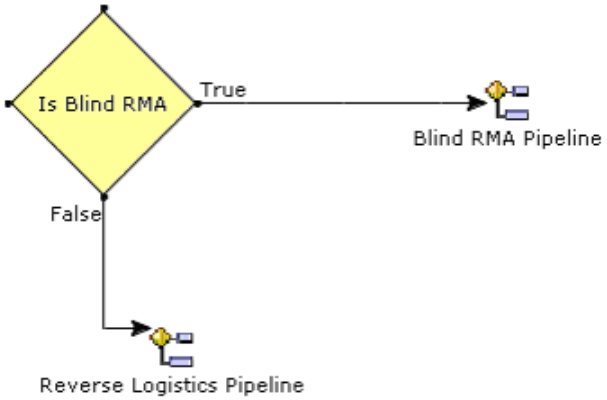
2. Navigate to ReverseLogistics Repository > Actions and create an action, say `SendReturnReleaseToDCS`. This action should invoke `YantraSendReturnReleaseToDCSService`. Configure ON_RELEASE_CREATION_OR_CHANGE event of the SCHEDULE RETURN transaction and ON_SUCCESS event of the changeRelease API to invoke this action (in the case of Reverse Logistics, the SCHEDULE RETURN transaction also does the release).

2.4.3.2 Configuration for Receiving Blind RMA

Return Receipts for Blind RMAs created at the warehouse and the receipt details are uploaded as regular return receipts. The receipt upload agent

creates the Return Order with a ‘03’ order type in Selling and Fulfillment Foundation.

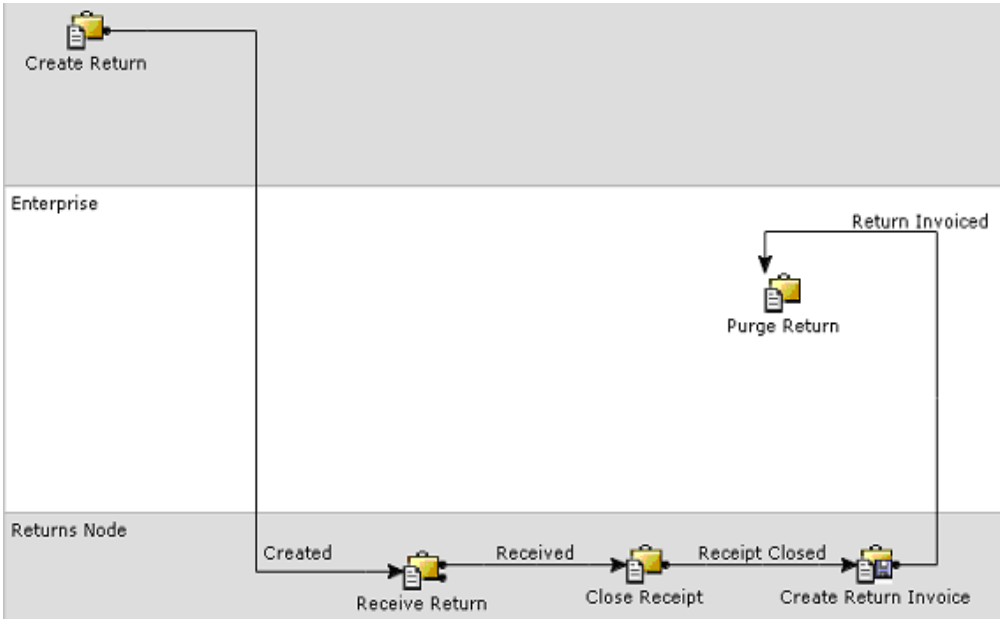
Based on the pipeline determination condition given below, the Blind RMA Pipeline is used for Blind RMA Return Order fulfillment.



Note: The condition "Is Blind RMA" mentioned in the figure is configured as OrderType=‘03’.

Return releases are not created for these return orders. However, a receipt is recorded against the Return Order.

The Blind RMA Pipeline should be configured according to the following pipeline:



2.4.4 Return Order Interface Data Mapping

This section describes the Return Order Interface Data Mapping.

2.4.4.1 Return Order Release Download Data Mapping

The Return Order Release Download Data Mapping are listed in this section.

2.4.4.1.1 RMAHDR - Return Release Download Header

Table 2–22, "RMAHDR Record Type - Return Release Download Header Interface Mapping" lists the header information required by the Return Release Download API.

Table 2–22 RMAHDR Record Type - Return Release Download Header Interface Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
WHSE	OrderRelease/@ReceivingNode	5	1
RECORD-TYPE	“RMAHDR”	6	6
ACTION-CODE	“CH” or “CL” based on modification or closure	2	12
RMA-NUMBER	OrderRelease/Order/@OrderNo	15	14
RMA-RELEASE-NO	OrderRelease/@ReleaseNo	3	29
RMA-TYPE	OrderRelease/Order/@OrderType	2	32
EXPECTED-NO-OF-CASES	N/A	5	34
EXPECTED-NUMBER-OF-PALLETS	N/A	5	39
EXPECTED-NUMBER-OF-UNITS	N/A	7	44
TRAILER-NO	N/A	20	51
FREIGHT-COLLECT-FLAG	OrderRelease/Order/@TermsCode	1	71
EXPECTED-DATE	OrderRelease/Order/@OrderDate	8	72
CARRIER-CODE	OrderRelease/Order/@SCAC	4	80
INVOICE-NUMBER	N/A	20	84
SHIP-TO-CUST-ID	OrderRelease/OrderLine/@ShipToID	10	104
BILL-TO-CUST-ID	OrderRelease/Order/@BillToID	10	114
ENTRY-DATE	OrderRelease/Order/@OrderDate	8	124
SHIP-TO-NAME	OrderRelease/PersonInfoShipTo/ @FirstName + @LastName	25	132
BILL-TO-SHORT-NAME	OrderRelease/Order/PersonInfoBillTo/@FirstName + @LastName	12	157
SHIP-TO-ADDR-1	OrderRelease/PersonInfoShipTo/ @AddressLine1	30	169
SHIP-TO-ADDR-2	OrderRelease/PersonInfoShipTo/ @AddressLine2	30	199

Table 2–22 RMAHDR Record Type - Return Release Download Header Interface Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
SHIP-TO-ADDR-3	OrderRelease/PersonInfoShipTo/ @AddressLine3	30	229
SHIP-TO-CITY	OrderRelease/PersonInfoShipTo/ @City	30	259
SHIP-TO-STATE-CODE	OrderRelease/PersonInfoShipTo/ @State	2	289
SHIP-TO-ZIP	OrderRelease/PersonInfoShipTo/ @ZipCode	9	291
SHIP-TO-COUNTRY-CODE	OrderRelease/PersonInfoShipTo/ @Country	5	300
CLAIM-NO	N/A	20	305
PICKTICKET-NO	N/A	20	325
REASON-CODE	N/A	4	345
PRO-NUMBER	N/A	20	349
REFERENCE-FIELD-1	OrderRelease/Order/ @EnterpriseCode	20	369
REFERENCE-FIELD-2	N/A	20	389
REFERENCE-FIELD-3	N/A	20	409
REFERENCE-FIELD-4	N/A	20	429
REFERENCE-FIELD-5	N/A	20	449
REFERENCE-FIELD-6	N/A	20	469
REFERENCE-FLAG-1	N/A	1	489
REFERENCE-FLAG-2	N/A	1	490
REFERENCE-FLAG-3	N/A	1	491
REFERENCE-FLAG-4	N/A	1	492

Table 2–22 RMAHDR Record Type - Return Release Download Header Interface Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
REFERENCE-FLAG-5	N/A	1	493
REFERENCE-FLAG-6	N/A	1	494
WMS-BUFFER	Defaulted with blank spaces	30	495

2.4.4.1.2 RMADTL - Return Release Download Detail

Table 2–23, "RMADTL Record Type - Return Release Download Detail Interface Mapping" lists the detail or line information required by the Return Release Download API.

Table 2–23 RMADTL Record Type - Return Release Download Detail Interface Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
WHSE	OrderRelease/@Receiving Node	5	1
RECORD-TYPE	“RMADTL”	6	6
ACTION-CODE	“CH” or “CL” based on modification or closure	2	12
RMA-NUMBER	OrderRelease/Order/@OrderNo	15	14
RMA-RELEASE-NO	OrderRelease/@ReleaseNo	3	29
RMA-LINE-NO	OrderRelease/OrderLine/@PrimeLineNo	5	32
RMA-SUB-NO	Default value ‘0’	5	37
ITEM-ID	OrderRelease/OrderLine/Item/@ItemID	24	42
PRODUCT-CLASS	OrderRelease/OrderLine/Item/@ProductClass	2	66
QUALITY-STATUS	Defaulted in INTERFACE_DEFAULTS	2	68
PACK-TYPE	Defaulted in INTERFACE_DEFAULTS	4	70
EXPECTED-QUANTITY	OrderRelease/OrderLine/OrderStatuses/OrderStatus/@ StatusQuantity	9	74
RMA-REASON-CODE	OrderRelease/OrderLine/@ReturnReason	4	83
DISPOSITION-CODE	N/A	2	87
CREDIT-FLAG	OrderRelease/Order/@TermsCode	1	89

Table 2–23 RMADTL Record Type - Return Release Download Detail Interface Mapping

DCS Parameter	Selling and Fulfillment Foundation Parameter	Field Size	Start Position
PSEUDO-SERIAL-NUMBER	N/A	20	90
INVOICE-NUMBER	N/A	20	110
PICKTICKET-NO	N/A	20	130

2.4.4.1.3 RMACMT- Return Release Download Comments

Table 2–24, "RMACMT Record Type - Return Release Download Comments Interface Mapping" lists the comment information required by the Return Release Download API.

Table 2–24 RMACMT Record Type - Return Release Download Comments Interface Mapping

DCS Parameter	Release 5.0 Parameter	Field Size	Start Position
WHSE	OrderRelease/@Receiving Node	5	1
RECORD-TYPE	“RMACMT”	6	6
ACTION-CODE	“CH” or “CL” based on modification or closure	2	12
RMA-NUMBER	OrderRelease/Order/@OrderNo	15	14
RMA-RELEASE-NO	OrderRelease/@ReleaseNo	3	29
RMA-LINE-NO	OrderRelease/OrderLine/@PrimeLineNo or ‘0’ for header level comment	5	32
RMA-SUB-NO	Default value ‘0’	5	37
COMMENT-SEQ-NO	OrderRelease/Order/Instructions/ Instruction/@SequenceNo OR OrderRelease/Orderline/Instructions/ Instruction/@SequenceNo	5	42
COMMENT-TYPE	Maps to appropriate DCS Comment Type	2	47
COMMENT-TEXT	OrderLine-> InstructionText OR Order -> InstructionText– ‘0’ as return line number	80	49

2.4.4.2 Return Receipt Upload Data Mapping

The Return Receipt Upload Data Mapping are listed in this section.

2.4.4.2.1 Data mapping to create Return Order for blind return

Table 2–25, "Return Receipt Upload Interface mapping for input XML to createOrder API for blind return" lists the interface attribute mapping to create return orders if they do not exist in Selling and Fulfillment Foundation.

Table 2–25 Return Receipt Upload Interface mapping for input XML to createOrder API for blind return

Selling and Fulfillment Foundation	DCS
Order/DocumentType	Default value for Return Document ‘0003’
Order/OrderDate	RARHDR.RECEIVED-DATE
Order/OrderNo	RARHDR.RMA_NUMBER
Order/OrderType	Default Value ‘03’
Order/SCAC	RARHDR. CARRIER-CODE
Order/TermsCode	RARHDR. FREIGHT-COLLECT
Order/PersonInfoShipTo/ FirstName	RARHDR. CUSTOMER-NAME
Order/PersonInfoShipTo/ AddressLine1	RARHDR. ADDRESS-1
Order/PersonInfoShipTo/ AddressLine2	RARHDR. ADDRESS-2
Order/PersonInfoShipTo/ AddressLine3	RARHDR. ADDRESS-3
Order/PersonInfoShipTo/ City	RARHDR. CITY
Order/PersonInfoShipTo/ State	RARHDR. STATE

Table 2–25 Return Receipt Upload Interface mapping for input XML to createOrder API for blind return

Selling and Fulfillment Foundation	DCS
Order/PersonInfoShipTo/ ZipCode	RARHDR. ZIP
Order/PersonInfoShipTo/ Country	RARHDR. COUNTRY-CODE
Order/PersonInfoBillTo/ FirstName	RARHDR. CUSTOMER-NAME
Order/PersonInfoBillTo/ AddressLine1	RARHDR. ADDRESS-1
Order/PersonInfoBillTo/ AddressLine2	RARHDR. ADDRESS-2
Order/PersonInfoBillTo/ AddressLine3	RARHDR. ADDRESS-3
Order/PersonInfoBillTo/ City	RARHDR. CITY
Order/PersonInfoBillTo/ State	RARHDR. STATE
Order/PersonInfoBillTo/ ZipCode	RARHDR. ZIP
Order/PersonInfoBillTo/ Country	RARHDR. COUNTRY-CODE
Order/EnterpriseCode	RARHDR.REFERENCE-1, if the value of RARHDR.REFERENCE-1 is a valid Organization with Enterprise role. If the value of RARHDR.REFERENCE-1 is blank, this becomes the primary enterprise of the receiving node's organization. If the value of RARHDR.REFERENCE-1 is not a valid enterprise code, the system throws an error.
OrderLine/ReceivingNode	RARHDR.WHSE

Table 2–25 Return Receipt Upload Interface mapping for input XML to createOrder API for blind return

Selling and Fulfillment Foundation	DCS
Order/Instructions/ InstructionText	RARCMT.COMMENT-TEXT (if RARCMT.RMA-LINE-NO=0)
Order/Instructions/ InstructionType	RARCMT.COMMENT-TYPE (if RARCMT.RMA-LINE-NO=0)
Order/Instructions/ SequenceNo	RARCMT.SEQ_NUMBER (if RARCMT.RMA-LINE-NO=0)
OrderLine/PrimeLineNo	RARDTL.RMA-LINE-NO
OrderLine/OrderedQuantity	RARDTL.QUANTITY
OrderLine/Item/ItemID	RARDTL.ITEM_ID
OrderLine/Item/ProductClass	RARDTL.PRODUCT_CLASS
OrderLine/SubLineNo	Default Value ‘0’
OrderLine/Item/UnitofMeasure	Default Value ‘EACH’
OrderLine/ReturnReason	RARDTL.RMA-REASON-CODE
OrderLine/Instructions/Instruction /InstructionText	RARCMT.COMMENT-TEXT
OrderLine/Instructions/Instruction /InstructionType	RARCMT.COMMENT-TEXT
OrderLine/Instructions/Instruction /SequenceNo	RARCMT.SEQ-NUMBER

2.4.4.2.2 Data mapping to record return receipts

Table 2–26, "Return Receipt Upload Interface mapping for input XML to receiveOrder API for return receipt" lists the interface attribute mapping to record return receipts on Selling and Fulfillment Foundation.

Table 2–26 Return Receipt Upload Interface mapping for input XML to receiveOrder API for return receipt

Selling and Fulfillment Foundation	DCS
Receipt/ReceiptNo	RARHDR.WORKSHEET-NO
Receipt/EnterpriseCode	RARHDR.REFERENCE-1 if the receipt is not against a blind RMA. Otherwise the enterprise code is same as that of the blind RMA.
Receipt/ReleaseNo	RARHDR.RMA-RELEASE-NO
ReceiptLine/InspectedBy	RARDTL.USERID
ReceiptLine/InspectionComments	RARDTL.RMA-REASON-CODE
ReceiptLine/DispositionCode	RARDTL.DISPOSITION-CODE
ReceiptLine/InspectionDate	RARHDR.RECEIVED-DATE
Receipt/OrderNo	RARDTL.RMA-NUMBER
ReceiptLine/PrimeLineNo	RARDTL.RMA-LINE-NO
ReceiptLine/Quantity	RARDTL.QUANTITY
ReceiptLine/SerialNo	RARDTL.SERIAL-NO
ReceiptLine/SubLineNo	Default Value ‘1’

2.4.5 Assumptions and Limitations

The assumptions and limitations in the integration of Selling and Fulfillment Foundation with DCS for returns interface are listed below:

- The integration to DCS is at return release rather than return creation. This is done to support returns that may require a manual credit check or approval before it is accepted (released).

  To send a return order to DCS whenever a return is created, you can model a service to call Return Release upon creation, based on a return type.

- The Return Order number in Selling and Fulfillment Foundation is unique across all enterprises.

- All Return Order lines must use consecutive prime line numbers, with all sub line numbers as ‘0’. The RMADTL record always sets the RMA_SUB_NO as ‘0’.
- Only one release is supported for each receiving node of the Return Order. To apply this, enable the document type level rule ‘Consolidate New Releases’ for the ‘Reverse Logistics’ document type. This allows the new lines added to the Return Order to be included in the existing release.
- Receipt is allowed only for items included in the Return Order. To receive an item that is not in the return, a line with that item should be added into the return release and downloaded into DCS again.
- Inventory updates during return receipt upload should be turned off. Inventory adjustments for return receipts should be done through the inventory adjustment interface with DCS. Whenever inventory is updated in DCS, the inventory is updated in Selling and Fulfillment Foundation too through this interface.
- The following modifications are allowed on a Return Order:
  - Order Level
    - ADD_LINE: A new line can be added to the Return Order. This line is added in the created status. Based on the ‘Consolidate New Releases’ setting in the ‘Reverse Logistics’ document type level, this new line is added into the existing release during the release process and the entire release is downloaded to DCS.
  - Order Line Level
    - Modifications are not allowed in the Return Order line level.
  - Order Release Level
    - ADD_LINE: A new release line can be added.
    - CANCEL: Sterling Commerce recommends that you disallow cancellation once the return release is sent to DCS. This is because the return receipt upload agent throws an exception if the return is being received or has already been received in DCS while it is getting cancelled on Selling and Fulfillment Foundation.
    - ADD_QUANTITY: A release line quantity can be added.

  - The other modifications allowed are Add Note, Change BillTo, Change Carrier, Change Carrier Account No, Change Carrier Service Code, Change Freight Terms, Change Delivery Code, Change MarkFor, Change ReqShipDate, and Change ShipTo.

- Receipt overage is not allowed in DCS. A new return line must be created on Selling and Fulfillment Foundation and downloaded to DCS upon release.
- Return Orders with Kit items should be created as blind returns on DCS. They cannot be created for sales orders in Selling and Fulfillment Foundation.
- Return Orders with Kit items should contain return lines for kit components.
- Return Orders can be created for multiple sales orders and can be received in DCS.
- The return receipt upload agent does not upload instructions to Selling and Fulfillment Foundation if the instruction text is blank.
- The configuration assumptions for DCS are:
  - The creation of a return in DCS is enabled only for blind returns.
  - For blind returns on DCS, a new function should be configured to "Create Blind RMA" with RMA_Type='03' defaulted and protected.This ensures that blind RMAs are always created with RMA_Type = '03'.
  - The ability to receive an overage item or a different item on a return is disabled.

For more information about configuring DCS Inventory updates, see the *Yantra 5x Configuration Guide*.

3

Integrating with Stand-Alone Sterling WMS

Note: The services described in this chapter are not supported in a multischema environment.

3.1 Installing Integration Pack for Receipt and Inventory Change Upload Interfaces on a WMS Instance

To install the receipt and inventory change upload components on the WMS instance:

1. Set the environment variable `INSTALL_DIR` to point to the Selling and Fulfillment Foundation installation directory.
2. Change the directory to `<INSTALL_DIR>/bin`, and run the following command for UNIX or Linux:

   `sci_ant.sh -f wms_integration_pack_installer.xml` (or `sci_ant.cmd` for Windows).
3. After you run the above command, check the contents of the `wms_integration_pack_fc_installer.xml.restart` file located in the `<INSTALL_DIR>/database/FactorySetup/install/` directory. In the `wms_integration_pack_fc_installer.xml.restart` file make sure that the "Completed" attribute of the TaskInfo element is set to "Y". If this is set to "N", fix the integration pack installation problems, and repeat Step 2.

3.2 Installing Integration Pack for Receipt and Inventory Change Upload Interfaces on a DOM Instance

To install the receipt and inventory change upload components on the DOM instance:

Note: If your DOM instance is on a release that is prior to Release 8.5, you must copy the following files located in the runtime directory of the WMS instance to the runtime directory of the DOM instance.

- <INSTALL_DIR>/bin/omp_integration_pack_installer.xml
- <INSTALL_DIR>/database/FactorySetup/install/omp_integration_pack_fc_installer.xml
- <INSTALL_DIR>/database/FactorySetup/IntegrationPack/IP_OMP_*.xml

1. Set the environment variable INSTALL_DIR to point to the Selling and Fulfillment Foundation installation directory.
2. Change the directory to <INSTALL_DIR>/bin, and run the following command for UNIX or Linux:

   sci_ant.sh -f omp_integration_pack_installer.xml (or sci_ant.cmd for Windows).
3. After you run the above command, check the contents of the omp_integration_pack_fc_installer.xml.restart file located in the <INSTALL_DIR>/database/FactorySetup/install/ directory. In the omp_integration_pack_fc_installer.xml.restart file make sure that the "Completed" attribute of the TaskInfo element is set to "Y". If this is set to "N", fix the integration pack installation problems, and repeat Step 2.

3.3 Uploading Receipts

Selling and Fulfillment Foundation supports integration between DOM and WMS for uploading receipts and receipt adjustments. To integrate DOM and WMS, you must configure a common JMS queue. You must also model the node on both instances. For the DOM instance, model the node as a non-WMS integrated node.

Uploading receipt has the following integration touch points:

- WMS Components
  - Uploading the Receipt Information
  - Uploading the Receipt Adjustment Information
- DOM Components
  - Loading the Receipt Information from a Node
  - Loading the Receipt Adjustment Information from a Node

3.3.1 Uploading the Receipt Information

To upload the receipt details from WMS to DOM, use the ReceiptUpload-751 service.

3.3.1.1 The ReceiptUpload-751 Service

This service is invoked from the WMS instance.

The receiveOrder API is invoked during the receiving process. When the receiving process for a case or pallet is complete, and the user closes the case or pallet, or when receiving for a loose SKU is complete, one of the ON_CASE_RECEIPT, ON_PALLET_RECEIPT, and ON_SKU_RECEIPT events of the RECEIVE_RECEIPT transaction is raised. To invoke the ReceiptiUpload-751 service, ensure that the UploadReceipt action, under Order>PO Order Receipt>Actions>Receipt Upload, is configured on these events.

The ReceiptUpload-751 service then translates the API output and serves as an input to the receiveOrder API. This is published as a message to the JMS queue of the web server of the DOM instance.

This service invokes the getReceiptLinesList API. The getReceiptLinesList API has been modified to use an additional flag called RelevantItemLinesOnly. If this flag is set to "Y", the API returns the

relevant lines exploding the hierarchical information of LPNs as necessary, satisfying the input criteria.

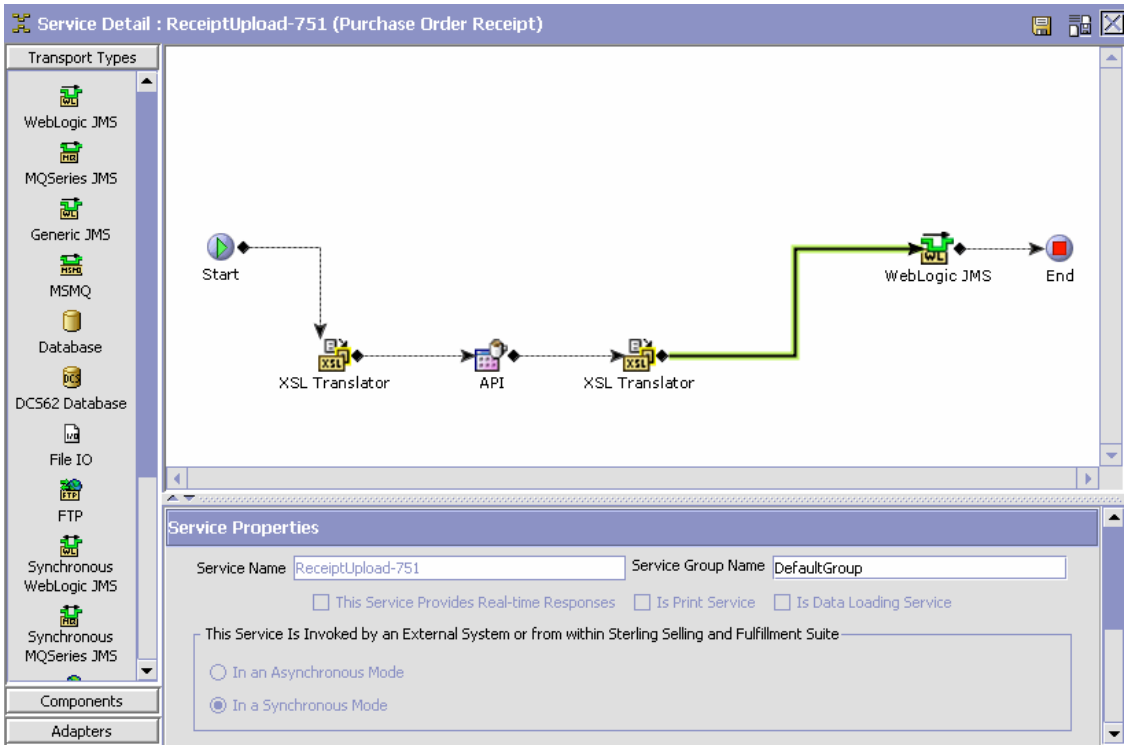
This flag is relevant if:

- Either the case identifier or pallet identifier is passed as input.
- The case identifier or pallet identifier passed is not shipped out of the warehouse.

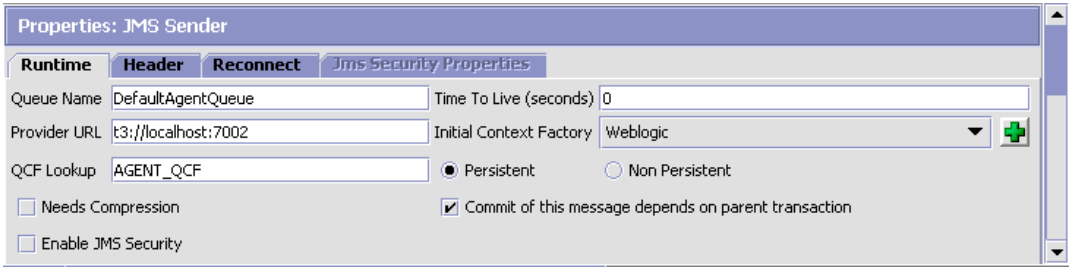
3.3.1.2 Configuring the ReceiptUpload-751 Service

To configure the ReceiptUpload-751 service:

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the Order tab. In the Process Types swimlane, right-click the Purchase Order Receipt process type and click Model Process. The Repository Details window and work area are displayed for the Order process type.
4. Click the Service Definitions tab.
5. Expand the DefaultGroup branch.
6. Right-click ReceiptUpload-751 and select Details. The Service Detail window appears in the work area.



7. Click the green connector that connects the XSL Translator and the WebLogic JMS. The JMS Sender properties displays as shown.



8. In the Runtime tab, make sure that the "Commit of this message depends on parent transaction" box is checked.

For field value descriptions, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

3.3.2 Uploading the Receipt Adjustment Information

To upload receipt adjustment details from WMS to DOM, use the AdjustReceiptUpload-751 service.

3.3.2.1 The AdjustReceiptUpload-751 Service

This service is invoked from the WMS instance.

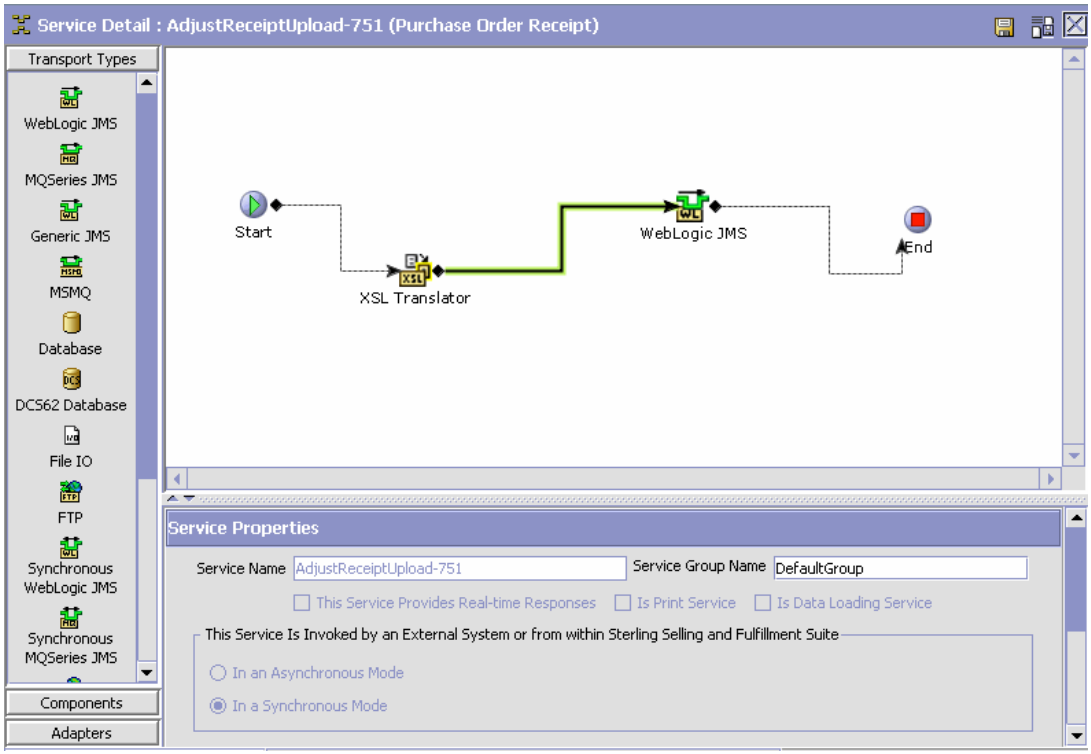
The unreceiveOrder API is invoked during the unreceiving process. When the unreceiving process is complete, the ON_SUCCESS event of the UNRECEIVE_RECEIPT transaction is raised. To invoke the AdjustReceiptUpload-751 service, ensure that the adjustReceiptUpload action, under Order>PO Order Receipt>Actions>Receipt Upload, is configured on the ON_SUCCESS event.

The AdjustReceiptUpload-751 service then translates the API output and serves as an input to the unreceiveOrder AP1. This is published as a message in the JMS queue of the web server of the DOM instance.

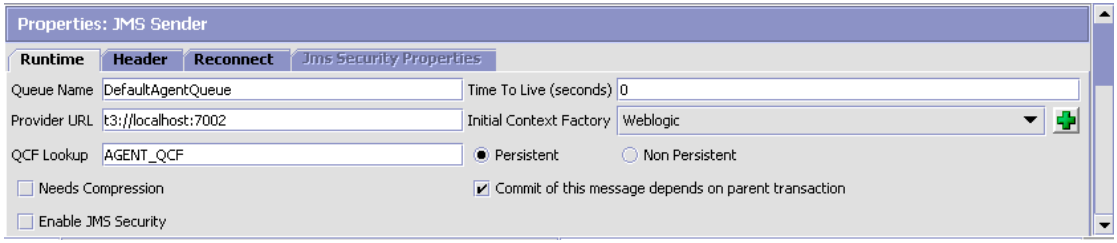
3.3.2.2 Configuring the Updated Receipt Adjustment Information from a Node

To configure the AdjustReceiptUpload-751 service:

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the Order tab. In the Process Types swimlane, right-click the Purchase Order Receipt process type and click Model Process. The Repository Details window and work area are displayed for the Order process type.
4. Click the Service Definitions tab.
5. Expand the DefaultGroup branch.
6. Right-click AdjustReceiptUpload-751 and select Details. The Service Detail window appears in the work area.



7. Click the green connector that connects the XSL Translator and the WebLogic JMS. The JMS Sender properties displays as shown.



8. In the Runtime tab, make sure that the "Commit of this message depends on parent transaction" box is checked.

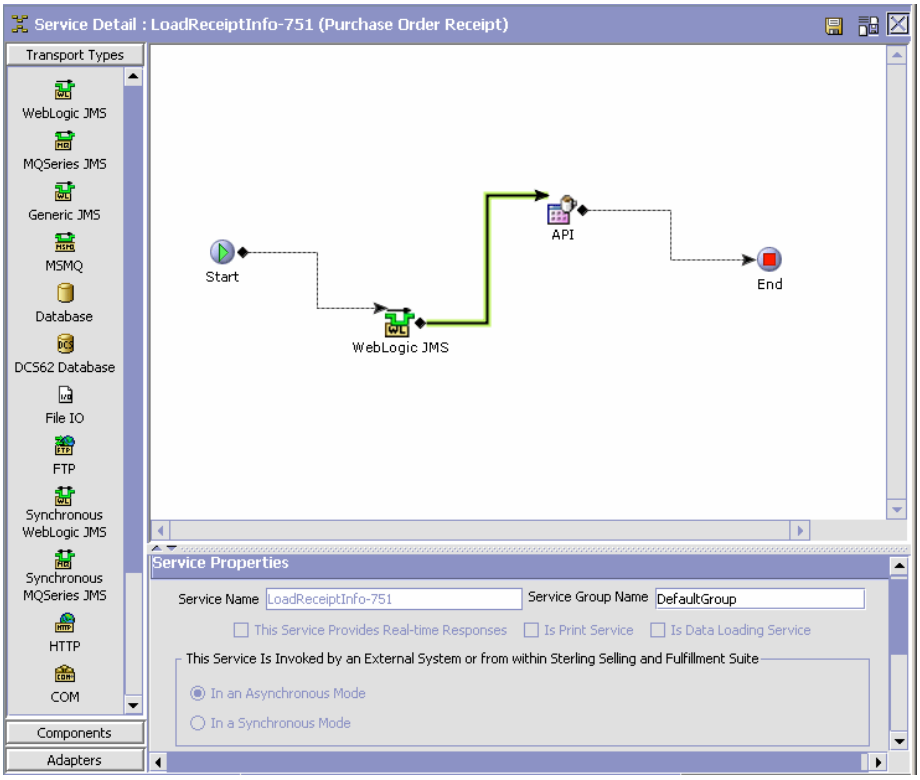
For field value descriptions, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

3.3.3 Loading the Receipt Information from a Node

The LoadReceiptInfo-751 service is used at the DOM instance to retrieve receipt details from the node.

To retrieve receipt details, set up the LoadReceiptInfo-751 service for DOM instance.

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the Order tab. In the Process Types swimlane, right-click the Purchase Order Receipt process type and click Model Process. The Repository Details window and work area are displayed for the Order process type.
4. Click the Service Definitions tab.
5. Expand the DefaultGroup branch.
6. Right-click LoadReceiptInfo-751 and select Details. The Service Detail window appears in the work area.



3.3.3.1 The LoadReceiptInfo-751 service

This service is invoked from the DOM instance.

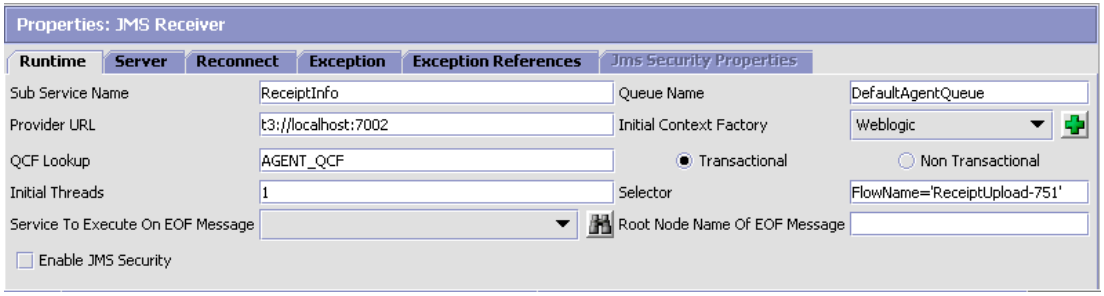
> **Note:** Although we have used Weblogic JMS as an example, the Selling and Fulfillment Foundation also supports the use of IBM WebSphere and JBoss Messaging JMS.

From WebLogic JMS to API

The LoadReceiptInfo-751 service reads the message from the JMS queue and invokes the receiveOrder API.

To configure the service:

1. In the Service Detail: LoadReceiptInfo-751 window, click the green connector that connects the WebLogic JMS and the API. The JMS Receiver properties displays as shown.



For field value descriptions of the fields, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

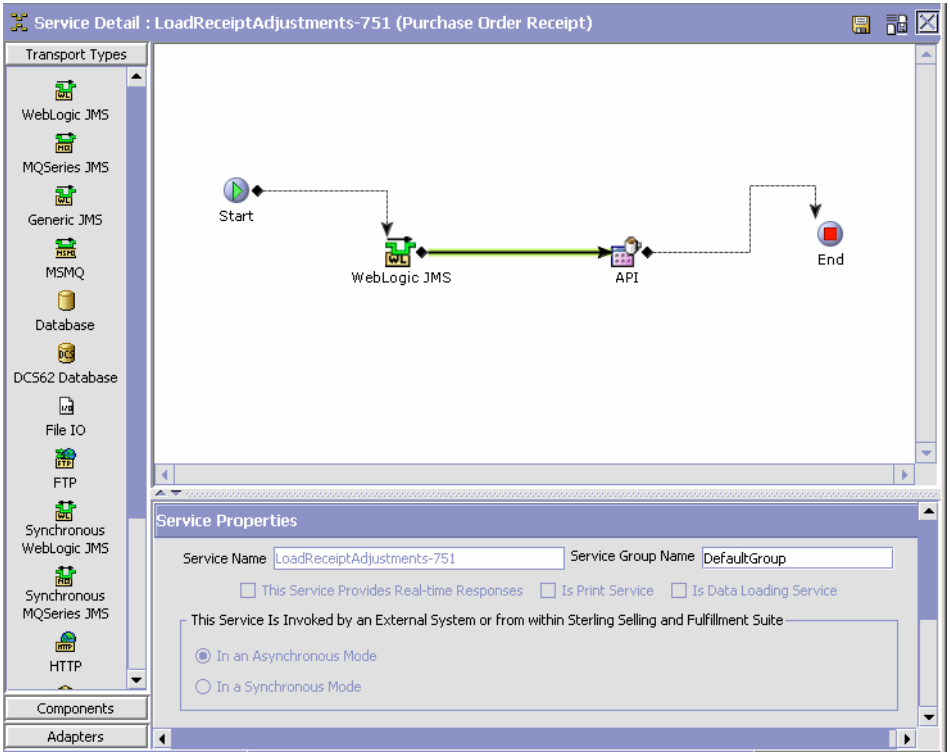
3.3.4 Loading the Receipt Adjustment Information from a Node

The LoadReceiptAdjustments-751 service is used at the DOM instance to retrieve receipt details from the node.

To retrieve receipt details, set up the LoadReceiptAdjustments-751 service for the DOM instance.

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the Order tab. In the Process Types swimlane, right-click the Purchase Order Receipt process type and click Model Process. The Repository Details window and work area are displayed for the Order process type.
4. Click the Service Definitions tab.
5. Expand the DefaultGroup branch.

6. Right-click LoadReceiptAdjustments-751 and select Details. The Service Detail window appears in the work area.



3.3.4.1 The LoadReceiptAdjustments-751 service

This service is invoked from the DOM instance.

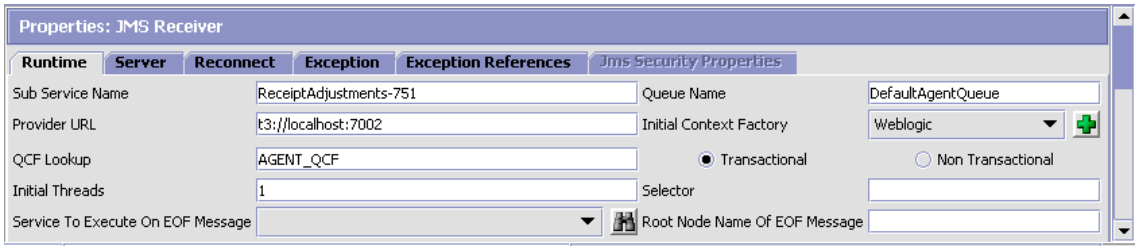
> **Note:** Although we have used WebLogic JMS as an example, Selling and Fulfillment Foundation supports the use of IBM WebSphere and JBoss Messaging JMS.

From WebLogic JMS to API

The LoadReceiptAdjustments-751 service reads the message from the JMS queue and invokes the unreceiveOrder API.

To configure the service:

1. In the Service Detail: LoadReceiptAdjustments-751 window, click the green connector that connects the WebLogic JMS and the API. The JMS Receiver properties displays as shown.



For field value descriptions, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

3.4 Uploading Inventory Changes at a Node

Selling and Fulfillment Foundation provides inventory integration between DOM and WMS that are running on two different instances. To synchronize inventory between separate DOM and WMS instances, you must configure a common JMS queue. You must also model the node on both instances. For a DOM instance, model the node as a non-WMS integrated node.

The uploading process is performed in two phases:

- Uploading the Updated Inventory Information
- Loading Inventory Information from a Node

3.4.1 Uploading the Updated Inventory Information

To keep inventory information between DOM and WMS instances in synchronization, use the InventoryChangeUpload-751 service.

3.4.1.1 The InventoryChangeUpload-751 Service

Inventory information needs to be transmitted to the DOM instance for all adjustment types other than RECEIPT, RETURN, and SHIPMENT.

(Inventory for these adjustment types would typically be transmitted by means of receipt or shipping interfaces). The InventoryChangeUpload-751 service is invoked from the WMS instance on the SUPPLY_CHANGE event of the INVENTORY_CHANGE transaction, which is raised whenever inventory changes at a node. To invoke the InventoryChangeUpload-751 service, ensure that the UploadInventoryChange action, under General>General>Actions>Inventory Synchronization, is configured on the SUPPLY_CHANGE event.

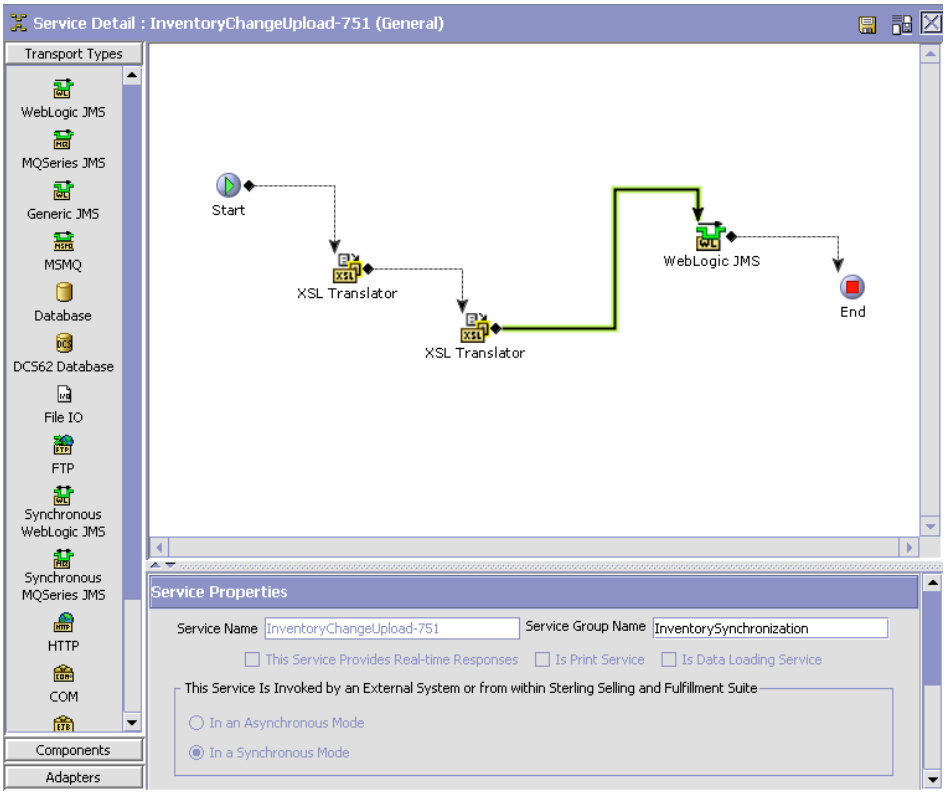
This service then translates the output of the SUPPLY_CHANGE event and creates an input XML for the adjustInventory API. This input XML is published as a message to the JMS queue of the web server of the DOM instance.

The new "doesAdjustmentTypeRequiresTransmission" condition is used to determine which inventory changes require transmission. This condition returns true if the adjustment type is any value other than RECEIPT, RETURN, and SHIPMENT.

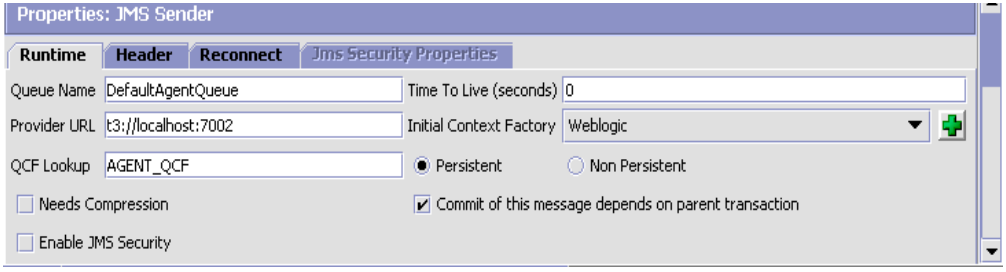
3.4.1.2 Configuring the Updated Inventory Information from a Node

To configure the service:

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the General tab. In the Process Types swimlane, right-click the General process type and select Model Process. The Repository Details window and work area displays for the General process type.
4. Click the Service Definitions tab.
5. Expand the InventorySynchronization branch.
6. Right-click InventoryChangeUpload-751 and select Details. The Service Detail window appears in the work area.



7. Click the green connector that connects the XSL Translator and the WebLogic JMS. The JMS Sender properties displays as shown.



8. In the Runtime tab, make sure that the "Commit of this message depends on parent transaction" box is checked.

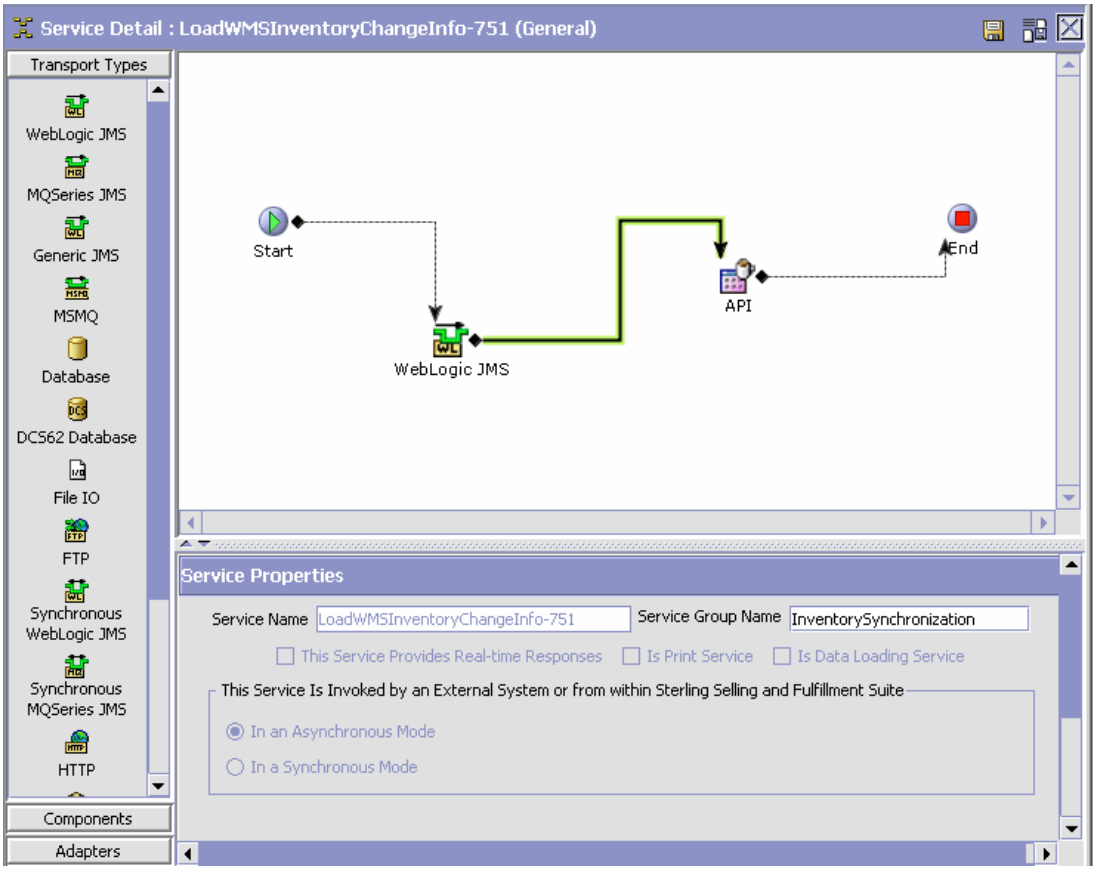
   For field value descriptions, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

3.4.2 Loading Inventory Information from a Node

In order to reconcile the inventory picture between DOM and WMS, the inventory picture at the WMS instance must be loaded to the DOM instance.

To reconcile the inventory picture, set up the LoadWMSInventoryChangeInfo-751 service for the DOM instance.

1. From the Applications menu of the Applications Manager, select Application Platform.
2. From the tree in the application rules side panel, double-click Process Modeling.
3. Click the General tab. In the Process Types swimlane, right-click the General process type and select Model Process. The Repository Details window and work area displays for the General process type.
4. Click the Service Definitions tab.
5. Expand the InventorySynchronization branch.
6. Right-click LoadWMSInventoryChangeInfo-751 and select details. The Service Detail window appears in the work area.



3.4.2.1 The LoadWMSInventoryChangeInfo-751 service

This service is invoked from the DOM instance.

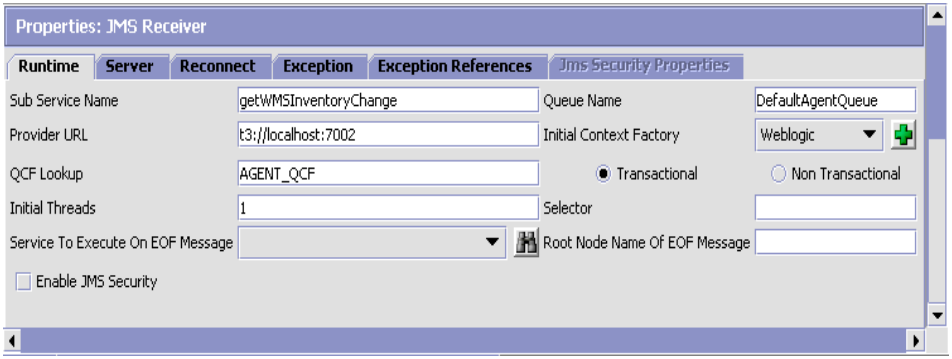
> **Note:** Although we have used WebLogic JMS as an example, Selling and Fulfillment Foundation also supports the use of IBM WebSphere and JBoss Messaging JMS.

From WebLogic JMS to API

The LoadWMSInventoryChangeInfo-751 service reads the message from the JMS queue and invokes the adjustInventory API.

To configure the service:

1. In the Service Detail: LoadWMSInventoryChangeInfo-751 window, click the green connector that connects the WebLogic JMS and API. The JMS Receiver properties displays as shown.



For field value descriptions, refer to the Service Builder Nodes and Parameters appendix of the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

3.5 Uploading the Inventory Snapshot

Selling and Fulfillment Foundation provides the ability to upload inventory snapshots for integrating WMS and DOM that are running on different instances. This involves loading the inventory picture from a WMS instance to a DOM instance.

3.5.1 Generating Inventory Snapshot Files

A single XML file is generated by running the inventory snapshot component at a WMS instance where data is fetched from the YFS_Inv_SnapShot_VW view. This view is derived from the following tables:

- YFS_INVENTORY_ITEM
- YFS_INVENTORY_SUPPLY
- YFS_INVENTORY_TAG

To run the inventory snapshot component on a WMS instance:

1. Go to the `<INSTALL_DIR>/bin` directory.

2. For UNIX or Linux, run this command:

```
sci_ant.sh -f runInventorySnapShot.xml
-DFilePath=<FilePath> -DShipNode=<ShipNode>
-DReasonCode=<ReasonCode> -DReasonText=<ReasonText>
-DItemsPerGroup=<ItemsPerGroup>
```

For Windows, run this command:

```
sci_ant.cmd -f runInventorySnapShot.xml
-DFilePath=<FilePath> -DShipNode=<ShipNode>
-DReasonCode=<ReasonCode> -DReasonText=<ReasonText>
-DItemsPerGroup=<ItemsPerGroup>
```

Table 3–1 Parameters Passed for Inventory Snapshot

Field	Description
FilePath	The absolute path of the directory where the generated XML file is stored.
ShipNode	The ship node for which the XML file is generated.
ReasonCode	The reason code that is defined by the user.
Reason Text	The reason code text that is that is defined by the user.
ItemsPerGroup	The number of item tags in the items tag element. The recommended value is 100. However, you could specify any value from 1 to 100.

Note: The time taken to generate an XML file on a WMS instance is not more than 3 minutes when the number of records in the YFS_INVENTORY_SUPPLY table are 430,000 and 512 M heap is used.

These generated XML files can be shared by both WMS and DOM instances through NFS mounts or can also be transferred through FTP to the DOM instance.

For more information about uploading inventory snapshot components on a DOM instance, refer to the *Sterling Global Inventory Visibility: Configuration Guide*.

4

Integrating with Third-Party Warehouse Management Systems

Selling and Fulfillment Foundation enables you to integrate with external third-party warehouse management systems in order to identify external ship nodes, manage external inventory and distribution of items, and coordinate external warehouse activities.

The Selling and Fulfillment Foundation, Release 8.5 provides complete functionality for Distributed Order Management and Warehouse Management systems without the need for integration. For more information about the Sterling Warehouse Management System, see the *Sterling Warehouse Management System: Concepts Guide*.

This chapter describes how Selling and Fulfillment Foundation provides integration with software that controls inventory and directs activities from shipping to receiving for third-party warehouse management systems.

4.1 Third-Party Warehouse Management Systems

Selling and Fulfillment Foundation provides XML-based integration to third-party warehouse management systems (WMS). To integrate Selling and Fulfillment Foundation with third-party warehouse management systems, configure them using services, as indicated in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*. In addition, use the following APIs when necessary:

- getUnprocessedImportDataEx() – Retrieves unprocessed data from import tables.

4.1.1 Third-Party Shipment Advice

When creating shipment advice data for third-party software, use services to stage your data. For more information about using services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

4.1.2 Third-Party Inventory Change

Selling and Fulfillment Foundation enables XML-based integration with third-party warehouse management inventory control systems through services or though system APIs. The following APIs enable integration with third-party systems for inventory change:

- getInventorySnapShot – Obtains total number of items in inventory at all ship nodes.
- getInventoryMismatch – Detects or corrects mismatches between the global inventory picture on Selling and Fulfillment Foundation and the global inventory picture on the external system.
- adjustInventory – Applies corrections to the global inventory picture in Selling and Fulfillment Foundation. This could also be used to correct a mismatch when the getInventoryMismatch API is used to detect the mismatches.

5

Integrating with the Loftware Print Server and Label Manager

This chapter deals with the specific settings required for a successful integration of the Sterling WMS with the Loftware Print Server (LPS) and Loftware Label Manager (LLM), and the configuration of custom prints using the same.

For more information about installing and configuring the Loftware Print Server, see the *Selling and Fulfillment Foundation: Installation Guide*.

For more information about server requirements and installation guidelines of Loftware Label Manager, see the *Loftware Print Server User's Guide* and *Loftware Label Manager User's Guide*.

For more information about configuring printers, see the *Sterling Warehouse Management System: Configuration Guide*.

The Sterling WMS provides the following Standard Labels:

- UCC 128 Container Shipping Label
- VICS Bill Of Lading
- Packing Slip
- Batch Sheets
    - Item Pick Batch Sheet
    - Cart Manifest Batch sheet
- Count Sheet
- UPS Carrier Label
- UPS Pickup Summary

Important: The factory shipped Cart Manifest Print is based on the following assumptions:
1. Each cart location is assumed to have 1 slot, when the number of locations in the cart is greater than 8,
2. Each cart location is assumed to have 2 slots, when the number of locations in the cart is less than or equal to 8.

For example, if the cart locations in the cart are named as A, B, C, ... H, then the Cart Manifest Print has locations such as A1, A2, B1, B2, C1, C2, ... H1, H2.

Thus, the task type "Number of containers allowed per location in the equipment" should always be set at 1 or 2.

For other configurations of the Cart, the Print has to be customized.

To print these standard labels, the Sterling WMS provides services associated with events. By default, the events are disabled. Enable the events if you want to print the standard labels. Refer to Table 5–1 "Services provided in the Sterling WMS" for a list of services provided in the Sterling WMS.

Table 5–1 Services provided in the Sterling WMS

Service Name	Event	Description
PrintShippingLabel	ADD_TO_CONTAINER.ON_CONTAINER_PACK_COMPLETE	Prints a UCC-128 Shipping Label for a container
PrintShipmentContainerLabels	Reprint Request from console	Prints UCC-128 Container Labels for Containers in the Shipment
PrintShipmentBOL	CONFIRM_SHIPMENT.ON_SUCCESS	Print a VICS BOL for Shipment

Table 5–1 Services provided in the Sterling WMS

Service Name	Event	Description
PrintTaskList	Reprint Request from console	Prints a BatchSheet (CartManifest or ItemPickBatch Sheet) or a CountSheet, based on the ActivityGroup for the Batch. If the Batch belongs to the ActivityGroup COUNT, the CountSheet is printed.
PrintLoadBOL	RECEIVE_IN_TRANSIT_ UPDATES.ON_SUCCESS	Prints a VICS BOL for Load
PrintWave	PRINT_WAVE.ON_ SUCCESS	Prints PickList (BatchSheets), Container Labels and pre-generates PackLists for Shipments in the Wave
PrintPackList	ADD_TO_CONTAINER.ON_ SHIPMENT_PACK_ COMPLETE	Prints a PackList
PickListPrint	PRINT_PICKLIST.ON_ SUCCESS	Prints PackLists for Shipments in the PickList
PrintTaskSheets	COMPLETE_TASK.TASK_ COMPLETED	Creates a Batch for successor Tasks of the completed task and Prints a BatchSheet for the same
PrintMoveTickets	RELEASE_MOVE_ REQUEST.ON_SUCCESS	Creates a Batch for the MoveRequest and prints a BatchSheet for the same
PrintPostPickContainers	POST_PICK_ CONTAINERIZATION.ON_ SUCCESS	Prints UCC-128 Shipping Labels for containers created as part of Post Pick Containerization

5.1 Designing Custom Labels

Use Loftware Label Manager to design a label (creates an `.lwl` file). For more information about creating new labels using Loftware Label Manager, see *Loftware Label Manager User's Guide*.

> **Note:** The Sterling WMS requires the repeating fields in a label to have names in the format of `<fieldname>_<integer>`. The integer in the field name takes values like 1, 2, 3.

The Loftware Label Manager, used for designing labels, may be installed on any compatible PC. For more information about server requirements and installation guidelines, see *Loftware Print Server User's Guide*.

> **Note:** While designing a custom label, it is recommended that you use the '.LST' file in order to maintain uniformity in label field names across different labels. For more information about LST file(s), see *Loftware Label Manager User's Guide.*

Displaying Page Numbers

To display Page Numbers and Total Number of Pages in the print output, the following fields need to be added to the Label (`.lwl` file):

- `PageNo`
- `TotalPages`

This ensures that the page numbers are displayed in the format Page X of N.

File Naming Convention

The Sterling WMS requires the following naming convention be followed while creating labels (`.lwl` files) using Loftware Label Manager:

- The first page of the label file created should be named in the format `<filename>.lwl`
- The middle page of the label file created should be named in the format `<filename>_Mid.lwl`

- The last page of the label file created should be named in the format <filename>_Last.lwl

The first page of the label and the last page of the label are always single pages. The middle page, on the other hand, is used n number of times in accordance with the total number of label pages to be printed.

For example, if a label print is six pages, the first page and last page make two pages, and the middle page (<filename>_Mid.lwl) is repeated four times.

You can print a label in single-page or multi-page format depending on the number of lines in the label. If the number of lines can be accommodated on the first page itself, you can print the label in single-page format. For this, you must create a new label format (<filename>_SinglePage.lwl). For more information about creating a label format, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*. After you create the new label format, the print service calls the xsl file to check the number of lines in the label. Depending on the number of lines, a single-page or multi-page label is printed. For example, the LTL Manifest Label can be printed in single-page or multi-page format.

After you create the custom label, copy it to Runtime > Template > Label > Extn directory.

Creation of Mapping XML File

The GenLabelMappingXML.java tool is used to generate Mapping XML for a label designed using Loftware Label Manager. The output XML contains all the field names of the label. XPath bindings for the label fields have to be specified.

To generate a Mapping XML for a label, use the command given below to invoke the GenLabelMappingXML tool:

```
java -classpath <classpath>
com.yantra.tools.labelxmlmapping.GenLabelMappingXML
<parameter1> <parameter2>
```

Ensure that the classpath has the following jar files:

- platform_afc.jar
- log4j-1.2.12.jar
- xercesImpl.jar

<parameter1>
This should be the file name of the .tab file generated when the label (.lwl) file is saved in Loftware Label Manager.

The full path, excluding the extension should be specified.

<parameter2>

This should be the file name of the XML file generated by the tool.

The full path, excluding the extension should be specified.

For example, to generate a Mapping XML for the label BOL.lwl, the .tab file name is BOL.tab

In this example, the command used to invoke the tool is:

```
java -classpath platform_
afc.jar;log4j-1.2.12.jar;xercesImpl.jar
com.yantra.tools.labelxmlmapping.GenLabelMappingXML
<path-of-the-file>/BOL <path-of-file>/BOLMap
```

XML File Settings

In the Mapping XML file generated using the GenLabelMappingXML.java tool:

- Each Label Field has a corresponding LabelField element
- Label Fields which are repeating are present in the RepeatingField element.
- Each of the Repeating Fields has a MaxFirstPage, MaxMidPage, and MaxLastPage, which denote the number of times the field is repeated in the First page, Middle Pages, and Last Page respectively.
- To repeat the same set of values of the field in all the pages, the RepeatValuesOnEachPage attribute should be set to "Y" in the RepeatingField element.

The following is an example of a Mapping XML file:

Example 5–1 Illustration of a Mapping XML

```
<?xml version="1.0" encoding="UTF-8"?>
<LabelFieldMap>
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@AddressLine1"
LabelFieldName="FromAddressLine1" RepeatingElement=""/>
```

```
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@AddressLine2"
LabelFieldName="FromAddressLine2" RepeatingElement=""/>
    <LabelField
Binding="concat(/Shipment/SellerOrganization/CorporatePersonInfo/@FirstName,
' ',/Shipment/SellerOrganization/CorporatePersonInfo/@LastName)"
LabelFieldName="FromName" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@City"
LabelFieldName="FromCity" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@State"
LabelFieldName="FromState" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@Country"
LabelFieldName="FromCountry" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/SellerOrganization/CorporatePersonInfo/@ZipCode"
LabelFieldName="FromZip" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/@ShipmentNo" LabelFieldName="ShipmentNo"
RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/@ActualShipmentDate"
LabelFieldName="ShipmentDate" RepeatingElement="" DataType="Date"/>
    <LabelField Binding="concat(/Shipment/ToAddress/@FirstName,'
',/Shipment/ToAddress/@LastName)" LabelFieldName="ToName"
RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@AddressLine1"
LabelFieldName="ToAddressLine1" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@AddressLine2"
LabelFieldName="ToAddressLine2" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@City" LabelFieldName="ToCity"
RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@State"
LabelFieldName="ToState" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@ZipCode"
LabelFieldName="ToZip" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/ToAddress/@Country"
LabelFieldName="ToCountry" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="concat(/Shipment/BillingInformation/AlternateParty/@FirstName,'
',/Shipment/BillingInformation/AlternateParty/@LastName)"
LabelFieldName="BillToName" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/BillingInformation/AlternateParty/@AddressLine1"
LabelFieldName="BillToAddressLine1" RepeatingElement="" DataType="Text"/>
```

```
    <LabelField
Binding="/Shipment/BillingInformation/AlternateParty/@AddressLine2"
LabelFieldName="BillToAddressLine2" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/BillingInformation/AlternateParty/@City"
LabelFieldName="BillToCity" RepeatingElement="" DataType="Text"/>
   <LabelField Binding="/Shipment/BillingInformation/AlternateParty/@State"
LabelFieldName="BillToState" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/BillingInformation/AlternateParty/@ZipCode"
LabelFieldName="BillToZip" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/BillingInformation/AlternateParty/@Country"
LabelFieldName="BillToCountry" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/Carrier/@ScacDesc" LabelFieldName="SCAC"
RepeatingElement="" DataType="Text"/>
    <LabelField Binding="/Shipment/BillingInformation/@ShipmentChargeType"
LabelFieldName="FreightTerms" RepeatingElement="" DataType="Text"/>
    <LabelField Binding="concat(/Shipment/MarkForAddress/@FirstName,'
',/Shipment/MarkForAddress/@LastName)" LabelFieldName="MarkFor"
RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/Instructions/Instruction[@InstructionType='SHIP']/@Instru
ctionText" LabelFieldName="SpecialInstruction" RepeatingElement=""
DataType="Text"/>
    <LabelField
Binding="/Shipment/ShipmentLines/ShipmentLine/OrderLine/@CustomerPONo"
LabelFieldName="CustomerPONo" RepeatingElement="ShipmentLine"
DataType="Text"/>
    <LabelField Binding="/Shipment/ShipmentLines/ShipmentLine/@ItemID"
LabelFieldName="ItemId" RepeatingElement="" DataType="Text"/>
    <LabelField
Binding="/Shipment/ShipmentLines/ShipmentLine/OrderLine/Item/@CustomerItem"
LabelFieldName="CustItemId" RepeatingElement="ShipmentLine"
DataType="Text"/>
    <LabelField
Binding="/Shipment/ShipmentLines/ShipmentLine/OrderLine/Item/@ItemDesc"
LabelFieldName="ItemDesc" RepeatingElement="ShipmentLine" DataType="Text"/>
    <LabelField
Binding="/Shipment/ShipmentLines/ShipmentLine/@UnitOfMeasure"
LabelFieldName="UOM" RepeatingElement="ShipmentLine" DataType="Text"/>
    <LabelField Binding="/Shipment/ShipmentLines/ShipmentLine/@OrderedQty"
LabelFieldName="OrdQty" RepeatingElement="ShipmentLine" DataType="Text"/>
    <LabelField Binding="/Shipment/ShipmentLines/ShipmentLine/@Quantity"
LabelFieldName="Quantity" RepeatingElement="ShipmentLine" DataType="Text"/>
    <LabelField
```

```
Binding="/Shipment/ShipmentLines/ShipmentLine/@BackOrderedQty"
LabelFieldName="BOQty" RepeatingElement="ShipmentLine" DataType="Text"/>
    <LabelField Binding="" LabelFieldName="Line" RepeatingElement=""
Sequence="Y" DataType="Text"/>
    <RepeatingFields>
        <RepeatingField LabelFieldName="CustomerPONo" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="ItemId" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="CustItemId" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="ItemDesc" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="UOM" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="OrdQty" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="Quantity" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="BOQty" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
        <RepeatingField LabelFieldName="Line" MaxFirstPage="12"
            MaxLastPage="12" MaxMidPage="12"/>
    </RepeatingFields>
</LabelFieldMap>
```

The map file (XML) generated for a label (LWL) must be edited to associate the XML data to the fields required on the label.

> **Note:** XPath Functions can be used in the binding, provided the XPath Binding for a `RepeatingField` represents a Nodeset.

3. **Sequence:**
   Sequence="Y" setting is to be used in instances where a labelfield represents a sequence of numbers. For example, serial numbers in a table.

4. **DataType:**
   Set up the relevant DataType for the `LabelField`. Valid values are Text, Date, and DateTime.

5. **Repeating Element:**
   Specify the `RepeatingElement` for the XPath Binding.

   If no Repeating Element is specified, the element containing the attribute is used as the `RepeatingElement` by default.

   In this example, the `ShipmentLine` is the `RepeatingElement`:

```
<LabelField
Binding="/Shipment/ShipmentLines/ShipmentLine/OrderLine/Ite
m/@ItemDesc" LabelFieldName="ItemDesc"
RepeatingElement="ShipmentLine" DataType="Text"/>
```

Relocation of XML Mapping File

The edited XML map file needs to be copied over into the Selling and Fulfillment Foundation Runtime Template folder:

1. Copy the relevant XML Mapping File from the folder where it has been generated.
2. Paste the copied XML Mapping File to Runtime > Template > Label > Extn directory.

5.2 Defining Custom Print Services

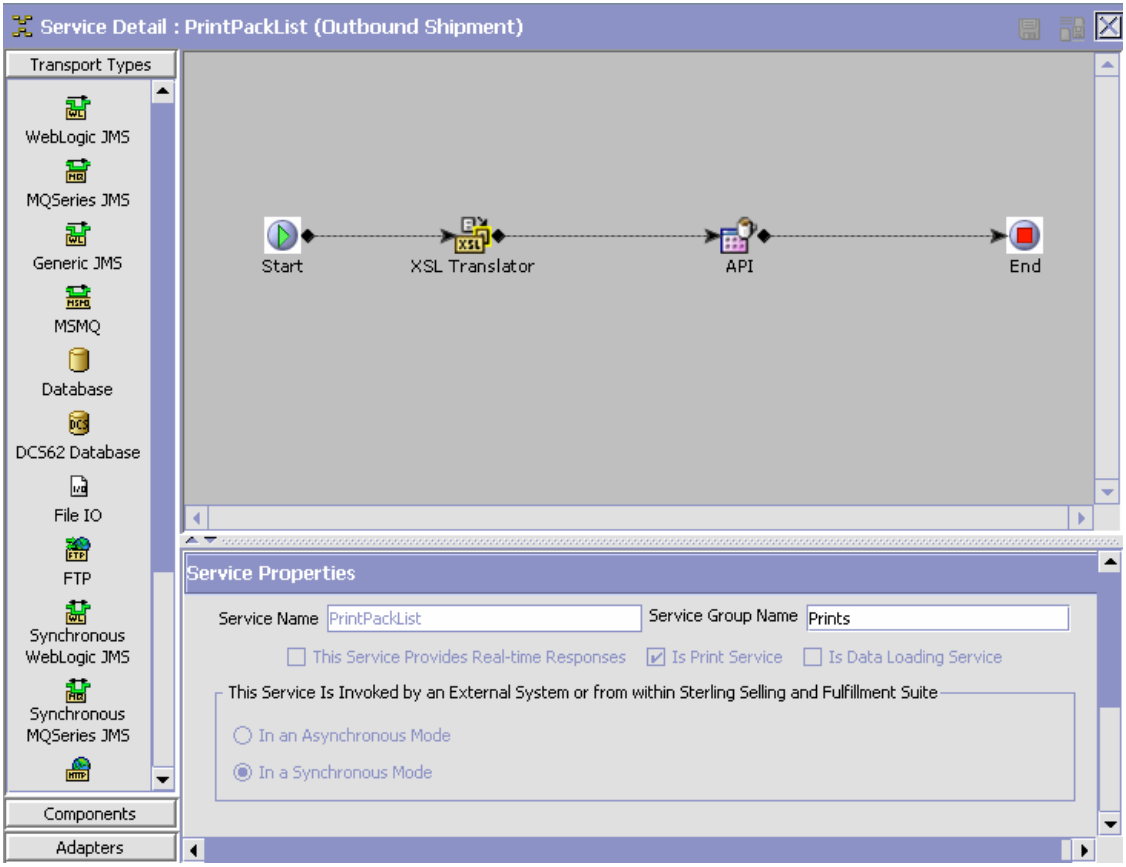
This section illustrates the services required for printing a Pack List. The services explained herein are supplied by default within the Selling and Fulfillment Foundation framework. The examples provided here may be used as a reference point to create custom Prints.

Creating Service Definitions

Prints are required to be configured as services to be invoked from an event or the console (UI).

To configure a Print Pack List service:

1. From the Application Platform tree, choose Process Modeling > Container > Pack Process. The Pack Process window is displayed.
2. Choose Actions tab. From Pack Process Repository > Prints, choose PrintPackList.
3. The Service Detail: PrintPackList (Pack Process) window is displayed.



For more information about configuring Service Details, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

The Input XML to the service definition is transformed into the input of the `PrintDocumentSet()` API using an XSL Translator.

For more information about the input to `PrintDocumentSet()` API and the description of the XML attributes, refer to the JavaDocs.

The following is an example of a typical XSL that generates the input to the `PrintDocumentSet()` API:

Example 5–2 XSL Translator Input to PrintDocumentSet() API

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version =
"1.0">
<xsl:output indent="yes"/>
<xsl:template match="Print | Shipment">
<PrintDocuments>
<xsl:attribute name="PrintName">
<xsl:text>packList</xsl:text>
</xsl:attribute>
<xsl:attribute name="FlushToPrinter">
<xsl:text>Y</xsl:text>
</xsl:attribute>
<PrintDocument>
<xsl:attribute name="BeforeChildrenPrintDocumentId">
<xsl:text>PACKLIST</xsl:text>
</xsl:attribute>
<xsl:attribute name="DataElementPath">
<xsl:text>xml:/Shipment</xsl:text>
</xsl:attribute>
<xsl:choose>
<xsl:when test="name()=&quot;Print&quot;">
<xsl:copy-of select="PrinterPreference"/>
<xsl:copy-of select="LabelPreference"/>
</xsl:when>
<xsl:when test="name()=&quot;Shipment&quot;">
<PrinterPreference>
<xsl:attribute name="PrinterId"/>
<xsl:attribute name="UsergroupId"/>
<xsl:attribute
name="UserId"><xsl:text>xml:/Shipment/@Modifyuserid</xsl:text></xsl:attribut
e>
<xsl:attribute name="WorkStationId"/>
<xsl:attribute
name="OrganizationCode"><xsl:text>xml:/Shipment/ShipNode/@NodeOrgCode</xsl:t
ext></xsl:attribute>
</PrinterPreference>
<LabelPreference>
<xsl:attribute name="EnterpriseCode">
<xsl:text>xml:/Shipment/@EnterpriseCode</xsl:text>
</xsl:attribute>
<xsl:attribute name="BuyerOrganizationCode">
<xsl:text>xml:/Shipment/@BuyerOrganizationCode</xsl:text>
</xsl:attribute>
```

```
<xsl:attribute name="SellerOrganizationCode">
<xsl:text>xml:/Shipment/@SellerOrganizationCode</xsl:text>
</xsl:attribute>
</LabelPreference>
</xsl:when>
</xsl:choose>
<KeyAttributes>
<KeyAttribute>
<xsl:attribute name="Name"><xsl:text>ShipmentKey</xsl:text></xsl:attribute>
</KeyAttribute>
</KeyAttributes>
<InputData>
<xsl:attribute name="FlowName">
<xsl:text>GetPackListData</xsl:text>
</xsl:attribute>
<Shipment>
<xsl:choose>
<xsl:when test="name()=&quot;Print&quot;">
<xsl:copy-of select="Shipment/@*" />
</xsl:when>
<xsl:when test="name()=&quot;Shipment&quot;">
<xsl:copy-of select="@*" />
</xsl:when>
</xsl:choose>
</Shipment>
<Template>
<Api Name="getShipmentDetails">
<Template>
<Shipment>
<SellerOrganization>
<CorporatePersonInfo/>
</SellerOrganization>
<Carrier/>
<MarkForAddress/>
<BillingInformation>
<AlternateParty/>
</BillingInformation>
<Instructions>
<Instruction/>
</Instructions>
<FromAddress/>
<ToAddress/>
<ShipmentLines>
<ShipmentLine CountryOfOrigin="" FifoNo="" ItemDesc="" ItemID=""
OrderHeaderKey="" OrderLineKey="" OrderNo="" OrderReleaseKey=""
```

```
PrimeLineNo="" ProductClass="" Quantity="" ReleaseNo="" Segment=""
SegmentType="" ShipmentKey="" ShipmentLineKey="" ShipmentLineNo=""
SubLineNo="" UnitOfMeasure="" BackOrderedQty="" ShipmentSubLineNo="">
<Order/>
<OrderLine>
<Item/>
<OrderStatuses>
<OrderStatus OrderHeaderKey="" OrderLineKey="" OrderLineScheduleKey=""
OrderReleaseKey="" OrderReleaseStatusKey="" PipelineKey="" ReceivingNode=""
ShipNode="" Status="" StatusDate="" StatusDescription="" StatusQty=""
StatusReason="" TotalQuantity="">
<OrderStatusTranQuantity StatusQty="" TotalQuantity="" TransactionalUOM=""
/>
<Details ExpectedDeliveryDate="" ExpectedShipmentDate="" ShipByDate=""
TagNumber="">
</Details>
</OrderStatus>
</OrderStatuses>
</OrderLine>
</ShipmentLine>
</ShipmentLines>
<Containers>
<Container>
<ContainerDetails>
<ContainerDetail>
<ShipmentLine>
<OrderLine>
<Item/>
</OrderLine>
</ShipmentLine>
</ContainerDetail>
</ContainerDetails>
</Container>
</Containers>
<ShipNode>
<ShipNodePersonInfo/>
</ShipNode>
</Shipment>
</Template>
</Api>
</Template>
</InputData>
</PrintDocument>
</PrintDocuments>
</xsl:template>
```

```
</xsl:stylesheet>
```

The Input XML to the above XSL translator should belong to either of the following formats:

```
<Shipment ShipmentKey=""/>
```

OR

```
<Print><Shipment ShipmentKey=""/><LabelPreference
EnterpriseCode=""/><PrinterPreference UserId=""
UsergroupId=""/></Print>
```

The former input XML is passed when the service is invoked from an event, while the latter is passed when the service is invoked from the console (UI).

The following is an example of the XML generated after the XSL Translation using the above mentioned XSL:

Example 5–3 XML Generated After XSL Translation

```
<?xml version = "1.0" encoding = "UTF-8"?>
<PrintDocuments PrintName="packList" FlushToPrinter="Y">
<PrintDocument Localecode="xml:/Shipment/ShipNode/@Localecode">
<InputData APIName="getShipmentDetails">
<Shipment ShipmentKey="">
</Shipment>
<Template>
<Shipment>
<ShipNode>
<ShipNodePersonInfo/>
</ShipNode>
</Shipment>
</Template>
</InputData>
</PrintDocument>
<PrintDocument BeforeChildrenPrintDocumentId="PACKLIST"
DataElementPath="xml:/Shipment">
<PrinterPreference PrinterId="" UserId="xml:/Shipment/@Modifyuserid"
UsergroupId="" WorkStationId=""
OrganizationCode="xml:/Shipment/ShipNode/@NodeOrgCode"/>
<LabelPreference EnterpriseCode="xml:/Shipment/@EnterpriseCode"
BuyerOrganizationCode="xml:/Shipment/@BuyerOrganizationCode"
SellerOrganizationCode="xml:/Shipment/@SellerOrganizationCode" />
<KeyAttributes>
```

```
<KeyAttribute Name="ShipmentKey"/>
</KeyAttributes>
<InputData FlowName="GetPackListData">
<Shipment ShipmentKey=""/>
<Template>
<Api Name="getShipmentDetails">
<Template>
<Shipment ShipmentKey="" ShipmentNo="" ActualShipmentDate=""
ExpectedShipmentDate="">
<SellerOrganization OrganizationCode="">
<CorporatePersonInfo AddressLine1="" AddressLine2="" FirstName=""
MiddleName="" LastName="" City="" State="" Country="" ZipCode="" />
</SellerOrganization>
<Carrier Scac="" ScacDesc=""/>
<MarkForAddress/>
<BillingInformation ShipmentChargeType=""/>
<Instructions>
<Instruction InstructionType="" InstructionText=""/>
</Instructions>
<ToAddress/>
<ShipmentLines>
<ShipmentLine ItemDesc="" ItemID="" OrderHeaderKey="" OrderLineKey=""
OrderNo="" OrderReleaseKey="" PrimeLineNo=""  Quantity="" ReleaseNo=""
ShipmentKey="" ShipmentLineKey="" ShipmentLineNo="" SubLineNo=""
UnitOfMeasure="" BackOrderedQty="" ShipmentSubLineNo="">
<Order OrderHeaderKey="" OrderNo="">
<PersonInfoBillTo AddressLine1="" AddressLine2="" FirstName="" MiddleName=""
LastName="" City="" State="" Country="" ZipCode=""  />
</Order>
<OrderLine  CustomerPONo=""  OrderLineKey="" OrderedQty=""
OriginalOrderedQty=""   Status="" StatusQuantity="" SubLineNo="" >
<Item CustomerItem=""/>
<OrderStatuses>
<OrderStatus OrderLineKey="" OrderReleaseStatusKey=""  Status=""
StatusQty="" TotalQuantity=""/>
</OrderStatuses>
</OrderLine>
</ShipmentLine>
</ShipmentLines>
<ShipNode NodeOrgCode=""/>
</Shipment>
</Template>
</Api>
</Template>
</InputData>
```

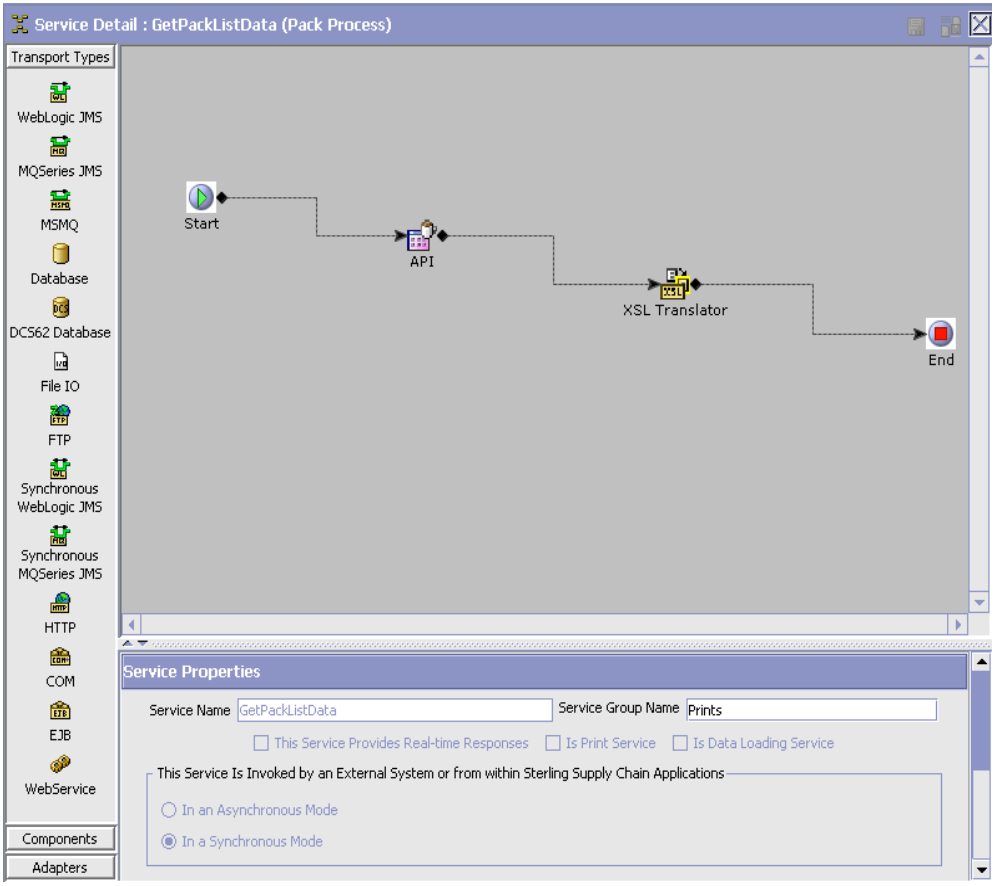
```
</PrintDocument>
</PrintDocuments>
```

This XML prints a Packing Slip (PACKLIST) as specified by the `BeforeChildrenPrintDocumentId` attribute in the `PrintDocument` node.

The data required to print the packlist is obtained by invoking the `GetPackListData` service as specified by the `FlowName` attribute in the `InputData` node.

To configure the `GetPackListData` service definition:

1. From the Application Platform tree, choose Process Modeling > Container > Pack Process. The Pack Process window is displayed.
2. Choose Service Definitions Tab. From Pack Process Repository > Prints, choose GetPackListData.
3. The Service Details: GetPackListData (Pack Process) window is displayed.



For more information about configuring Service Details, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

The `GetPackListData` service calls the `GetShipmentDetails()` API and the output is transformed using the XSL Translator.

The XSL translator (as reproduced below) calculates the backordered quantity for the shipment lines returned by the `GetShipmentDetails()` API:

Example 5–4 XSL Translator Output from GetShipmentDetails() API

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" version =
"1.0">
<xsl:output indent="yes"/>
<xsl:template match="/Shipment">
<Shipment>
<xsl:choose>
<xsl:when test="not(@ActualShipmentDate) or
(@ActualShipmentDate=&quot;&quot;)">
<xsl:attribute name="ActualShipmentDate"><xsl:value-of
select="@ExpectedShipmentDate"/></xsl:attribute>
</xsl:when>
<xsl:otherwise>
<xsl:attribute name="ActualShipmentDate"><xsl:value-of
select="@ActualShipmentDate"/></xsl:attribute>
</xsl:otherwise>
</xsl:choose>
<xsl:message>ActualShipmentDate<xsl:value-of
select="@ActualShipmentDate"/></xsl:message>
<xsl:for-each select="@*">
<xsl:if test="not(name()= &quot;ActualShipmentDate&quot;)">
 <xsl:attribute name="{name()}"><xsl:value-of select="."/></xsl:attribute>
</xsl:if>
</xsl:for-each>
<xsl:copy-of select="SellerOrganization"/>
<xsl:copy-of select="Carrier"/>
<xsl:copy-of select="ShipNode"/>
<xsl:copy-of select="ToAddress"/>
<xsl:copy-of select="MarkForAddress"/>
<xsl:copy-of select="BillingInformation"/>
<xsl:copy-of select="Instructions"/>
<xsl:copy-of select="Containers"/>
<ShipmentLines>
<xsl:for-each select="ShipmentLines/ShipmentLine[@ShipmentSubLineNo='0']">
<ShipmentLine>
<xsl:variable name="qty"
select="sum(OrderLine/OrderStatuses/OrderStatus[@OrderLineKey=current()/@Ord
erLineKey and substring(@Status,1,4)='1300']/@StatusQty)"/>
<xsl:attribute name="OrderedQty">
<xsl:value-of
select="sum(OrderLine/OrderStatuses/OrderStatus[@OrderLineKey=current()/@Ord
erLineKey and not(substring(@Status,1,4)='1400')]/@StatusQty)"/>
</xsl:attribute>
<xsl:attribute name="BackOrderedQty">
```

```
<xsl:value-of select="$qty"/>
</xsl:attribute>
<xsl:copy-of select="@*"/>
<xsl:copy-of select="OrderLine"/>
</ShipmentLine>
</xsl:for-each>
</ShipmentLines>
</Shipment>
</xsl:template>
</xsl:stylesheet>
```

Associating Services to Events

Once a service has been created for a print, it should be associated to an appropriate event. For more information about Service Association, see the *Sterling Warehouse Management System: Configuration Guide*.

6

Integrating with the Parcel Carrier Adapters

The Parcel Carrier Adapters (Carrier Adapter) manages all the carrier integration-related functions of Selling and Fulfillment Foundation. Selling and Fulfillment Foundation interfaces with the Carrier Adapter to use its carrier-integration functions.

The Carrier Adapter is regularly updated with the latest carrier data, such as rates and special services, and can act as a centralized carrier-integration database and business rules manager. The Carrier Adapter helps you to quickly meet the changing requirements initiated by both carriers and customers, in the most efficient way.

The Carrier Adapter has a data-driven design. The functionality is defined in terms of the relations between data elements stored in the database. Carriers having similar functionality can be incorporated into an installation with minimal engineering effort.

The Carrier Adapter is now integrated into Selling and Fulfillment Foundation. For more information about the Carrier Adapter and how to configure it, see the *Parcel Carrier: Adapter Guide*.

6.1 APIs Invoked During the Parcel Carrier Adapters Integration

The APIs invoked during the Sterling WMS integration with the Carrier Adapter are:

APIs Invoked During the Carrier Adapter Integration with UPSN

- openManifest API
- shipCarton API

- deleteCarton API
- closeManifest API

APIs Invoked During the Carrier Adapter Integration with FedEx

- openManifest API
- shipCarton API
- deleteCarton API
- closeManifest API

The Sterling WMS integrates with the Carrier Adapter using the following APIs:

- **openManifest API:** The openManifest API is used to open a manifest for a carrier server. This API calls the openManifest API in the Carrier Adapter. For field level mapping details between these APIs, see the section "Field-Level Mapping Between the openManifest API on the Sterling WMS and the openManifest API on the Carrier Adapter (Input XML)".
- **addContainertToManifest API:** The addContainerToManifest API is used to add a container to a manifest. This API calls the shipCarton API in the Carrier Adapter. For field level mapping details between these APIs, see the sections "Field-Level Mapping Between the addContainerToManifest API on Sterling WMS and the shipCarton API on the Carrier Adapter (Input XML)" and "Field-Level Mapping Between the addContainerToManifest API on the Sterling WMS and the shipCarton API on the Carrier Adapter (Output XML)".
- **removeContainerFromManifest API:** The removeContainerFromManifest API is used to delete a carton from a manifest. This API calls the deleteCarton API on the Carrier Adapter. For field level mapping details between these APIs, see the section "Field-Level Mapping Between the removeContainerFromManifest API on the Sterling WMS and the deleteCarton API on the Carrier Adapter (Input XML)".
- **closeManifest API:** The closeManifest API is used to close a manifest. This API calls the closeManifest API on the Carrier Adapter. For field level mapping details between these APIs, see the section

"Field-Level Mapping Between the closeManifest API on the Sterling WMS and the closeManifest API on the Carrier Adapter (Input XML)".

> **Note:** For the FedEx carrier, the Carrier Adapter supports label prints when a container is added to a manifest if the FedEx Printer is configured on the FedEx Carrier Server.
>
> For the UPSN carrier, the Carrier Adapter supports label prints when a container is added to a manifest or a manifest is closed.
>
> For more information about Label Prints, see the *Sterling Warehouse Management System: User Guide*.

Field-Level Mapping Between the openManifest API on the Sterling WMS and the openManifest API on the Carrier Adapter (Input XML)

Table 6–1 Mapping to the Carrier Adapter openManifest API

Field Name	Comments	Platform
Carrier	Required	YFS_Manifest.SCAC
ManifestNumber	Required	YFS_MANIFEST.manifest_no (as entered by the user. If not entered, posted with one upsequence number generated)
PickupSummaryNumber	Required for UPSN	YFS_MANIFEST.pickup_ summary_no (as entered by the user)
ShipperAccountNumber	Required	YFS_MANIFEST.shipper_ account_no(as entered by the user)
PickupDate	Required	YFS_MANIFEST.manifest_date (as entered by the user)

No output XML is generated for the openManifest API. A confirmation message is displayed on success, while an error message is displayed in the event of a failure.

Field-Level Mapping Between the addContainerToManifest API on Sterling WMS and the shipCarton API on the Carrier Adapter (Input XML)

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
UPSPLD		
Carrier	Required	YFS_SHIPMENT.scac
PackageLevelDetail	**The Package Level Detail Record (0100) - is written for every package shipped. This is a mandatory record for both domestic and international shipments.**	
ManifestNumber	Required	YFS_MANIFEST.manifest_no (open manifest as obtained by packShipment API for a given shipnode and carrier)
ShipId	Required	YFS_SHIPMENT_ CONTAINER.container_no.
PickupDate	Required	YFS_MANIFEST.manifest_date
ShipperAccountNumber	Required	YFS_MANIFEST.shipper_ account_no
BookNumber	Required	YFS_MANIFEST.pickup_ summary_no (substring 0-7)
PageNumber	Required	YFS_MANIFEST.pickup_ summary_no (substring 8-10)
ShipmentNumber	Required	YFS_SHIPMENT.shipment_no
PackageTrackingNumber	Required	<spaces>
SPFVersion	Required	Default 0505
Acctnumber	Conditional	Computed based on YFS_ FREIGTH_TERMS.charges_ paid_by. It can be YFS_ SHIPMENT.Custcarrier_ Account_No/YFS_SCAC_ Ex.account1.

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
CompanyName	Required	YFS_PERSON_INFO.company corresponding to YFS_ SHIPMENT.to_address_key.
ConsigneeAttn	Conditional	YFS_PERSON_INFO.first_ name + YFS_PERSON_ INFO.middle_name + YFS_ PERSON_INFO.last_name corresponding to YFS_ SHIPMENT.to_address_key.
CAddr1	Required	YFS_PERSON_INFO.address_ line1 corresponding to YFS_ SHIPMENT.to_address_key.
CAddr2	Optional	YFS_PERSON_INFO.address_ line2 corresponding to YFS_ SHIPMENT.to_address_key.
CAddr3	Optional	YFS_PERSON_INFO.address_ line3 corresponding to YFS_ SHIPMENT.to_address_key.
CCity	Required	YFS_PERSON_INFO.city corresponding to YFS_ SHIPMENT.to_address_key.
CStateProv	Conditional	YFS_PERSON_INFO.state corresponding to YFS_ SHIPMENT.to_address_key.
CPostalCode	Conditional	YFS_PERSON_INFO.zip_code corresponding to YFS_ SHIPMENT.to_address_key.
CPhone	Conditional	YFS_PERSON_INFO.day_ phone corresponding to YFS_ SHIPMENT.to_address_key.
ShipmentChgType	Required	Computed based on YFS_ FREIGHT_TERMS.charges_ paid_by and corresponding YFS_SCAC_Ex entry. Possible values are COL,TPB, PRE.
CWTInd	Conditional	Set to '0' (zero) to indicate Not HunderedWeight.

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
ServiceType	Required	YFS_SCAC_AND_SERVICE.electronic_code corresponding to YFS_SHIPMENT.scac and YFS_SHIPMENT.carrier_service_code.
Packagetype	Required	"02" to indicate Package.
DeliveryZone	Optional	<spaces>
Actualweight	Required	YFS_SHIPMENT_CONTAINER.container_gross_weight after applying the carrier locale weight UOM.
PkgpublishedDimWt	Required	Computed
UOMWeight	Optional	Weight UOM of the Ship Node
UOMDim	UOM Dim	Dimension UOM of the Ship Node
CODAmount	Required	0
CODFundsInd	Conditional	<spaces>
Currencycode	Required	YFS_SHIPMENT.currency.
CallTag_ARSInd	Required	0 - to indicate no call tag.
Calltag_ARSSchedulePickDate	Optional	<spaces>
MerchandiseDescription	Conditional	<spaces>
SatDeliveryInd	Required	"0" for not opting for this service.
SaturdayPickupInd	Required	"0" for not opting for this service.
OversizePackageInd	Required	YFS_SHIPMENT_CONTAINER.oversized_flag is Y, then indicator is passed as 1, or else 0.
DeclaredValueInsurance	Required	YFS_SHIPMENT_CONTAINER.declared_value

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
ResInd	Required	YFS_PERSON_INFO.company corresponding to YFS_ SHIPMENT.to_address_key is nonblanks, it is assumed to be 0 to indicate commercial or else 1 for residential.
DCISType	Conditional	<spaces>
CustomerRefNumberType1	Optional	<spaces>
CustomerRefNumber1	Optional	<spaces>
CustomerRefNumberType2	Optional	<spaces>
CustomerRefNumber2	Optional	<spaces>
ShipmentReferenceNoType1	Optional	<spaces>
ShipmentReferenceNo1	Optional	<spaces>
ShipmentReferenceNoType2	Optional	<spaces>
ShipmentReferenceNo2	Optional	<spaces>
CODControlNumber	Optional	<spaces>
CallTag_ARSNumber	Optional	<spaces>
CODInd	Required	<spaces>
CODCurrencycode	Conditional	<spaces>
IncrementalPldInd	Required	<spaces>
DocInd	Required	Default to ‘3’ to indicate non document/package.
ShipperEIN	Optional	<spaces>
ShipperCountry	Required	YFS_PERSON_INFO.country corresponding to YFS_ SHIPMENT.shipnode_key's YFS_SHIP_NODE.shipnode_ address_key.
SenderName	Optional	<spaces>
ConsigneeTagID	Optional	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
ConsigneeCountry	Required	YFS_PERSON_INFO.country corresponding to YFS_ SHIPMENT.to_address_key.
CalculatedRatesInd	Required	<spaces>
SourceTypeCode	Required	Default to ‘20’ to indicate host access.
AccessorialRecord	**AccessorialRecord (0200) is valid for both domestic and international shipments. This record is written only when UPS special services are used.**	
ShipperCreditCardNo	Required	<spaces>
ShipperCreditCardExpDate	Required	<spaces>
AdditionalHandlingInd	Required	Default to ‘0'.
ExtendedDestInd	Required	<spaces>
HazMat	Required	YFS_SHIPMENT.hazardous material is Y, then indicator is 1, else 0.
HoldForPickupInd	Required	Default to '0' (do not hold for pickup).
ModifyInd	Required	Default to ‘0’.
OCAIndicator	Required	Default to ‘0’.
VoidInd	Required	0
PackageLength	Required	YFS_SHIPMENT_ CONTAINER.container_length
PackageWidth	Required	YFS_SHIPMENT_ CONTAINER.container_width
PackageHeight	Required	YFS_SHIPMENT_ CONTAINER.container_height
SpecialInstructions	Optional	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
VerbalConfirmationName	Conditional	YFS_PERSON_INFO.first_name + YFS_PERSON_INFO.middle_name + YFS_PERSON_INFO.last_name corresponding to YFS_SHIPMENT.to_address_key.
VerbalConfirmationPhone	Conditional	YFS_PERSON_INFO.day_phone corresponding to YFS_SHIPMENT.to_address_key.
EarliestDeliveryTime	Optional	<spaces>
ShipmentCreditCardNumber	Conditional	<spaces>
ShipmentCreditCardExpDate	Conditional	<spaces>
ConsigneeNumber	Optional	<spaces>
ConsigneeCreditCardNo	Required	<spaces>
ConsigneeCreditCardExpDate	Required	<spaces>
DCISNumber	Optional	<spaces>
ConsigneeFaxDestinationInd	Optional	<spaces>
ConsigneeFax	Optional	<spaces>
ExperssCODTrackingNumber	Required	<spaces>
CustomerReferenceNumberType3	Optional	<spaces>
CustomerReferenceNumber3	Optional	<spaces>
CustomerReferenceNumberType4	Optional	<spaces>
CustomerReferenceNumber4	Optional	<spaces>
CustomerReferenceNumberType5	Optional	<spaces>
CustomerReferenceNumber5	Optional	<spaces>
PackageTrackingNumber	Required	YFS_Shipment_Container.Tracking_No

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
AlternatePartyRecord	**AlternateParty Record (0300) is valid for both domestic and international shipments. For domestic, this record is written only when freight term is 'Third Party Billing'. For International shipments, this record is written for Importer and Exporter Address.**	
AlternatePartyType	Required	For domestic shipments: This field is set to '03'/'04'. For international shipments: This field is set to '02' always.
ID_AcctNumber	Conditional	YFS_SCAC_EX.account1
PODReplyType	Conditional	<spaces>
APCompanyName	Required	YFS_PERSON_INFO.company corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.
APAttention	Conditional	YFS_PERSON_INFO.first_ name + YFS_PERSON_ INFO.last_name corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.
APAddr1	Required	YFS_PERSON_INFO.address_ line1 corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.
APAddr2	Optional	YFS_PERSON_INFO.address_ line2 corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.
APAddr3	Optional	YFS_PERSON_INFO.address_ line3 corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.
APCity	Required	YFS_PERSON_INFO.city corresponding to YFS_ SHIPMENT.enterprise_code's billing_address_key.

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
APStateProv	Conditional	YFS_PERSON_INFO.state corresponding to YFS_SHIPMENT.enterprise_code's billing_address_key. **Note**: This field value can only contain a maximum of 5 characters.
APPostalCode	Conditional	YFS_PERSON_INFO.zip_code corresponding to YFS_SHIPMENT.enterprise_code's billing_address_key.
APcountry	Required	YFS_PERSON_INFO.country corresponding to YFS_SHIPMENT.enterprise_code's billing_address_key. If International it is hardcoded to 'US'.
Filler1	Required	
APPhone	Conditional	YFS_PERSON_INFO.day_phone_no corresponding to YFS_SHIPMENT.enterprise_code's billing_address_key
APFaxDestInd	Conditional	<spaces>
APFax	Optional	<spaces>
LangCode	Optional	<spaces>
CreditCardNo	Required	<spaces>
CreditCardExpDate	Required	<spaces>
TaxId	Optional	<spaces>
AddrType	Required	<spaces>
PackageTrackingNumber	Required	YFS_Shipment_Container.Tracking_No
AdvisoryInformationRecord	**AdvisoryInformationRecord (0400) is required for E-mail or Fax Shipment Notification.**	
AdvisoryInfoLevel	Required	Default to 'P'.

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
SNFaxDestInd1	Conditional	If YFS_PERSON_INFO.day_fax_no != "" set this field to 0. US, PR, CA, and VI Fax/Phone only 1 Fax/Phone to all other countries.
SNFaxNumber1	Conditional	YFS_PERSON_INFO.day_fax_no corresponding to YFS_SHIPMENT.to_address_key.
SNLangCode	Optional	<spaces>
SNCompName1	Optional	YFS_PERSON_INFO.company corresponding to YFS_SHIPMENT.to_address_key.
SNAttnName1	Conditional	YFS_PERSON_INFO.first_name + YFS_PERSON_INFO.middle_name + YFS_PERSON_INFO.last_name corresponding to YFS_SHIPMENT.to_address_key.
SNContactPhone1	Conditional	YFS_PERSON_INFO.day_phone corresponding to YFS_SHIPMENT.to_address_key.
SNFaxDestInd2	Conditional	<spaces>
SNFaxNumber2	Conditional	<spaces>
SNLangCode2	Optional	<spaces>
SNCompanyName2	Optional	<spaces>
SNAttnName2	Conditional	<spaces>
SNContactPhone2	Conditional	<spaces>
AltrofileAccessNumber	Required	<spaces>
SNTypeDestination1	Required	<spaces>
SNEmailAddrDest1	Conditional	YFS_PERSON_INFO.email_id corresponding to YFS_SHIPMENT.to_address_key.
SNTypeDestination2	Required	Set to ‘0’
SNEmailAddrDest2	Conditional	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
SNMemo	Optional	<spaces>
PackageTrackingNumber	Required	YFS_Shipment_ Container.Tracking_No
InternationalRecord	**InternationalRecord (0500) is required if Importer, Exporter, Shipper To Consignee, or Commodity information is provided and whenever shipper and consignee countries are not the same. This record is written once for one shipment. If a shipper has 3 packages, only one 0500 record is written, whereas three 0100 records are written.**	
RecordType	Required	0500
InvoiceDate	Optional	YFS_MANIFEST.manifest_date (manifest no from YFS_ SHIPMENT).
WaybillPrintInd	Conditional	0
InvoiceLineTotals	Required	YFS_CONTAINER_ DETAILS.quantity * YFS_ ORDER_LINE * unit_price (for all lines in the container).
InvoiceCurrencyCode	Conditional	YFS_SHIPMENT.currency
ShipmentInsuranceDeclaredValue	Required	YFS_MANIFEST.manifest_date (manifest no from YFS_ SHIPMENT).
ConsolidatedClearQty	Required	0
UltimateDestCountry	Conditional	YFS_PERSON_INFO.country corresponding to YFS_ SHIPMENT.to_address_key.
Filler		<spaces>
SEDCode	Optional	<spaces>
ShipmentSEDCASNum	Optional	<spaces>
InvoiceNumber	Optional	YFS_SHIPMENT.shipment_no
PONumber	Optional	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
DescriptionOfGoods	Required	YFS_ITEM.nmfc_code. Item_ Id taken from CONTAINER_ DETAILS.item_id with YFS_ SHIPMENT_ CONTAINER.container_no (leadpackage) as criteria.
SpecialInstructions	Optional	<spaces>
PartiesToTrans	Conditional	<spaces>
TermsOfShipment	Optional	<spaces>
PaymentTerms	Optional	<spaces>
Filler		<spaces>
FreightCharges	Required	0
InsuranceCharges<	Required	0
DiscountRebate	Required	0
OtherCharges	Required	0
WaybillNumber/BrokerageID	Conditional	YFS_SHIPMENT.shipment_no
COCode	Optional	<spaces>
OtherDocCode	Optional	<spaces>
ReasonForExport	Optional	<spaces>
InvoiceSubTotal	Required	<spaces>
TotalInvoiceAmount	Required	<spaces>
BrokerCode	Optional	<spaces>
DestinationControl	Conditional	<spaces>
ShipmentCommodityOrigin	Conditional	<spaces>
Filler3	Required	
PackageTrackingNumber	Required	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
CommodityRecord	**CommodityRecord (0600) contains commodity information that is used for rating and customs clearance purposes. It is required if the shipment travels within the European Union and contains "Goods Not in Free Circulation". One 0600 record is written for each line in the shipper. If a shipper on the Sterling WMS has 4 records in the YFS_SHIPMENT_DTL table, four 0600 records are written.**	
RecordType	Required	0600
InvoiceLineNumber	Required	YFS_SHIPMENT_LINE.prime_line_no for the corresponding YFS_CONTAINER_DETAILS record.
CommodityCode	Optional	YFS_ITEM.harmonized_code of YFS_CONTAINER_DETAILS.item_id (catalog org and uom).
PartNumber	Optional	YFS_ITEM.item_id of YFS_CONTAINER_DETAILS.item_id (catalog org and uom).
LineOriginCountry	Required	YFS_ITEM.country_of_origin of YFS_CONTAINER_DETAILS.item_id (catalog org and uom).
LineCurrencyCode	Optional	YFS_SHIPMENT.currency
ECCN	Optional	YFS_ITEM.eccn_no of YFS_CONTAINER_DETAILS.item_id (catalog org and uom).
LineUnitAmtPrice	Required	YFS_ORDER_LINE.line_price of YFS_CONTAINER_DETAILS.order_line_key * YFS_CONTAINER_DETAILS.quantity. If its shipment container, we compute by getting item object from shipment and shipment container.

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
LineQuantity	Required	sum(YFS_CONTAINER_ DETAIL.quantity) for every unique item.
LineQtyUOM	Required	YFS_CONTAINER_ DETAILS.uom
LineLicenseInfo	Conditional	YFS_SHIPMENT_ CONTAINER.export_license_ no
LineLicenseExpDate	Conditional	YFS_SHIPMENT_ CONTAINER.export_license_ exp_date
LineMerchDesc1	Required	YFS_ITEM.item_desc of YFS_ CONTAINER_DETAILS.item_id (catalog org and uom).
LineMerchDesc2	Optional	<spaces>
LineMerchDesc3	Optional	<spaces>
CertOfOriginNo	Optional	YFS_SHIPMENT.shipment_no
CertOfOriginCode	Conditional	<spaces>
AgreementType	Optional	<spaces>
CommodityRemarks	Optional	<spaces>
QuantityScheduledUnits	Conditional	YFS_CONTAINER_ DETAIL.quantity
Marks&Numbers	Optional	<spaces>
CommodityWeight	Required	YFS_ORDER_LINE.item_ weight of YFS_CONTAINER_ DETAILS.order_line_key * YFS_CONTAINER_ DETAILS.quantity. If its shipment conatainer, we compute by getting the item object from shipment and shipment container.
NumberOfPackagesPerCmmdt y	Conditional	<spaces>

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
SEDLineAmt	Required	YFS_ORDER_LINE.line_price of YFS_CONTAINER_ DETAILS.order_line_key * YFS_CONTAINER_ DETAILS.quantity. If its shipment conatainer, we compute by getting the item object from shipment and shipment container.
COType	Required	Defaulted to 0.
SEDInd	Required	Defaulted to 0.
LineExtendedAmt	Required	YFS_ORDER_LINE.line_price of YFS_CONTAINER_ DETAILS.order_line_key * YFS_CONTAINER_ DETAILS.quantity. If its shipment conatainer, we compute by getting the item object from shipment and shipment container.
Filler		<spaces>
PackageTrackingNumber	Required	YFS_Shipment_ Container.Tracking_No
AdditionalCommentsRecord	AdditionalCommentsRecord **(0700) contains additional statements and information for an international shipment.**	
RecordType	Required	0700
DeclarationStatement	Optional	<spaces>
AdditionalComments	Optional	<spaces>
Filler1		<spaces>
PackageTrackingNumber	Required	<spaces>
SpecialServicesRecord	**SpecialServicesRecord contains SpecialService child elements for each of the special service the shipment/order have.**	
Service	Optional	YFS_SPECIAL_SERVICE_ REF.service_code

Table 6–2 Mapping to the Carrier Adapter shipCarton API (Input XML)

Field Name	Comments	Platform
ExtraFieldsRecord	**ExtraFieldsRecord contains statements and information extra fields.**	
LableFormatValue	Optional	<spaces>
ReferenceNotes	Optional	YFS_SHIPMENT.shipment_no+YFS_SHIPMENT_CONTAINER.container_Scm.
SunDeliveryInd	Optional	<spaces>
ThermalLabelPrinterID	Optional	Determined by calling getPrinterId.

Field-Level Mapping Between the addContainerToManifest API on the Sterling WMS and the shipCarton API on the Carrier Adapter (Output XML)

Table 6–3 Mapping to the Carrier Adapter shipCarton API (Output XML)

Field Name	Platform
TotalErrors	The total number of errors returned by the Carrier Server
ErrorCode	The error code returned by the Carrier Server
ErrorDescription	The description of the error code returned by the Carrier Server.
CODReturnTrackingNo	YFS_SHIPMENT_CONTAINER.COD_Return_tracking_No
TrackingNumber	YFS_SHIPMENT_CONTAINER.tracking_no
TotalSurchargeAmt	YFS_SHIPMENT_CONTAINER.special_services_surcharge
NetCharge	YFS_SHIPMENT_CONTAINER.actual_freight_charge
BilledWeight	YFS_SHIPMENT_CONTAINER.applied_weight
PrintBuffer	The print buffer returned by the Carrier Server.
DeliveryDay	YFS_SHIPMENT_CONTAINER.delivery_day

Table 6–3 Mapping to the Carrier Adapter shipCarton API (Output XML)

Field Name	Platform
UPS_Routing_Code	YFS_SHIPMENT_CONTAINER.UPS_Routing_Code

Field-Level Mapping Between the removeContainerFromManifest API on the Sterling WMS and the deleteCarton API on the Carrier Adapter (Input XML)

Table 6–4 Mapping to the Carrier Adapter DeleteCarton API

Field Name	Comments	Platform
Carrier	Required	YFS_SHIPMENT.scac
MeterNo	Required only for FedEx	YFS_SCACEx.portal_account_2
TrackingNumber	Required	YFS_SHIPMENT_CONTAINER.tracking_no of the package that is being unpacked or removed from the manifest.

No output XML is generated for the removeContainerFromManifest API. A confirmation message is displayed on success, while an error message is displayed in the event of a failure.

Field-Level Mapping Between the closeManifest API on the Sterling WMS and the closeManifest API on the Carrier Adapter (Input XML)

Table 6–5 Mapping to closeManifestAPI

Field Name	Comments	Platform
Carrier	Required	YFS_SHIPMENT.scac
ManifestNumber	Required	YFS_MANIFEST.manifest_no (as generated on the platform for the ship node and carrier combination)

Table 6–5 Mapping to closeManifestAPI

Field Name	Comments	Platform
PickupSummaryNumber	Required for UPSN	YFS_MANIFEST.pickup_ summary_no (as keyed in from the user)
ShipperAccountNumber	Required	YFS_MANIFEST.shipper_ account_no

No output XML is generated for the closeManifest API. A confirmation message is displayed on success, while an error message is displayed in the event of a failure.

6.2 Integration Dependencies

Sterling WMS integration with the Carrier Adapter is dependent on the following:

- Carrier Adapter APIs are called only if SCAC Integration is required for the Shipment. This is set up at Node/SCAC level.

7

Integrating with Material Handling Equipment

The Sterling WMS can integrate with various material handling equipment (MHE).

The automation enabled through the integration enables increased efficiency in various processes of a warehouse, like Receiving, Picking, Packing, Putaway or Replenishment, Outbound QC, VAS, Manifesting, Weighing, Item Measurements, and Trailer Loading.

7.1 Integration Overview

The material handling equipments that the Sterling WMS can integrate with include:

- Pick-to-Light
- Put-to-Light
- Carousels or Automated Storage & Retrieval System (ASRS)
- Automatic Guided Vehicle (AGV)
- Inbound Sorter
- Pack Sorter
- Shipping Sorter
- Cube-a-Scan
- Weighing Scale

7.2 Integrating with Pick-to-Light System

The Sterling WMS integrates with the pick-to-light systems after the Sterling WMS allocates and creates pick/move tasks.

1. For tasks that are in the pick-to-light zone, details regarding shipment/batch/carton (reference tag) level that indicate item and quantity to pick are sent to the system.

APIs Involved

- createTask()
- changeTask()
- createBatch()
- getTaskList()
- cancelTask()

Events Raised

The following event is raised by the createTask() API:

- CREATE_TASK.TASK_CREATED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

The following event is raised by the createBatch() API:

- CREATE_BATCH.BATCH_CREATED

The following event is raised by the cancelTask() API:

- CANCEL_TASK.TASK_CANCELED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

2. References are scanned in the pick-to-light system and appropriate slots are lit indicating quantity to pick.
3. Upon pick completion, status information is sent from the pick-to-light system to the Sterling WMS. All serial/tag number level

information required for pick completion is also passed back to the Sterling WMS.

APIs Involved

- registerTaskCompletion()
- registerBatchCompletion()
- changeTask()

Events Raised

The following events are raised by the registerTaskCompletion() and registerBatchCompletion() APIs:

- COMPLETE_TASK.TASK_COMPLETED
- COMPLETE_BATCH.BATCH_COMPLETED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.3 Integrating with Put-to-Light System

The Sterling WMS integrates with the put-to-light systems after the Sterling WMS allocates and creates pick/move tasks after wave release.

1. For tasks that are in the put-to-light zone, details regarding shipment/order level that indicate item and quantity to pick are sent to the system. The Sterling WMS is configured to create the required number of shipments in a wave, to match the number of slots.

APIs Involved

- getShipmentDetails()
- createTask()
- changeTask()

- createBatch()
- getTaskList()
- cancelTask()

Events Raised

The following event is raised by the createTask() API:

- CREATE_TASK.TASK_CREATED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

The following event is raised by the createBatch() API:

- CREATE_BATCH.BATCH_CREATED

The following event is raised by the cancelTask() API:

- CANCEL_TASK.TASK_CANCELED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

2. Item Ids are scanned in the put-to-light system, and appropriate slots are lit indicating quantity to be placed.
3. Container numbers are associated to each slot, and the container is closed. This information is sent back to the Sterling WMS.

APIs Involved

- registerTaskCompletion()
- registerBatchCompletion()
- addToContainer()
- changeTask()

Events Raised

The following events are raised by the registerTaskCompletion() and registerBatchCompletion() APIs:

- COMPLETE_TASK.TASK_COMPLETED
- COMPLETE_BATCH.BATCH_COMPLETED

The following events are raised by the addToContainer() API:

- CREATE_CONTAINER.ON_SUCCESS
- ADD_TO_CONTAINER.ON_SUCCESS
- ADD_TO_CONTAINER.ON_CONTAINER_PACK_COMPLETE
- ADD_TO_CONTAINER.ON_CONTAINER_PACK_PROCESS_COMPLETE
- ADD_TO_CONTAINER.ON_SHIPMENT_PACK_COMPLETE
- ADD_TO_CONTAINER.ON_SHIPMENT_PACK_PROCESS_COMPLETE

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

4. Quantities at the shipment level (each slot) are taken to appropriate packing locations to complete packing steps.

7.4 Integrating with Carousel or Automated Storage and Retrieval System

The Sterling WMS can integrate with Carousels or Automated Storage and Retrieval Systems (ASRS) during these instances:

- Integration When a Product is Being Put Away
- Integration When a Product is Being Retrieved
- Integration When a Product is Being Counted

7.4.1 Integration When a Product is Being Put Away

When a product is being put away, the Sterling WMS integrates with Carousels or Automated Storage and Retrieval Systems (ASRS) as follows:

1. The first step task brings the product to the drop-off location attached to the carousel/ASRS location. Upon completion of this task secondary step tasks are created. These secondary tasks based on task type and zone are sent to the carousel system.

APIs Involved

- createTask()
- createBatch()

Events Raised

The following event is raised by the createTask() API:

- CREATE_TASK.TASK_CREATED

The following event is raised by the createBatch() API:

- CREATE_BATCH.BATCH_CREATED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

2. User scans item for putaway into carousel system, which retrieves appropriate location/bin to the user station. Product is placed in the bin.
3. Upon the location/bin being placed in appropriate slot, the task completion information is sent to WMS. All serial/tag number level information required for pack completion is also passed back to WMS

APIs Involved

- registerTaskCompletion()
- registerBatchCompletion()
- changeTask()

Events Raised

The following events are raised by the registerTaskCompletion() and registerBatchCompletion() APIs:

- COMPLETE_TASK.TASK_COMPLETED
- COMPLETE_BATCH.BATCH_COMPLETED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.4.2 Integration When a Product is Being Retrieved

When a product is being retrieved, the Sterling WMS integrates with Carousels or Automated Storage and Retrieval Systems (ASRS) as follows:

1. Tasks created to retrieve product from the carousel/ASRS are sent from the Sterling WMS.

APIs Involved

- createTask()
- createBatch()

Events Raised

The following event is raised by the createTask() API:

- CREATE_TASK.TASK_CREATED

The following event is raised by the createBatch() API:

- CREATE_BATCH.BATCH_CREATED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

2. User initiates retrieval on carousel system and selects task for retrieval. On retrieval, system sends completion of task from bin/location to drop-off location at user station.

APIs Involved

- registerTaskCompletion()
- registerBatchCompletion()
- changeTask()

Events Raised

The following events are raised by the registerTaskCompletion() and registerBatchCompletion() APIs:

- COMPLETE_TASK.TASK_COMPLETED
- COMPLETE_BATCH.BATCH_COMPLETED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

3. Secondary step tasks are automatically created by the Sterling WMS to putaway quantity to final destination location.

7.4.3 Integration When a Product is Being Counted

When a product is being counted, the Sterling WMS integrates with Carousels or Automated Storage and Retrieval Systems (ASRS) as follows:

- User on the Sterling WMS is given location to count.
- This is entered on carousel system for location retrieval.
- Count is completed on the Sterling WMS.

7.5 Integrating with Automatic Guided Vehicle

The Sterling WMS integrates with Automatic Guided Vehicles (AGV) to complete putaway or pick. These interfaces are task-based integrations.

APIs Involved

- createTask()
- changeTask()
- createBatch()
- getTaskList()
- cancelTask()

Events Raised

The following event is raised by the createTask() API:

- CREATE_TASK.TASK_CREATED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

The following event is raised by the createBatch() API:

- CREATE_BATCH.BATCH_CREATED

The following event is raised by the cancelTask() API:

- CANCEL_TASK.TASK_CANCELED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

Upon completion of task the confirmation is sent back to the Sterling WMS.

APIs Involved

- registerTaskCompletion()
- registerBatchCompletion()
- changeTask()

Events Raised

The following events are raised by the registerTaskCompletion() and registerBatchCompletion() APIs:

- COMPLETE_TASK.TASK_COMPLETED
- COMPLETE_BATCH.BATCH_COMPLETED

The following events are raised by the changeTask() API:

- CHANGE_TASK.TASK_CHANGED
- CHANGE_TASK.TASK_PUT_ON_HOLD
- CHANGE_TASK.TASK_RELEASED_FROM_HOLD

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.6 Integrating with Inbound Sorter

Inbound Sorters are typically used when expected LPN information is available on WMS.

The Sterling WMS integrates with the inbound sorters as follows:

1. A shipment/ASN captures expected quantities. User indicates start of receipt of the ASN when container/truck pulls into the dock door. Information for the ASN is sent to sorter system along with lane sorting information, if applicable.

APIs Involved

- startReceipt()
- getShipmentDetails()
- getActivityDemand()

Events Raised

The following event is raised by the startReceipt() API:

- START_RECEIPT.ON_START_RECEIPT

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

2. LPNs are sorted to respective destination zones based on QC profiling and product characteristics.
3. The Sterling WMS is notified when LPN reaches destination.

APIs Involved

- receiveOrder()

Events Raised

The following events are raised by the receiveOrder() API:

- RECEIVE_RECEIPT.ON_SUCCESS
- RECEIVE_RECEIPT.ON_SKU_RECEIPT
- RECEIVE_RECEIPT.ON_CASE_RECEIPT
- RECEIVE_RECEIPT.ON_PALLET_RECEIPT
- RECEIVE_ORDER.INVENTORY_COST_CHANGE
- RECEIVE_ORDER.INVENTORY_COST_WRITEOFF
- RECEIVE_ORDER.INVENTORY_VALUE_CHANGE

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

4. Putaway task is automatically generated on the Sterling WMS.

7.7 Integrating with Pack Sorter

Pack sorters are used when loose items are picked and need to be sent to pack stations.

The Sterling WMS integrates with pack sorters as follows:

1. A tag indicating the shipment is associated with the pick before placing on the conveyor system.
2. Data is published to sorter on wave release with association of shipment to a pack location.

APIs Involved

- releaseWave()
- getShipmentDetails()

Events Raised

The following events are raised by the releaseWave() API:

- RELEASE_WAVE.ON_SUCCESS
- RELEASE_WAVE.SHORTAGES_DETECTED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

3. Information from outbound sorter regarding cartons diverted or quantity diverted can update a status value in the pipeline.

APIs Involved

- changeShipmentContainer()
- changeShipmentStatus()

Events Raised

The following events are raised by the changeShipmentContainer() API:

- CHANGE_CONTAINER.ON_SUCCESS
- CHANGE_CONTAINER_STATUS.ON_SUCCESS

The following event is raised by the changeShipmentStatus() API:

- CHANGE_SHIPMENT_STATUS.ON_SUCCESS

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.8 Integrating with Shipping Sorter

Outbound Sorters are typically used during high volume pick, pack ship operations.

The Sterling WMS integrates with outbound sorters as follows:

1. For pre-pick containerization, carton level information is sent after wave release. For loose items, data interfaced after post-pick containerization is completed.
2. Wave release level information is sent to sorter containing lane information.

APIs Involved

- releaseWave()
- getShipmentDetails()

Events Raised

The following events are raised by the releaseWave() API:

- RELEASE_WAVE.ON_SUCCESS
- RELEASE_WAVE.SHORTAGES_DETECTED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

3. Information from outbound sorter regarding cartons diverted or quantity diverted can update a status value in the pipeline.

APIs Involved

- changeShipmentContainer()

Events Raised

The following events are raised by the changeShipmentContainer() API:

- CHANGE_CONTAINER.ON_SUCCESS
- CHANGE_CONTAINER_STATUS.ON_SUCCESS

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.9 Integrating with Cube-a-Scan

Cube-a-scan is typically used during inbound operations to determine the dimensions or properties of an item/SKU.

The Sterling WMS integrates with cube-a-scan by updating the item details in the Sterling WMS.

APIs Involved

- manageItem()

Events Raised

The following events are raised by the manageItem() API:

- ITEM_DEFINITION.AFTER_MODIFY_ITEM
- ITEM_DEFINITION.AFTER_DELETE_ITEM

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

7.10 Integrating with Weighing Scale

A Weighing Scale is an equipment that returns the weight of a container placed on it. Weighing scales are typically used in manifest stations for parcel shipments. For more information about setting up a weighing scale, see the *Sterling Warehouse Management System: Configuration Guide*.

7.10.1 Integrating with Mettler Toledo Weighing Scales

The Sterling WMS supports out-of-the-box integration with the Mettler Toledo PS Weighing Scale, which is compatible with various shipping systems including UPS, and FedEx.

For more information about installing the Mettler Toledo Weighing Scale, see the *Selling and Fulfillment Foundation: Installation Guide*.

For more information about configuring the Mettler Toledo Weighing Scale on the Sterling WMS, see the *Sterling Warehouse Management System: Configuration Guide*.

7.10.2 Integrating with Other Weighing Scales

Additional weighing scale connectors can be built by implementing the `YCPWeighingScaleConnector` interface available in the package `com.yantra.ycp.ui.io` in the Java Archive File `platform_afc.jar`.

The following is a sample code for implementing the `YCPWeighingScaleConnector` interface:

Example 7–1 Sample Code for Implementing YCPWeighingScale Interface

```
public class CustomScaleConnector implements YCPWeighingScaleConnector {

    private YFCSerialIO sio;
    /* This assumes that the weighing scale is connected through serial port.
    You will need to write custom code to support other ports such as USB.*/
    private YFCPortConfig config;

    public CustomScaleConnector() {
    }

    public void init(YFCElement configEle) {
        sio = new YFCSerialIO();
        String portId = configEle.getAttribute("PortId");
        config = new YFCPortConfig(PortId);
    }

    public double getWeight() {
        sio.openConnection(config);
        sio.write("W"); // command to get weight from the scale
        sio.waitForResponse(20, 1000); // sleep 20ms. every time and timeout out
        after 1 sec.
        String response = sio.read();
        return processResponse(response);
    }

    private double processResponse(String response) {
        double weight = -1;
        // process the response appropriately
        return weight;
    }

    public void resetScale() {
        // send reset command if required
    }
}
```

During initialization, the `init` method is called once by the `YCPWeighingFactory` interface.

At `init` time, a config XML is passed to the `CustomScaleConnector`. This XML is stored in the Selling and Fulfillment Foundation config database (in Device Configuration) with the class name `CustomScaleConnector`.

The config XML format used for the Mettler Toledo Weighing Scale is as follows:

```
<DeviceParamsXML>
        <Attributes>
            <Attribute Name="ClassName" Value="" />
            <Attribute Name="PortId" Value="" />
            <Attribute Name="BaudRate" Value="" />
            <Attribute Name="DataBits" Value="" />
            <Attribute Name="StopBits" Value="" />
            <Attribute Name="Parity" Value="" />
            <Attribute Name="FlowIn" Value="" />
            <Attribute Name="FlowOut" Value="" />
            <!-- other extended attributes specific to weighing scale
connector implementations -->
            <Attribute Name="" Value="" />
        </Attributes>
    </DeviceParamsXML>
```

The config XML can be configured using the Device Configuration of Type ‘Weighing Scale’ in the Applications Manager. For more information see the *Sterling Warehouse Management System: Configuration Guide*.

NOTE: The implementation of the `YCPWeighingFactory` interface must ensure that an instance can be reused across invocations. The `YCPWeighingFactory` interface calls `init` once during initialization, and subsequently reuses the initialized instance.

For more details about integrating the Sterling WMS with other weighing scales, see Java Doc referring to the `com.yantra.ycp.ui.io` package.

8

Integrating with Enterprise Resource Planning Systems

An Enterprise Resource Planning (ERP) system is a packaged business software system that allows a company to automate and integrate the majority of its business processes. This enables the company to share common data and practices across the entire enterprise, and to produce and access information in a real-time environment.

The Sterling WMS can integrate with an ERP system to utilize any additional functions that are available in the existing environment.

For example, the Sterling WMS can integrate with an ERP system to enable users to:

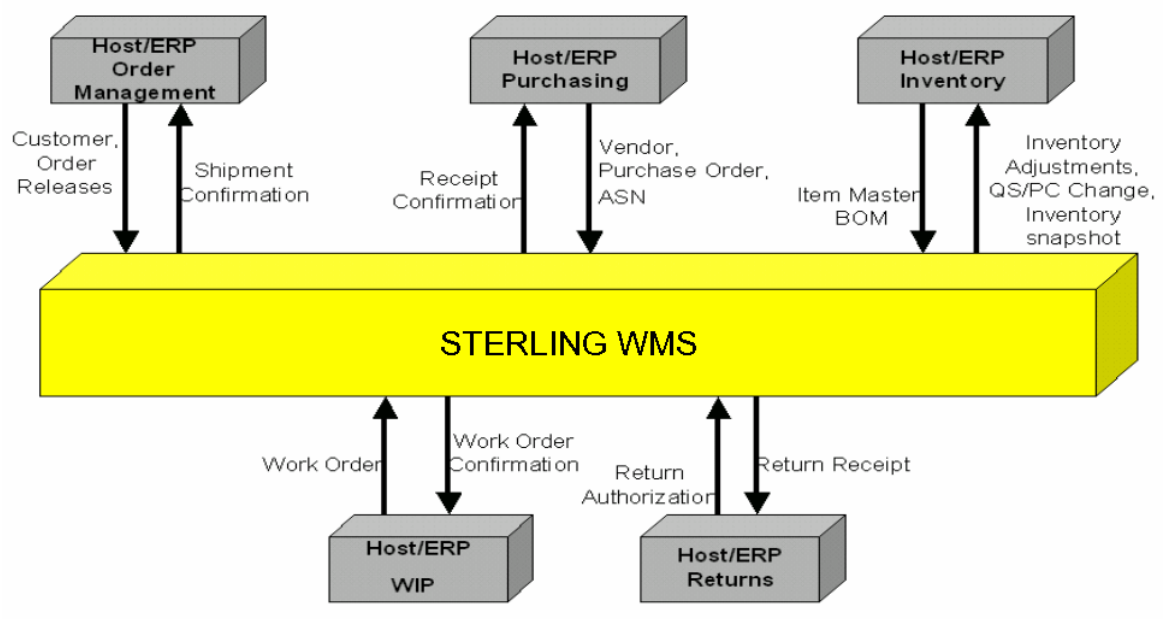
- Enter information in one system and ensure the accessibility and accuracy of the same information across the other application, if necessary, without duplication of data entry.
- Maintain the data entry and ownership at one point, the source module. Synchronize reference (common) data based on the static or dynamic nature of the data, and/or, as deemed necessary in a business environment.
- Perform the necessary business functions involving data sharing and transfer without having to be aware of the system links, the transfer mechanism and the programming details.
- Define and maintain the implementation setup of the integration to suit specific business needs. Typically, the user-definable parameters correspond to the modules installed, the active interfaces, frequency of data synchronization and real time or batch data transfer options.

8.1 Integration Overview

The Sterling WMS can be integrated with one or more of the following components of an ERP system:

- Order Management
- Purchasing
- Inventory
- WIP
- Returns

8.2 Integration Data Flow Diagram



8.3 Integration Protocol

8.3.1 Data exchange from an ERP System to the Sterling WMS

Selling and Fulfillment Foundation provides APIs to integrate the Sterling WMS with ERP applications, and transfer data from an ERP system to the Sterling WMS. These APIs can be invoked from the Service Definition Framework.

Data exchange from an ERP application to the Sterling WMS can be carried out using the Service Definition Framework in two modes:

- Asynchronous Mode (DB, JMS, MSMQ)
- Synchronous Mode (HTTP, EJB, LOCAL)

For more information about configuring these modes to facilitate integration, see the Programming Transactions chapter in the *Selling and Fulfillment Foundation: Extending Transactions Guide* .

8.3.2 Data exchange from the Sterling WMS to an ERP System

The Selling and Fulfillment Foundation APIs raise Events, which can be configured to transfer data from the Sterling WMS to an ERP application.

For more information about configuring Events, see the Programming Transactions chapter in the *Selling and Fulfillment Foundation: Extending Transactions Guide* .

8.4 Integration Specification Details

8.4.1 ERP Integration – Order Management

The Sterling WMS can be integrated with an ERP system to exchange the following information:

- Customer profile from an ERP system to the Sterling WMS
- Shipment or order release from an ERP system to the Sterling WMS
- Shipment confirmation back to an ERP system from the Sterling WMS

8.4.1.1 Customer Download from an ERP System to the Sterling WMS

Vendor information is downloaded from an ERP system to the Sterling WMS.

APIs Involved

- manageCustomer()

For more information about APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.1.2 Shipment/Order Release Download from an ERP System to the Sterling WMS

Order releases or Shipment requests are downloaded from an ERP system to the Sterling WMS.

APIs Involved

- createShipment()
- consolidateToShipment()

For more information about APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.1.3 Shipment Confirmation Upload from the Sterling WMS to an ERP System

Order releases or Shipment requests are uploaded from the Sterling WMS to an ERP system.

APIs Involved

- confirmShipment()

Events Raised

The following events are raised by the confirmShipment() API:

- CONFIRM_SHIPMENT.ON_SUCCESS
- CREATE_CONFIRM_SHIPMENT.ON_SUCCESS
- SHIP_SHIPMENT.ON_SHIP_CONFIRM_POST_VOID

- SHIP_ORDER.ON_SHIP_CONFIRM_POST_VOID
- INVENTORY_CHANGE.ON_CHANGE
- INVENTORY_COST_CHANGE.INVENTORY_VALUE_CHANGE

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.2 ERP Integration – Purchasing

The Sterling WMS can be integrated with an ERP system to exchange the following information:

- Vendor profile from an ERP system to the Sterling WMS
- Purchase Order information from an ERP system to the Sterling WMS
- Purchase Order closure information from an ERP system to the Sterling WMS
- ASN information from an ERP system to the Sterling WMS
- Receipt information sent back from the Sterling WMS to an ERP system

8.4.2.1 Vendor Download from an ERP System to the Sterling WMS

Vendor information is downloaded from an ERP system to the Sterling WMS.

APIs Involved

- manageVendor()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.2.2 Purchase Order Download from an ERP System to the Sterling WMS

Purchase Orders are created on an ERP system and downloaded to the Sterling WMS. PO modifications are also downloaded to the Sterling WMS.

APIs Involved

- createOrder()
- changeOrder()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.2.3 Purchase Order Closure Download from an ERP System to the Sterling WMS

When a PO or PO line is closed on an ERP system, it is downloaded to the Sterling WMS.

APIs Involved

- shortOrder()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.2.4 ASN Download from an ERP System to the Sterling WMS

When an ASN is created on an an ERP system, it can be downloaded to the Sterling WMS.

APIs Involved

- confirmShipment()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.2.5 Receipt Upload from the Sterling WMS to an ERP System

Receipt information can be uploaded as and when a receipt is made or when a receipt is closed.

APIs Involved

- closeReceipt() or
- receiveOrder()

Events Raised

The following events are raised by the closeReceipt() API:

- RECEIPT_COMPLETE.ON_RECEIPT_COMPLETE

The following events are raised by the receiveOrder() API:

- RECEIVE_RECEIPT.ON_SUCCESS
- RECEIVE_RECEIPT.ON_SKU_RECEIPT
- RECEIVE_RECEIPT.ON_CASE_RECEIPT
- RECEIVE_RECEIPT.ON_PALLET_RECEIPT
- INVENTORY_COST_CHANGE.INVENTORY_COST_CHANGE
- RECEIVE_ORDER.INVENTORY_COST_WRITEOFF
- RECEIVE_ORDER.INVENTORY_VALUE_CHANGE

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.3 ERP Integration - Inventory

The Sterling WMS can be integrated with an ERP system to exchange the following information:

- Item information from an ERP system to the Sterling WMS
- Item information sent from the Sterling WMS to an ERP system
- Inventory modification information sent from the Sterling WMS to an ERP system
- Inventory snapshot information sent from the Sterling WMS to an ERP system

8.4.3.1 Item Download from an ERP System to the Sterling WMS

New items are created on an ERP system and then downloaded to the Sterling WMS. Typically, the ERP system is the master. However, several attributes of items required for warehouse operations are maintained in the WMS after the download of item information from the ERP system.

APIs Involved

- manageItem()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.3.2 Item Attributes Upload from the Sterling WMS to an ERP System

Some of the item attributes, such as item dimensions and weight, can be maintained in the Sterling WMS and then uploaded to an ERP system.

APIs Involved

- manageItem()

Events Raised

The following events are raised by the manageItem() API:

- ITEM_DEFINITION.AFTER_MODIFY_ITEM
- ITEM_DEFINITION.AFTER_DELETE_ITEM

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.3.3 Inventory Change Upload from the Sterling WMS to an ERP System

Inventory changes from the Sterling WMS are uploaded to an ERP system.

APIs Involved

- adjustInventory()

Events Raised

The following events are raised by the adjustInventory() API:

- INVENTORY_CHANGE.INVENTORY_CHANGE
- INVENTORY_CHANGE.SUPPLY_CHANGE
- INVENTORY_COST_CHANGE.INVENTORY_VALUE_CHANGE

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.3.4 Inventory Snapshot Upload from the Sterling WMS to an ERP System

Inventory snapshot information may need to be uploaded from the Sterling WMS to an ERP system.

APIs Involved

- getInventoryMismatch()
- getInventorySnapshot()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.4 ERP System Integration - WIP

The Sterling WMS can be integrated with an ERP system to exchange the following information:

- Bill of Materials (BOM) information from an ERP system to the Sterling WMS
- Work Order information from an ERP system to the Sterling WMS
- Manually created work order information sent from the Sterling WMS to an ERP system
- Work Order confirmation information sent from the Sterling WMS to an ERP system
- Work Order closure information sent from the Sterling WMS to an ERP system

8.4.4.1 BOM Download from an ERP System to the Sterling WMS

Bill of Materials (BOM) information can be maintained on an ERP system and downloaded to the Sterling WMS.

APIs Involved

- manageItem()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.4.2 Work Order Download from an ERP System to the Sterling WMS

Work Orders can be downloaded from an ERP system to the Sterling WMS for execution.

APIs Involved

- createWorkOrder()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.4.3 Work Order Demand Upload for Manually Created Work Orders from the Sterling WMS to ERP

When work orders are created manually in the Sterling WMS, work order information needs to be uploaded to an ERP system so that component items are allocated on the ERP system.

APIs Involved

- createWorkOrder()
- cancelWorkOrder()
- modifyWorkOrder()

Events Raised

The following events are raised by the createWorkOrder() API:

- CREATE_WORK_ORDER.ON_SUCCESS

The following events are raised by the cancelWorkOrder() API:

- CANCEL_WORK_ORDER.ON_SUCCESS
- CANCEL_WORK_ORDER.WORK_ORDER_ACTIVITIES_COMPLETED

The following events are raised by the modifyWorkOrder() API:

- MODIFY_WORK_ORDER.ON_SUCCESS

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.4.4 Work Order Confirmation Upload from the Sterling WMS to an ERP System

When a Work Order is confirmed, information needs to be uploaded to the ERP system indicating quantity of work order confirmed or built.

With some ERP systems, this data may not be uploaded as and when quantity built. Instead, only work order closure is uploaded to the ERP system, indicating total quantity built for the work order.

APIs Involved

- confirmWorkOrderActivity()

Events Raised

The following events are raised by the confirmWorkOrderActivity() API:

- CONFIRM_WORK_ORDER.ON_SUCCESS
- CONFIRM_WORK_ORDER.WORK_ORDER_ACTIVITIES_COMPLETED
- CONFIRM_WORK_ORDER.LPN_ACTIVITIES_COMPLETED
- CONFIRM_WORK_ORDER.SKU_ACTIVITIES_COMPLETED
- CONFIRM_WORK_ORDER.SNO_ACTIVITIES_COMPLETED

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.4.5 Close Work Order from the Sterling WMS to an ERP System

When all quantities for a work order is completed or the remaining quantity is canceled, data needs to be published to the ERP system indicating that work order is complete.

APIs Involved

- changeWorkOrderStatus()

Events Raised

The following events are raised by the changeWorkOrderStatus() API:

- CHANGE_WORK_ORDER_STATUS.ON_SUCCESS

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.5 ERP Integration – Returns

The Sterling WMS can be integrated with an ERP system to exchange the following information:

- Return Order information from an ERP system to the Sterling WMS
- Return Order closure information from an ERP system to the Sterling WMS
- Receipt information sent back from the Sterling WMS to an ERP system

8.4.5.1 Return Order Download from ERP to the Sterling WMS

Return Orders are created on an ERP system and downloaded to the Sterling WMS. Return Order modifications are also downloaded to the Sterling WMS.

APIs Involved

- createOrder()
- changeOrder()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.5.2 Return Order Closure Download from an ERP System to the Sterling WMS

When a return is closed on the host system, it is downloaded to the Sterling WMS. Typically, one return is one receipt. Hence, when a receipt is closed, return may be marked as Closed without a separate integration from host system.

APIs Involved

- shortOrder()

For more information about the APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

8.4.5.3 Receipt Upload from the Sterling WMS to an ERP System

Typically, return information is uploaded only when receipt is closed.

APIs Involved

- closeReceipt() or
- receiveOrder()

Events Raised

The following events are raised by the closeReceipt() API:

- RECEIPT_COMPLETE.ON_RECEIPT_COMPLETE

The following events are raised by the receiveOrder() API:

- RECEIVE_RECEIPT.ON_SUCCESS
- RECEIVE_RECEIPT.ON_SKU_RECEIPT
- RECEIVE_RECEIPT.ON_CASE_RECEIPT
- RECEIVE_RECEIPT.ON_PALLET_RECEIPT
- RECEIVE_ORDER.INVENTORY_COST_CHANGE
- RECEIVE_ORDER.INVENTORY_COST_WRITEOFF
- RECEIVE_ORDER.INVENTORY_VALUE_CHANGE

9

Integrating with Point of Sale Systems

Selling and Fulfillment Foundation enables you to integrate with point of sale systems used in stores for product check-outs and returns from customers. When a sales transaction is posted to the Sterling WMS from a point of sale (POS), the location from which inventory has to be deducted may not be known, and hence not passed. Under such circumstances, the Sterling WMS deducts the inventory from one or more locations that are configured for the purpose of adjustment (that is, for an Adjustment Reason Code). Depending on the availability at each location, the location is appropriately adjusted and then the next location is considered, if required. If a virtual location is one of the locations in the sequence, the inventory availability at the location is not checked and such a location is allowed to go negative.

For more information about the Sterling Warehouse Management System, see the *Sterling Warehouse Management System: Concepts Guide*.

This chapter describes how Selling and Fulfillment Foundation provides integration with the point-of-sale systems.

9.1 API Invoked During Point of Sale Integration

The API invoked during the integration of the Sterling WMS with Point Of Sale Systems is adjustLocationInventory().

This API adjusts location inventory. In point of sale systems, it is typically called with an inventory reason code associated with an adjustment sequence, without a Location ID. It can also be called with both the Location ID and the inventory reason code associated with an adjustment sequence. The transaction does not go through if the Location ID is not

passed and the inventory reason code passed does not have an adjustment sequence associated with it.

If the adjustLocationInventory API is called with an inventory reason code associated with an adjustment sequence and the Location ID is not passed:

- Inventory is deducted consecutively from the locations or zones specified in the adjustment sequence.
- Within a zone, inventory is deducted according to the pick sequence of the locations in the zone. For locations having the same pick sequence number, inventory is deducted in the alphabetical order of the Location ID.
- Inventory in non-virtual locations is deducted only to the extent of the available quantity of loose SKU (inventory in LPN is not considered). Available inventory is deducted consecutively from the configured locations until a virtual location, if configured in the adjustment sequence, is reached. The balance of the demanded quantity is then adjusted from this virtual location. If any other locations have been configured in the adjustment sequence after the virtual location, they are ignored.
- The transaction does not go through if there is insufficient inventory in the locations or zones specified in the adjustment sequence and a virtual location has not been configured in the adjustment sequence.

When the adjustLocationInventory API is called with a Location ID and an inventory reason code associated with an adjustment sequence, the inventory is adjusted in the specified location and the adjustment sequence is ignored. The transaction does not go through if there is insufficient inventory at the specified location.

When the adjustLocationInventory API is called for serialized items, the location sequence associated with an inventory reason code is always ignored.

- If the adjustLocationInventory API is called with a Location ID, inventory is deducted from that location. The transaction does not go through if the serial number is not found in the specified location.
- If the adjustLocationInventory API is called without a Location ID, inventory is deducted from any location where the serial number is

found. The transaction does not go through if the specified serial number is not found in any location of the node.

10

Integrating User and Item Data with External Systems

Selling and Fulfillment Foundation enables you to integrate with external systems used to sell products, through multiple channels. This integration enables information on orders, availability, products, and customers to be passed between the external system and Selling and Fulfillment Foundation.

You can trigger synchronization of this data three ways: near real-time, on-demand, and batch. For more information about triggering methods for data synchronization, see Table 10–1, "Methods of Triggering Data Synchronization".

Table 10–1 Methods of Triggering Data Synchronization

Method	Description
Near real-time	Changes are communicated to the appropriate system as soon as they are processed. **Note:** This is applicable to both user and product synchronization.
On-demand	Occurs as a result of a customer manually triggering the synchronization from an external system.
Batch	Occurs at a specified time and automatically determines which items or customers need to be synchronized.

10.1 Order Management

This section describes how Selling and Fulfillment Foundation order management integrates with external systems.

This integration enables the following:

- Order integration — Orders placed in the external system can be tracked and maintained in Selling and Fulfillment Foundation.
- Order details — When order details are viewed in the external system, they are retrieved in real time from Selling and Fulfillment Foundation.
- Order change and cancellation - Details about order changes or cancellations are communicated between systems.

10.1.1 APIs Invoked During Order Management Integration

The following APIs are invoked during order management integration:

- createOrder()
- changeOrder()
- getSalesOrderDetails()

For more information about these APIs, see the *Selling and Fulfillment Foundation: Javadocs*.

10.2 User and Item Synchronization

This section describes how Selling and Fulfillment Foundation synchronizes user and item data with an external system.

This synchronization enables you to integrate the following:

- Users - User synchronization involves synchronizing a defined set of users, including details such as address and payment information.
- Items - Item synchronization involves synchronizing all the relevant item information.

For both users and items, services are provided to send and receive changes. These services are:

- SendItemChanges
- ReceiveItemChanges
- SendCustomerChanges
- ReceiveCustomerChanges

These services function by either placing or retrieving information from a JMS queue, and then passing this information to an internal or external API or service.

10.2.1 Item Synchronization Services in Selling and Fulfillment Foundation

Selling and Fulfillment Foundation has two main services for the synchronization of items. These services leverage APIs as well as other services in order to send or receive changes to items.

10.2.1.1 SendItemChanges Service

The sendItemChanges service is used to relay changes made to items in Selling and Fulfillment Foundation to the external system. This service is triggered as soon as an update or change to an item is made. For more information about the process flow of the sendItemChanges service, see Figure 10–1.

Figure 10–1 The SendItemChanges Service

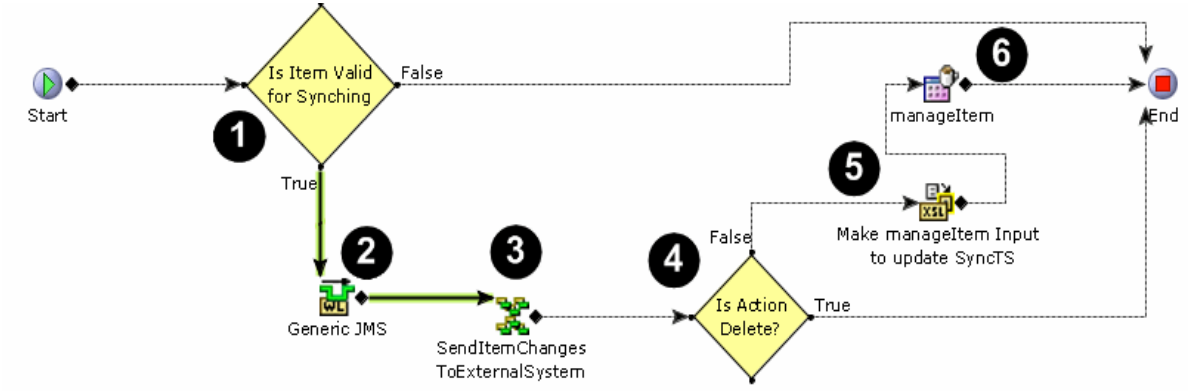


Table 10–2 The SendItemChanges Service

Step	Description
1 Is the item valid for synching?	If the item is valid for synchronization, the service continues; if it is not, the service ends. Items are deemed valid for synchronization if the ItemGroupCode is equal to PROD and the item is not a dynamic physical kit or a logical kit.
2 Generic JMS queue	The Generic JMS queue stores messages until they can continue through the service. **Note:** When configuring the SendItemChanges service, the Provider URL for both the JMS Sender and JMS Receiver must be manually configured. The queue name for both must also be set to `ItemSyncQueue`. For more information about configuring services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*
3 sendItemChangesTOExternalSystem	This service contains modules that provide an XSL translation to create a common XML file for the item, and send the XML to the external system. For more information about the common XML file, see Section 10.3. **Note:** The Java class name for the external client must be specified in the sendItemChangesTOExternalSystem service's API component.
4 Is the action a delete?	If the change being made to the item is deletion, the service ends. If it is not, the service continues.
5 Make manageItem input to SyncTS	An XSL translation takes place which adds a timestamp for when the synchronization took place. **Note:** SyncTS is the only column change that can occur in this XSL.
6 manageItem API	The item XML is passed to the manageItem API which commits the changes to Selling and Fulfillment Foundation.

Note: For more information about the SendItemChanges service, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

10.2.1.2 ReceiveItemChanges Service

The receiveItemChanges service accepts changes made to items in the external system and commits them to Selling and Fulfillment Foundation, if running in near-real-time mode. If batch mode is used, the service is called after the item synchronization cron job is run. For more

information about the process flow of the receiveItemChanges service, see Figure 10–2.

Figure 10–2 The receiveItemChanges Service

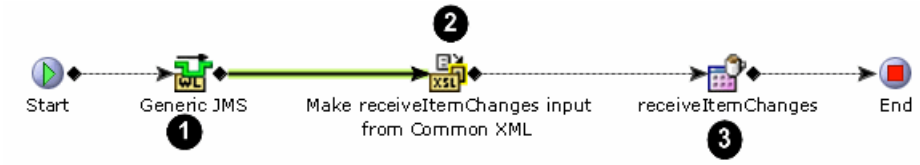


Table 10–3 The ReceiveItemChanges Service Explained

Step	Description
1 Generic JMS queue	The Generic JMS queue stores messages until they can continue through the service. **Note:** When configuring the ReceiveItemChanges service, the Provider URL for the JMS Receiver must be manually configured. The queue name must also be set to `ItemSyncReceiveItemChangesQueue`. For more information about configuring services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*
2 Make receiveItemChanges input from common XML	The XSL translation takes the common XML from the external system and removes all synchronization related data as well as transforms the XML into a format that can be read by Selling and Fulfillment Foundation. **Note:** By default, items are deleted from Selling and Fulfillment Foundation when a message for deletion is received from the external system. This can be avoided by modifying the XSL translator in this step by changing `Action="Delete"` to `Action="Modify"` and placing the item into a custom status.
3 receiveItemChanges API	The receiveItemChanges API accepts the item XML from the external system and invokes the functionality of the manageItem API.

10.2.2 Customer Synchronization Services in Selling and Fulfillment Foundation

Selling and Fulfillment Foundation has two main services for the synchronization of customers. These services leverage APIs as well as other services in order to send or receive changes to customers.

10.2.2.1 The SendCustomerChanges Service

The sendCustomerChanges service communicates changes made to customers in Selling and Fulfillment Foundation to the external system. For more information about the process flow of the sendCustomerChanges service, see Figure 10–3.

Figure 10–3 The SendCustomerChanges Service

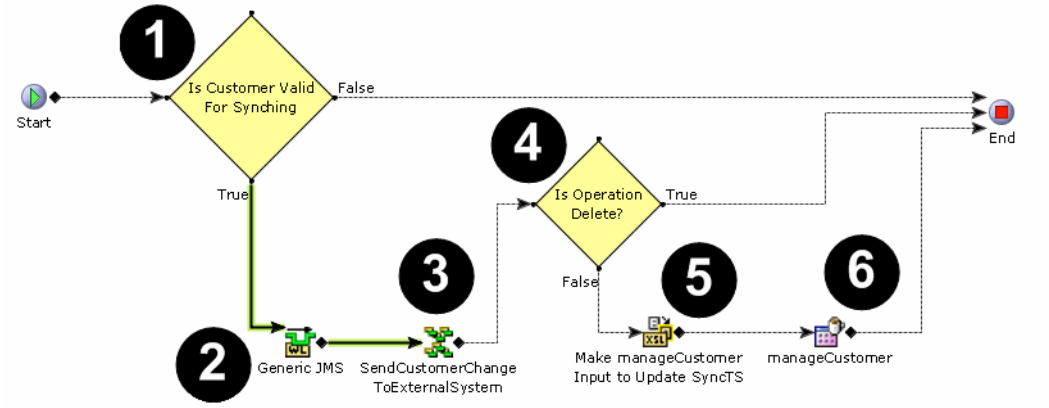


Table 10–4 The SendCustomerChanges Service Explained

Step	Description
1 Is the customer valid for synching?	If the customer is valid for synchronization, the service continues; if it is not, the service ends. Customers are deemed valid for synchronization if they are a consumer, have a user ID that is not blank, and have an IsSyncRequired flag set to 'Y'.
2 Generic JMS queue	The Generic JMS queue stores messages until they can continue through the service. **Note:** When configuring the sendCustomerChanges service, the URL for both the JMS Sender and JMS Receiver must be manually configured. The queue name for both must also be set to CustomerSyncQueue. For more information about configuring services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*
3 sendCustomerChangesTOExternalSystem	This service contains modules that provide an XSL translation to create a common XML file for the customer, and send the XML to the external system. **Note:** The Java class name for the external client must be specified in the sendCustomerChangesTOExternalSystem service's API component.
4 Is the operation a delete?	If the change being made to the customer is deletion, the service ends. If it is not, the service continues.
5 Make manageCustomer input to SyncTS	An XSL translation takes place which adds a timestamp for when the synchronization took place. **Note:** SyncTS is the only column change that can occur in this XSL.
6 manageCustomer API	The customer XML is passed to the manageCustomer API, which commits the changes to Selling and Fulfillment Foundation.

Note: For more information about the SendCustomerChanges service, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

10.2.2.2 The ReceiveCustomerChanges Service

The receiveCustomerChanges service accepts changes made to customers in the external system and commits them to Selling and

Fulfillment Foundation. This service is triggered as soon as an update or change to an customer is made. For more information about the process flow of the receiveCustomerChanges service, see Figure 10–4.

Figure 10–4 The ReceiveCustomerChanges Service

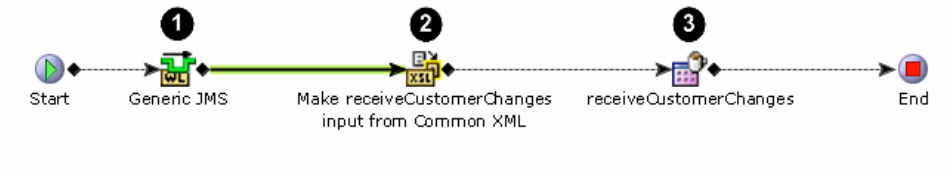


Table 10–5 The ReceiveCustomerChanges Service Explained

Step	Description
1 Generic JMS queue	The Generic JMS queue stores messages until they can continue through the service. **Note:** When configuring the sendCustomerChanges service, the URL for both the JMS Sender and JMS Receiver must be manually configured. The queue name for both must also be set to CustomerSyncQueue. For more information about configuring services, see the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*
2 Make receiveCustomerChanges input from common XML	The XSL translation takes the common XML from the external system and removes all synchronization related data as well as transforms the XML into a format that can be read by Selling and Fulfillment Foundation.
3 receiveCustomerChanges API	The receiveCustomerChanges API accepts the item XML from the external system and invokes the functionality of the manageItem API.

10.3 Customer Event Templates

Manual changes to the customer event template XML files are required to enable customer synchronization. To modify the customer event template XML files:

1. Navigate to the `<OF_INSTALL_DIR>/repository/xapi/template/merged/event/` directory.

2. Locate the CUSTOMER_DEFINITION.AFTER_CREATE_CUSTOMER.xml, CUSTOMER_DEFINITION.AFTER_DELETE_CUSTOMER.xml, and CUSTOMER_DEFINITION.AFTER_MODIFY_CUSTOMER.xml files.
3. Copy the files mentioned in Step 2 to the `<OF_INSTALL_DIR>/extensions/global/template/event` directory.
4. Modify the customer event templates listed in Step 2 to match the common XML provided in Section 10.4.1, "Customer Data Mapping".
5. Add the following attributes to the `<Customer>` element in the files mentioned in Step 2.
   - `IsSyncRequired=""`
   - `MaxModifyTS=""`
   - `SyncTS=""`

For more information about modifying template XML files, see the *Selling and Fulfillment Foundation: Customizing APIs Guide*.

10.4 Data Mapping

This section describes the mapping that takes place during the synchronization of items and customers between an external system and Selling and Fulfillment Foundation.

10.4.1 Customer Data Mapping

Table 10–6 provides mapping for the attributes related to customer synchronization from the external system and Selling and Fulfillment Foundation.

The following common XML is used to communicate with external systems:

```
<Customer OrganizationCode="" Operation="" CustomerType="">
   <CustomerContactList >
      <CustomerContact DayFaxNo="" DayPhone="" EmailID=""
EveningFaxNo=""
EveningPhone="" FirstName="" LastName="" MobilePhone=""
Title="" UserID="">
         <CustomerAdditionalAddressList Reset="y" >
            <CustomerAdditionalAddress
CustomerAdditionalAddressID="" IsShipTo=""
```

```
IsBillTo="" IsSoldTo="" IsDefaultShipTo="" IsDefaultBillTo=""
IsDefaultSoldTo="">
                  <PersonInfo AddressLine1="" AddressLine2=""
AddressLine3="" City="" Country=""
State="" ZipCode="" />
                  </CustomerAdditionalAddress>
               </CustomerAdditionalAddressList>
               <CustomerPaymentMethodList Reset="Y">
                  <CustomerPaymentMethod CreditCardExpDate=""
FirstName=""
MiddleName="" LastName="" CreditCardNo="" CreditCardType=""
PaymentType="" IsDefaultMethod="" />
               </CustomerPaymentMethodList>
         </CustomerContact>
      </CustomerContactList>
</Customer>
```

Table 10–6 Customer Data Mapping

Selling and Fulfillment Foundation Database Fields
YFS_CUSTOMER_CONTACT.USER_ID
YFS_CUSTOMER_CONTACT.LAST_NAME
YFS_CUSTOMER_CONTACT.FIRST_NAME
YFS_CUSTOMER_CONTACT.TITLE
YFS_CUSTOMER_CONTACT.EMAILID
YFS_CUSTOMER_PAYMENT_METHOD.PAYMENT_TYPE
YFS_CUSTOMER_PAYMENT_METHOD.CREDIT_CARD_NO
YFS_CUSTOMER_PAYMENT_METHOD.CREDIT_CARD_EXP_DATE
YFS_CUSTOMER_PAYMENT_METHOD.CREDIT_CARD_TYPE
YFS_CUSTOMER_PAYMENT_METHOD.FIRST_NAME
YFS_CUSTOMER_PAYMENT_METHOD.MIDDLE_NAME
YFS_CUSTOMER_PAYMENT_METHOD.LAST_NAME
YFS_CUSTOMER.ORGANIZATION_CODE
YFS_PERSON_INFO.ADDRESS_LINE1
YFS_PERSON_INFO.ADDRESS_LINE2
YFS_PERSON_INFO.ADDRESS_LINE3

Selling and Fulfillment Foundation Database Fields
YFS_PERSON_INFO.CITY
YFS_PERSON_INFO.ZIP_CODE
YFS_PERSON_INFO.STATE
YFS_PERSON_INFO.COUNTRY
YFS_CUSTOMER_ADDNL_ADDRESS.IS_SOLD_TO
YFS_CUSTOMER_ADDNL_ADDRESS.IS_SHIP_TO
YFS_CUSTOMER_ADDNL_ADDRESS.IS_BILL_TO
YFS_CUSTOMER_ADDNL_ADDRESS.IS_DEFAULT_SOLD_TO
YFS_CUSTOMER_ADDNL_ADDRESS.IS_DEFAULT_SHIP_TO
YFS_CUSTOMER_ADDNL_ADDRESS.IS_DEFAULT_BILL_TO
YFS_CUSTOMER_CONTACT.<table_name> **Note:** The <table_name> column is determined by the value of the CMGT_PHONES.PHONE_TYPE_CODE column. The customer phone number is then stored in this column.

Note: Extended attributes can be provided under the /Item/@Extn element.

10.4.2 Item Data Mapping

Table 10–7 provides the mapping for elements and attributes related to item synchronization from the external system and Selling and Fulfillment Foundation.

The following common XML is used to communicate with external systems:

```
<Item Action="Create/Modify/Delete" ItemID="" UnitOfMeasure=""
OrganizationCode="" ShortDescription="" ExtendedDescription=""
BundleFulfillmentMode="" LeadTime="" MinOrderQuantity="" IsModelItem="" Model=""
ModelItemUnitOfMeasure="" KitCode="" ConfiguredModelKey="" IsConfigurable=""
IsPreConfigured="">
            <ItemInstructionList Reset="">
               <ItemInstruction InstructionText="" SeqNo=""
InstructionType="ORDERING"/>
            </ItemInstructionList>
            <Components Reset="">
<Component ComponentItemID="" ComponentOrganizationCode=""
```

```
ComponentUnitOfMeasure="" KitQuantity=" "/>
            </Components>
</Item>
```

Table 10–7 Item Data Mapping

Attribute	Order Fulfillment Database Field	Comment
Item		
ItemID	YFS_ITEM.ITEM_ID	
UnitOfMeasure	YFS_ITEM.UOM	**Note:** The values in this field must be manually kept in synch between the two applications.
OrganizationCode	YFS_ITEM.ORGANIZATION_CODE	Assume that the catalog is maintained at the hub level.
ShortDescription	YFS_ITEM.SHORT_DESCRIPTION	This field is required to avoid errors.
Extended Description	YFS_ITEM.EXTENDED_DESCRIPTION	
BundleFulfillmentMode	YFS_ITEM.BUNDLE_FULFILLMENT_MODE	This value should be based on the following: ▫ "01" for ShipTogether when the configurable item is a non-container only item. ▫ "02" for Ship Independently when the configurable item is a container only item.
LeadTime	YFS_ITEM.LEAD_TIME	
MinOrderQuantity	YFS_ITEM.MIN_ORDER_QUANTITY	
Model	YFS_ITEM.MODEL	The existing MODEL field is used to store the parent SKU to represent the aggregate item. Do not confuse with CONFIGURED_MODEL_KEY of the configurable item.
ModelItemUnitOfMeasure	YFS_ITEM.MODEL_ITEM_UOM	This field stores the unit of measure of the parent SKU.
IsModelItem	YFS_ITEM.IS_MODEL_ITEM	The value stored in this field should be "Y" if the product is of the type "Aggregate".

Attribute	Order Fulfillment Database Field	Comment
KitCode	YFS_ITEM.KIT_CODE	The value for this field is based on the following: ▫ "PK" if the product is of the type "ASSEMBLY" and the CMGT_PRODUCT.COMPONENT_SUB_TYPE field indicates that the product is a physical kit. ▫ "BUNDLE" if the product is of the type "ASSEMBLY" and the CMGT_PRODUCT.COMPONENT_SUB_TYPE field indicates that the product is a bundle. ▫ "BUNDLE" if product is of the type "CONFIGURABLE"
ConfiguredModelKey	YFS_ITEM.CONFIGURED_MODEL_KEY	
IsConfigurable	YFS_ITEM.IS_CONFIGURABLE	The value of this field should be "Y" if the product is of the type "CONFIGURABLE".
IsPreConfigured	YFS_ITEM.IS_PRE_CONFIGURED	
ItemInstructionList/ItemInstruction		
InstructionText	YFS_ITEM_INSTRUCTION.INSTRUCTION_TEXT	
InstructionType	YFS_ITEM_INSTRUCTION.INSTRUCTION_TYPE	
SeqNo	YFS_ITEM_INSTRUCTION.SEQ_NO	
Components/Component		
ComponentItemID	YFS_KIT_ITEM.COMPONENT_ITEM_KEY Based on ComponentItemID,ComponentOrganizationCode, and ComponentUnitOfMeasure.	
ComponentOrganizationCode	YFS_KIT_ITEM.COMPONENT_IT Based on ComponentItemID, ComponentOrganizationCode, ComponentUnitOfMeasure	Assume that the catalog is maintained at the hub level.
ComponentUnitOfMeasure	YFS_KIT_ITEM.COMPONENT_ITEM_KEY Based on ComponentItemID, ComponentOrganizationCode, ComponentUnitOfMeasure	Unit of measure of the config line (configurable) or part (assembly)
KitQuantity	YFS_KIT_ITEM.KIT_QUANTITY	

Note: Extended attributes can be provided under the /Item/@Extn element.

Note: Product item statuses must be manually kept in sync between the external system and Selling and Fulfillment Foundation.

There are two scenarios in which statuses are updated during product item synchronization:

- A new product item is added to either the external system or Selling and Fulfillment Foundation. During synchronization, the product item is added, and the status updated to Held in Selling and Fulfillment Foundation or In Creation in the external system.
- If a product item is deleted in Selling and Fulfillment Foundation, the status in the external system is updated to Blocked. In addition, when that product item is retrieved in the UI of the external system, text is displayed in the UI to indicate that this product has been deleted in Selling and Fulfillment Foundation.
- If a product is deleted from the external system, a message is sent to Selling and Fulfillment Foundation. By default, items are removed from the database. This can be avoided by modifying the XSL translator in this step by changing `Action="Delete"` to `Action="Modify"` and placing the item into a custom status.

11

Integrating with JMS Systems

In order for some service nodes to communicate with external applications, external message queueing software must be configured. This appendix explains how to configure the following third-party message queueing applications:

- BEA WebLogic JMS
- IBM WebSphere MQ
- IBM WebSphere Default Messaging
- JBoss Messaging JMS
- TIBCO JMS

11.1 BEA WebLogic JMS

This section explains how to configure BEA WebLogic JMS as the messaging system for Selling and Fulfillment Foundation. For information specific to using WebLogic, see the documentation provided by BEA.

11.1.1 Configuring WebLogic JMS

To configure WebLogic JMS:

1. Invoke the WebLogic console by entering the URL for Application Consoles. For example, `http://<IP address of machine where weblogic is installed>:<port>/console`.
2. Log in as Administrator.
3. In the left-hand panel, click Services > JDBC > Connection Pools.

4. If message persistence or paging is required, right-click Connection Pools and choose configure a new JDBCConnectionPool.
5. Configure the new JDBC pool with the following values:
   - Name - Any name, for example, `MyJDBCPool`
   - URL - `jdbc:oracle:thin:@<IPAddress>:1521:<SID>`
   - DriverClassName - `oracle.jdbc.OracleDriver`
   - Properties -
     * user=<username>
     * password=<password>
6. Select the Targets tab. In the left-hand panel, select one or more servers. (Several choices may appear if your server is in a clustered environment.) Then click the right arrow button to move the servers you have selected to the panel on the right.
7. In the left-hand panel, right-click JMS > ConnectionFactories to configure a new Connection Factory.

   The JNDIName must match the QCFlookup value in the Applications Manager for the WebLogic JMS Transport Type.
8. Select the Targets tab. In the left-hand panel, select one or more servers. (Several choices may appear if your server is in a clustered environment.) Then click the right arrow button to move the selected server to the window on the right.
9. If message persistence or paging is required, right-click Stores, and configure a new JMSJDBCStore or Filestore.
   a. If you choose JDBCStore, using the Connection Pool drop-down list, select your connection pool.
   b. Right-click Servers and configure a new JMS server.
   c. Select the store from the drop-down list.
10. Select the Targets tab. In the left-hand window, select *one* server. (Several choices may appear if your server is in a clustered environment; you can select only one of them.). Then click the right arrow button to move the selected server to the window on the right.

11. Within the newly configured JMS server, click Destinations and configure all required JMS Queues. Now all of the JMS queues are configured.

    When configuring services that use WebLogic JMS, use the JNDI Name value from the WLS configuration as the message queue name.

12. Restart the WebLogic server for these new settings to take effect.

13. Launch the integration server by running `startIntegrationServer.sh` (or `cmd`) in `<INSTALL_DIR>/bin`.

14. If you need to run multiple servers, repeat Step 13 for each additional server.

11.1.2 WebLogic Time-Out Considerations for Transacted Sessions

When using WebLogic JMS as a messaging system to receive messages in transactional mode and no messages are received for a period of time equal to the WebLogic transaction time-out value (defaults to 3600 seconds), the following error message appears in the integration server. After this error message appears, no messages can be processed and you must relaunch the adapter in order to process any messages that recently arrived.

```
<date-time> [Thread-6] ERROR services.jms.JMSConsumer  -Could not successfully
process message
weblogic.jms.common.TransactionRolledBackException:
        at weblogic.rmi.internal.BasicOutboundRequest.sendReceive
        (BasicOutboundRequest.java:85)
        at weblogic.rmi.internal.BasicRemoteRef.invoke(BasicRemoteRef.java:135)
        at weblogic.rmi.internal.ProxyStub.invoke(ProxyStub.java:35)
        at $Proxy2.dispatchSyncNoTranFuture(Unknown Source)
        at weblogic.jms.dispatcher.DispatcherWrapperState.dispatchSyncNoTran
        (DispatcherWrapperState.java:341)
        at weblogic.jms.client.JMSSession.receiveMessage(JMSSession.java:347)
        at weblogic.jms.client.JMSConsumer.receive(JMSConsumer.java:333)
        at weblogic.jms.client.JMSConsumer.receive(JMSConsumer.java:279)
        at com.yantra.interop.services.jms.JMSConsumer.run(JMSConsumer.java:204)
        at java.lang.Thread.run(Thread.java:512)
```

For help with choosing an appropriate transaction time-out value for your system, see your WebLogic documentation.

11.2 IBM WebSphere MQ

This section explains how to configure a service for Selling and Fulfillment Foundation using IBM WebSphere MQ as the transport. For information specific to using WebSphere MQ, see the documentation provided by IBM.

These directions assume that the following have been successfully installed:

- WebSphere MQ software
- WebSphere MQ Java classes
- WebSphere MQ JMS support pack

11.2.1 Creating the Queue Manager and Queues

To create the Queue Manager and Queues:

1. Log in as the WebSphere MQ user or as a user belonging to the mqm user group.
2. Navigate to the directory where WebSphere MQ has been installed. Typically the location is as follows:
   - If you are using UNIX - `/opt/mqm/bin`
   - If you are using Windows - `<WebSphere MQ Install Directory>\bin`
3. Run the `dspmq` command to find out which queue managers, if any, exist.
   - If a suitable queue manager exists, start it using the `strmqm <qmgr>` command. The queue manager can be stopped by using the `endmqm <qmgr>` command.
   - If no queue manager exists, use the `crtmqm <MYQMGR>` command to create one.
4. Run the `runmqsc` command to send commands for creating queues. For examples of these commands, see below:

```
runmqsc MYQMGR
DEFINE QLOCAL ('getATP');
```

```
DEFINE QLOCAL ('createOrder');
END
```

Important: WebSphere MQ converts all characters to upper case, which causes errors. To use mixed case names, enclose them within single quotation marks, for example, `DEFINE QLOCAL ('getATP')`.

11.2.2 Configuring a Queue Manager to Client Connection

In order to send messages to a WebSphere MQ queue on another computer, the QManager must be configured for the server and the client computer.

When a new queue is created in WebSphere MQ, the following default values are assigned to it:

- MAXDEPTH - Maximum number of messages that a queue can hold. Defaults to 5000.
- MAXMSGL - Maximum size of a message. Defaults to 4 MB.

These settings may need to be adjusted depending on the load and speed of the third-party application that submits the messages, as opposed to the third-party application that retrieves the messages.

To create JMS bindings:

1. On the server computer, create a QueueManager <QManagerName>.
2. On the server computer's command line, run the following executable:

   ```
   <MQInstallDir>/bin/runmqlsr -m <QManagerName> -t TCP -p <PORT>
   ```

3. On the client computer, edit the `JMSAdmin.config` properties file to contain the following lines:

   ```
   INITIAL_CONTEXT_FACTORY=<JNDI_ICF>
   PROVIDER_URL=<JNDI_URL>
   ```

   where <JNDI_ICF> is the Initial Context Factory (ICF) class for use with the JNDI you have chosen. For example, `com.sun.jndi.fscontext.RefFSContextFactory`. <JNDI_URL> is

the path of the provider URL which is provided in the format expected by the JNDI server and ICF.

4. On the client computer, create a .scp command file that contains the following parameters:

```
def qcf( <QCFName> ) qmgr(<QManagerName>) transport(CLIENT) host(<ipaddress
of Server> ) channel(SYSTEM.DEF.SVRCONN) port( <PORT> )
def q(getATP) qu(getATP)
def q(reply_getATP) qu(reply_getATP)
def q(createOrder) qu(createOrder)
end
```

5. On the client computer, pass the .scp file to the WebSphere MQ JMSAdmin class using the following syntax:

```
java com.ibm.mq.jms.admin.JMSAdmin < intsetup.scp
```

This creates a .bindings file in the directory specified for the provider URL. All the JAR files in <MQ_HOME>/java/lib/ directory should be listed in your CLASSPATH environment variable.

To remove JMS bindings:

1. To unbind the queues from JNDI, create a .scp command file and pass it into the WebSphere MQ JMSAdmin program. The following are example commands:

```
del qcf(ivtQCF)
del q('getATP')
del q('reply_getATP')
del q('createOrder')
end
```

Archive Files

Since this configuration uses the client transport, the com.ibm.mqbind.jar file is not necessary. However, the client does use the following MQ-specific JAR files:

- /mqclient/java/lib/com.ibm.mq.jar
- /mqclient/java/lib/com.ibm.mqjms.jar
- /mqclient/java/lib/connector.jar
- /mqclient/java/lib/fscontext.jar

- /mqclient/java/lib/jms.jar
- /mqclient/java/lib/jndi.jar
- /mqclient/java/lib/jta.jar
- /mqclient/java/lib/providerutil.jar

11.2.3 Configuring Selling and Fulfillment Foundation to Use WebSphere MQ Queues

When configuring Selling and Fulfillment Foundation to use the WebSphere MQ queues, see the WebSphere MQ node in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

To configure a service definition:

1. Log in to Selling and Fulfillment Foundation as the user who belongs to the mqm user group (otherwise, the WebSphere MQ adapter does not launch).
2. Use the Applications Manager to configure the service. While configuring a WebSphere MQ service, enter the following:
   - The initial Context Factory `com.sun.jndi.fscontext.RefFSContextFactory`, and
   - A provider URL as `file:/<pathOfTheProviderURL>`

   Note: The values for the Context Factory and the Provider URL must match those in the `JMSadmin.config` file.
3. Launch the integration server by running `startIntegrationServer.sh` (or `cmd`) in `<INSTALL_DIR>/bin`.
4. If you need to run multiple servers, repeat Step 3 for each additional server.

11.2.4 Accessing WebSphere MQ Using WebSphere's JNDI Namespace

You can configure the WebSphere MQ queues for access by WebSphere's JNDI namespace rather than the typical file URL. This section describes how to make that configuration.

11.2.5 Before You Begin

For information about the version of WebSphere MQ which includes MQ JMS client software, see the *Selling and Fulfillment Foundation: Installation Guide*.

If needed, see the IBM Technical Tip "*Setting up MQ Java Message Service (JMS) Support in WebSphere Application Server"*.

To configure WebSphere MQ:

1. You should set the shared library path for UNIX and LINUX systems as follows:

   `set <Shared_Library_Path_Name>=<mqjava_install_path>/lib`

   where the `<Shared_Library_Path_Name>` is the shared library path environment variable for your operating system. For example:

   - In AIX it is `LIBPATH`.
   - In HP-UX it is `SHLIB_PATH`.
   - In Sun and Linux it is `LD_LIBRARY_PATH`.

2. Modify the `<mqjava_install_path>/bin/JMSAdmin.config` file as follows:

   ```
   INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
   PROVIDER_URL=CORBAloc::///<WAS_admin_IP_address>:<WAS_bootstrap_port>
   ```

3. Create an `ivtsetup.scp` command file that contains the following parameters:

   ```
   def qcf( <QCFName> ) qmgr(<QManagerName>) transport(CLIENT) host(<ipaddress
   of Server> ) channel(SYSTEM.DEF.SVRCONN) port( <PORT> )
   def q(JNDINameOfQueue) qu(QueueName)
   ```

   In the following example, a QueueConnectionFactory is created with the JNDI name ivtQCF. This QueueConnectionFactory is configured to access the Queue Manager SYSTEM.TEST. Using the 'CLIENT' (network based) transport on the computer 127.0.0.1,through port 1414 (WebSphere MQ default), through the server connection channel named SYSTEM.DEF.SVRCONN (WebSphere MQ default).

   Next, a queue object is created with the JNDI name getATP, which is configured to work with the getATP queue on QueueManager

SYSTEM.TEST. (Of course, you must ensure that you have created this queue on the queue manager as well.)

Finally, an end command is issued to shut down JMSAdmin.

Note that the .scp file can have any name, but the convention is ivtsetup.scp (ivt=installation verification test).

```
def qcf(ivtQCF) qmgr(SYSTEM.TEST) transport(CLIENT) host(127.0.0.1)
CHANNEL(SYSTEM.DEF.SVRCONN) port (1414)

def q(getATP) qu(getATP) QMGR(SYSTEM.TEST)
end
```

4. Set the PATH and CLASSPATH in the JMSAdmin script as follows:

```
MQJAVA_PATH=<path to ma88 installation>
PATH=$MQJAVA_PATH
CLASSPATH=$MQJAVA_PATH/lib:$MQJAVA_PATH/lib/com.ibm.mq.jar:$MQJAVA_
PATH/lib/com.ibm.mqjms.jar:$MQJAVA_PATH/lib/jms.jar
```

   For information about WebSphere JARs, see IBM documentation

5. Pass the .scp file to the WebSphere MQ JMSAdmin class using the following syntax:

```
java com.ibm.mq.jms.admin.JMSAdmin < intsetup.scp
```

11.2.5.1 Inside the Applications Manager

Configure a service that contains a WebSphere MQ node. Ensure that the link properties of the node match the Initial Context Factory, Provider URL, and the JNDI name specified for the desired queue.

The WebSphere MQ and WebSphere JAR files are also required for the IntegrationAdapter program and whatever client is putting the messages into the queue(s).

11.2.5.2 Inside the WebSphere Admin Console

In order to put messages into the WebSphere MQ queues from inside Selling and Fulfillment Foundation, as the Release agent needs to do or for services invoked by Actions and Events, follow the instructions provided in the *IBM Technical Tip "Configuring MQ JMS support in the WebSphere J2EE Environment".*

If you are running on an IBM AIX system, include the following line in the script that launches the IntegrationAdapter:

```
export LDR_CNTRL=MAXDATA=0x30000000
```

11.3 IBM WebSphere Default Messaging

This section explains how to configure a service for Selling and Fulfillment Foundation using IBM WebSphere Default Messaging as the transport. For more information specific to using the WebSphere Default Messaging, see the documentation provided by IBM. These directions assume that the following have been successfully installed:

- WebSphere Application Server with support for Default Messaging
- WebSphere Default Messaging Java classes
- WebSphere Default Messaging support pack

11.3.1 Configuring Selling and Fulfillment Foundation to Use WebSphere Default Messaging

When configuring Selling and Fulfillment Foundation to use the WebSphere Default Messaging queues, see the WebSphere Default Messaging Queue section in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*

11.3.2 Before you Begin

For information about the version of WebSphere Default Messaging that includes the Default Messaging client software, see the *Selling and Fulfillment Foundation: Installation Guide*.

To configure WebSphere Default Messaging:

1. Set the shared library path for UNIX and LINUX system as follows:

   ```
   set <Shared_Library_Path_Name>=<dm_java_install_path>/lib
   ```

   where <Shared_Library_Path_Name> is the shared library path environment variable for your operating system.

For example:

- On AIX it is: `LIBPAT`
- On Linux it is: `LD_LIBRARY_PATH`

2. If Agent or Integration Servers communicate with the Websphere Default Messaging, install the WAS client. The WAS client needs to be the exact version, including fix pack as the WAS server.
3. Add the following command to the `startIntegrationServer.sh` script prior to the java line:

   `${WAS_CLIENT_HOME}/bin/setupClient.sh`

   where `${WAS_CLIENT_HOME}` is the installation location of the WAS client.
4. Edit the `startIntegrationServer.sh` script to run the Agent or Integration Server to add the following system property:

   `-Djava.ext.dirs=$WAS_EXT_DIRS  $SERVER_ROOT $CLIENTSAS.`
5. Ensure that the following changes are made to the `startIntegrationServer.sh` (or cmd) startup script located in the `<INSTALL_DIR>/bin` directory to include the changes made in Step 3 and Step 4.

   ```
   WAS_CLIENT_HOME=<path of where WAS client installation>
   export WAS_CLIENT_HOME
   ${WAS_CLIENT_HOME}/bin/setupClient.sh
   java -Djava.ext.dirs=$WAS_EXT_DIRS  $SERVER_ROOT $CLIENTSAS
   ${BOOTCLASSPATH} ${JAVA_OPTIONS}  -cp ${CLASSPATH}
   com.yantra.integration.adapter.IntegrationAdapter "$1"
   ```

11.4 JBoss Messaging JMS

This section explains how to configure Red Hat JBoss JMS as the messaging system for Selling and Fulfillment Foundation. For information about using JBoss, see the documentation provided by Red Hat.

11.4.1 Creating Queues

This section explains how to create queues.

To create a Queue:

1. Edit the <JBOSS_HOME>/server/<SERVER_NAME>/deploy/jboss-messaging.sar/destination_service.xml file to configure a queue. Table 11–1 provides a list of attributes to use to configure a queue.

Table 11–1 JBoss JMS Attributes

Attribute	Description
DestinationManager	Specify the object name of the DestinationManager where the queue is deployed.
SecurityManager	Specify the object name of the SecurityManager where the SecurityConf is deployed.
SecurityConf	Specify the configuration interpreted by the SecurityManager.
JNDIName	Specify the JNDI binding of the queue. If you specify none, the system looks for a jmx attribute "name" in the queue's object name.
MaxDepth	Specify the maximum depth of the queue.
InMemory	When set to true, messages are not persisted. It also avoids message softening when NullPersistenceManager is used.
RedeliveryLimit	Specify the maximum number of times a message must not be acknowledged before it is sent to DLQ. Valid values are: • 0 - indicates do not redeliver • n - indicates redeliver n times
RedeliveryDelay	Specify the time (in milli seconds) to wait before a message is redelivered after it is not acknowledged.
MessageCounterHistoryDayLimit	Specify the number of days you want to keep the MessageCounter history.

Table 11–1 JBoss JMS Attributes

Attribute	Description
ReceiversImpl	Specify the class you want to use for the receivers implementation.
RecoveryRetries	Specify the recovery retries for the queue. By default, the value is set to 0 (zero). Specifies the number of times uncommitted transactions must be resolved before failing.

The following is a sample code for queue configuration:

```
<mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=testQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
<depends optional-attribute-name="ServerPeer">jboss.messaging:service=
ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
      <attribute name="SecurityConfig">
         <security>
            <role name="guest" read="true" write="true"/>
            <role name="publisher" read="true" write="true" create="false"/>
            <role name="noacc" read="false" write="false" create="false"/>
         </security>
      </attribute>
   </mbean>
```

11.4.2 Configuring Selling and Fulfillment Foundation to Use JBoss Messaging Queues

When configuring Selling and Fulfillment Foundation to use JBoss Messaging queues, see the section about the JBoss Messaging node in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Note: You must install JBoss messaging for using JBoss messaging queues. JBoss Messaging is supported with JBoss 4.2.0 or higher. However, for earlier versions of JBoss, only JBoss MQ is supported as a JMS.

To configure a service definition:

1. Log in to Selling and Fulfillment Foundation as an admin user.
2. Use the Applications Manager to configure the service. While configuring a Generic JMS service, enter the following:
   - The initial Context Factory `org.jnp.interfaces.NamingContextFactory`, and
   - A provider URL as `jnp://<IP address and port of the JBoss instance>`
3. Set up the CLASSPATH for the `startIntegrationServer` script by adding the required jars to the CLASSPATH. For more information about setting up the classpath, see the section on Setting Up the Classpath for the Runtime Utilities in the *Selling and Fulfillment Foundation: Installation Guide*.
4. Launch the integration server by running `startIntegrationServer.sh` (or `cmd`) in `<INSTALL_DIR>/bin`.
5. If you need to run multiple servers, repeat Step 4 for each additional server.

11.5 TIBCO JMS

This section explains how to configure TIBCO JMS as the messaging system for Selling and Fulfillment Foundation. For information about using TIBCO JMS, see the documentation provided by TIBCO.

11.5.1 TIBCO JMS Attributes

Table 11–2 provides a list of attributes used to create JMS objects, configure users, groups, and permissions in the TIBCO Enterprise Messaging Server using the `tibemsadmin` command.

The `tibemsadmin` command is a command line utility used to create JMS objects, and to configure user and group permissions in TIBCO. This tool is available in the `<TIBCO>/ems/5.0/bin` directory.

Table 11–2 TIBCO JMS Attributes

Attribute	Description
tibemsd	The command used to start the TIBCO server.
tibemsadmin	The command used to start the TIBCO admin tool.
create factory <Connection Factory Name> <Connection Factory Type>	The command used to create a Queue Connection Factory. Example: create factory secureqcf queue
addprop factory <Connection Factory Name> url=<url-string>	The command used to set up an URL to listen to an external address such that the Queue Connection Factory is accessible from other hosts. Example: addprop factory secureqcf url=tcp://devhost:7222
create queue <queue-name>	The command used to create a queue. Example: create queue securequeue
create user <username>	The command used to create a user. Example: create user secureuser
set password <username> <new password>	The command used to create a password for a user. Example: set password secureuser securepassword
create group <groupname>	The command used to create a group. Example: create group securegroup
add member <group name> <user to be added>	The command used to add a user to a group. Example: add member securegroup secureuser
grant queue <queuename> group=<groupname> <permission>	The command used to set the requisite permissions on a queue for a group Example: grant queue securequeue group=securegroup send grant queue securequeue group=securegroup receive grant queue securequeue group=securegroup browse
addprop queue <queuename> secure	The command used to enable authorization for a queue. Example: addprop queue securequeue secure

Configuring a JMS client on TIBCO Enterprise Messaging Server

The following parameters must be configured to connect to an unsecured queue on TIBCO:

- **URL:** tcp://<hostname>:<port>

  Example: tcp://tibserver:7222

- **ICF:** com.tibco.tibjms.naming.TibjmsInitialContextFactory
- **QCF:** The name of the Queue Connection Factory created during setup. Refer Table 11–2 for more information about the Queue Connection Factory.

If you need to connect to a secured queue, you must pass a username and password for both JNDI and "queuebased" security.

The following properties must be configured for use with the Service Definition Framework (SDF):

- sci.queuebasedsecurity.userid
- sci.queuebasedsecurity.password
- java.naming.security.principal
- java.naming.security.credentials

11.5.2 Configuring Selling and Fulfillment Foundation to use TIBCO Messaging Queues

When configuring Selling and Fulfillment Foundation to use TIBCO Messaging queues, see the section about the TIBCO Messaging node in the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

To configure a service definition:

1. Log in to Selling and Fulfillment Foundation as an admin user.
2. Use the Applications Manager to configure the service. While configuring a Generic JMS service, enter the following:
   - The initial Context Factory `com.tibco.tibjms.naming.TibjmsInitialContextFactory`, and
   - A provider URL as `tcp://<IP address and port of the TIBCO instance>`

3. Set up the CLASSPATH for the `startIntegrationServer` script by adding the required jars to the CLASSPATH. For more information about setting up the classpath, see the section on Setting Up the Classpath for the Runtime Utilities in the *Selling and Fulfillment Foundation: Installation Guide*.
4. Launch the integration server by running `startIntegrationServer.sh` (or `cmd`) in `<INSTALL_DIR>/bin`.
5. If you need to run multiple servers, repeat Step 4 for each additional server.

12

Integrating with Financial Systems

To use the data captured using the Selling and Fulfillment Foundation Inventory Cost Management feature with your financial system, you must:

- Load Initial Inventory Cost Data
- Configure Process-Specific Events

12.1 Load Initial Inventory Cost Data

Selling and Fulfillment Foundation provides an API to load the initial inventory value of an item at a ship node for a given quantity. The loadInventoryNodeCost API supports multiple items to be given in the input with inventory cost data for each ship node under that.

The loadInventoryNodeCost API validates the Quantity passed with the actual inventory supply information available for that item/ship node. This API only considers the supply types which are specified as on-hand and cost maintained. For more information about the input XML attributes, see the *Selling and Fulfillment Foundation: Javadocs*.

This API is called for the initial load of cost data at system start up time. This API should not be used after going into production with the Inventory Costing Management feature implemented.

The following query can be run to get the initial onhand supply quantity:

```
SELECT B.ORGANIZATION_CODE, B.ITEM_ID, B.UOM, B.PRODUCT_CLASS, A.SHIPNODE_KEY
SHIP_NODE, SUM(QUANTITY) QUANTITY
FROM YFS_INVENTORY_SUPPLY A,YFS_INVENTORY_ITEM B
WHERE A.INVENTORY_ITEM_KEY = B.INVENTORY_ITEM_KEY
AND SUPPLY_TYPE IN (
SELECT SUPPLY_TYPE FROM YFS_INVENTORY_SUPPLY_TYPE
```

```
WHERE ONHAND_SUPPLY = 'Y' AND COSTING_REQUIRED = 'Y')
GROUP BY B.ORGANIZATION_CODE, B.ITEM_ID, B.UOM, B.PRODUCT_CLASS, A.SHIPNODE_KEY
```

12.2 Configure Process-Specific Events

In order to interface with your financial system and use the Selling and Fulfillment Foundation Inventory Costing data, you must configure the applicable events for the following processes:

- Receipt
- Sales Order Creation
- Shipment Confirmation
- Invoice
- Work Order Confirmation
- Inventory Adjustment
- Return Order
- Callback from Financial System for Inventory Value Adjustment

12.2.1 Receipt Process

From the General Process Type, configure the following events for the INVENTORY_COST_CHANGE Transaction ID:

- INVENTORY_COST_CHANGE
- INVENTORY_COST_WRITEOFF

12.2.1.1 INVENTORY_COST_CHANGE

When is this event raised?

This event is raised for any order receipt such as a purchase order, return order and so on. For example, at the time of purchase order receipt this event is raised from the inventory management module for each receipt line containing details of a single receipt line to generate G/L level postings in a financial application. One event is published for each purchase order line as a receipt is recorded against it. If a purchase order line is received in multiple receipts, multiple events are raised.

For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries for accounts payable accruals and inventory value accounts.

12.2.1.2 INVENTORY_COST_WRITEOFF

When is this event raised?

When doing a receipt against an item or node that has a negative on-hand balance, Inventory Value and Average Cost calculations need to be modified. The application generates this second event to accompany the standard inventory cost change event (***INVENTORY_COST_CHANGE***). This second event publishes the delta between recalculated inventory value and the write off amount details. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries for variance and inventory value accounts.

12.2.2 Sales Order Creation Process

The unit cost for an order line is stored as the unit cost stored for the item master. If the unit cost was manually entered at the item level in the product master tables, the order line uses the manually entered unit cost. If no manual entry was made, the order line uses the computed unit cost stored at the item level. If no such cost was stored, the cost is reflected as $0.00 on the sales order line and the ORDER_CREATE.ON_ZERO_UNIT_COST event is triggered.

If the item definition is not stored in Selling and Fulfillment Foundation, the getItemDetails user exit may be implemented to return unit cost from an external source. For more information about the getItemDetails user exit, see the *Selling and Fulfillment Foundation: Javadocs*.

12.2.3 Shipment Confirmation Process

From the General Process Type, configure the following events for the INVENTORY_COST_CHANGE Transaction ID:

- INVENTORY_VALUE_CHANGE

12.2.3.1 INVENTORY_VALUE_CHANGE

When is this event raised?

When a sales order is shipped this event is raised from the inventory management module for each shipment line with the inventory value change information for the fulfillment location. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries for cost of goods sold, inventory value, and variance accounts.

12.2.4 Invoice Process

Using the CREATE_ORDER_INVOICE.0003 Transaction ID for returns or the CREATE_SHIPMENT_INVOICE.0001 Transaction ID for shipments, configure the following events for the Invoice process:

- ON_INVOICE_CREATION

12.2.4.1 ON_INVOICE_CREATION

When is this event raised?

During invoice creation, this event is raised for each invoice created. This event publishes the details about the invoice created. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This can be used to post sales and account receivables general ledger entries.

12.2.5 Work Order Confirmation Process

From the General Process Type, configure the following events for the INVENTORY_COST_CHANGE Transaction ID:

- INVENTORY_COST_CHANGE
- INVENTORY_COST_WRITEOFF
- INVENTORY_VALUE_CHANGE

12.2.5.1 INVENTORY_COST_CHANGE

When is this event raised?

During work order processing, when the production of a kit parent item is reported, this event is raised from the inventory management module for the parent with the inventory cost change information for the production location. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries on the financial system.

12.2.5.2 INVENTORY_COST_WRITEOFF

When is this event raised?

When reporting production of a kit parent item that has a negative on-hand balance at the production location, Inventory Value and Average Cost calculations need to be modified. The application generates this second event to accompany the standard inventory cost change event (***INVENTORY_COST_CHANGE***). This second event publishes the delta between recalculated inventory value and the write off amount details. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries on the financial system.

12.2.5.3 INVENTORY_VALUE_CHANGE

When is this event raised?

When reporting production of a kit, this event is raised for each kit component.For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update general ledger entries for cost of goods sold, inventory value, and variance accounts.

12.2.6 Inventory Adjustment Process

From the General Process Type, configure the following events for the INVENTORY_COST_CHANGE Transaction ID:

- INVENTORY_VALUE_CHANGE

12.2.6.1 INVENTORY_VALUE_CHANGE

When is this event raised?

When an inventory adjustment is done for an item at a fulfillment location this event is raised from the inventory management module with the inventory value change information for the fulfillment location. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update variance and inventory value accounts in the financial system.

12.2.7 Return Order Process

From the General Process Type, configure the following events for the INVENTORY_COST_CHANGE Transaction ID:

- INVENTORY_VALUE_CHANGE

12.2.7.1 INVENTORY_VALUE_CHANGE

When is this event raised?

At the time of return order receipt this event is raised from the inventory management module for each return receipt line with the inventory value change information for the return location. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update variance and inventory value accounts in the financial system.

12.2.8 Callback from Financial System for Inventory Value Adjustment

This interface is implemented as a call to the updateInventoryCost API in Selling and Fulfillment Foundation. This should be used whenever the Accounts Payable application generates a variance between expected PO cost and the actual cost on the Payables Invoice. The variance amount should be passed back to Selling and Fulfillment Foundation to be reflected in the inventory value. Selling and Fulfillment Foundation then tries to adjust the inventory value and re-compute the average cost. If the total on-hand is less than the purchase quantity (due to subsequent shipments or issues), the total variance is prorated and applied to the remaining on-hand inventory. An additional event is raised to adjust the difference in the financial system. For more information about the input attributes for the interface, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Selling and Fulfillment Foundation?

Inventory value is adjusted by the variance amount. Average cost is recomputed. If the total on-hand is less than what has to be adjusted, the total variance is prorated and applied on the remaining on-hand inventory. The amount not applied is passed back to the financial application so that it can be stored in an appropriate variance account.

Using the INVENTORY_COST_UPDATE Transaction ID, configure the following events for the Callback from Financial System process:

- COULD_NOT_APPLY_INV_VALUE_CHANGE

12.2.8.1 COULD_NOT_APPLY_INV_VALUE_CHANGE

When is this event raised?

The amount not applied on Selling and Fulfillment Foundation is passed back to the financial application by raising this event which publishes the variance amount details. For more information about the data published by the event, see the *Selling and Fulfillment Foundation: Javadocs*.

What are the expected updates on Financial System?

This event can be used to update the appropriate variance account on the financial system.

13

Rapid Deployment Features

This chapter explains the Selling and Fulfillment Foundation Rapid Deployment Tool (RDT) and how to utilize its features for the rapid deployment of Selling and Fulfillment Foundation.

The rapid deployment features include:

- Interface Field Mapping Documents
- Initial Data Loading

In addition to these rapid deployment features, Selling and Fulfillment Foundation provides a mechanism to create a new Sterling Warehouse Management System node from an existing node.

For more information about Copying an Existing Node to a New Node, Onboarding an Enterprise to a Node, Offboarding an Enterprise from a Node, and Deleting the Current Node, refer to the *Sterling Warehouse Management System: Configuration Guide*.

13.1 Interface Field Mapping Documents

An Interface Field Mapping Document specifies integration mapping between Selling and Fulfillment Foundation and an external system. Typically, it is a Microsoft Excel document based on the input and output XMLs of the Selling and Fulfillment Foundation APIs or custom APIs written at the implementation phase of a project.

This feature describes the methodology to generate a Microsoft Excel-compatible XML spreadsheet file from the input/output XML file of an API, which can be used to create the Interface Field Mapping Document.

Note: The Interface Field Mapping Template generation tool can only be used in Microsoft Windows environment.

13.1.1 Generating Interface Field Mapping Template Documents

Selling and Fulfillment Foundation provides a tool to generate Interface Field Mapping Template documents from input/output XMLs.

The input XML for this generation could be an Input/Output XML from a Selling and Fulfillment Foundation-exposed API or an XML for a custom API, which allows the generation of Interface Field Mapping Template documents for custom APIs created during implementation.

The tool generates the Interface Field Mapping Template document as a Microsoft Excel XML spreadsheet document, which can be opened in Microsoft Excel and modified to specify the mapping details.

13.1.1.1 Generating Interface Field Mapping Template Documents Using the Generation Tool

To generate the XML spreadsheet use the following command line tool:

```
generateExcelXML {INXML} {INXSL} {OUTXML} {HTML} {TITLE}
```

where,

- INXML – Name of the XML file for which the XML spreadsheet should be generated
- INXSL – Name of the XSL file which is used to generate the XML spreadsheet
- OUTXML – Name of the XML spreadsheet file to be generated
- HTML – Name of the HTML file which contains the description of the Input XML attributes.

Note: If you are running the RDT in a Unix environment, you must insert an extra "\" for every "\" that you use in the HTML file name attribute. For example, if the filename is \\server\directory\file.html, you must specify the filename as \\\\server\\directory\\file.html.

- TITLE - The title that is displayed after you generate the XML spreadsheet. Figure 13–1 shows the PO Download title.

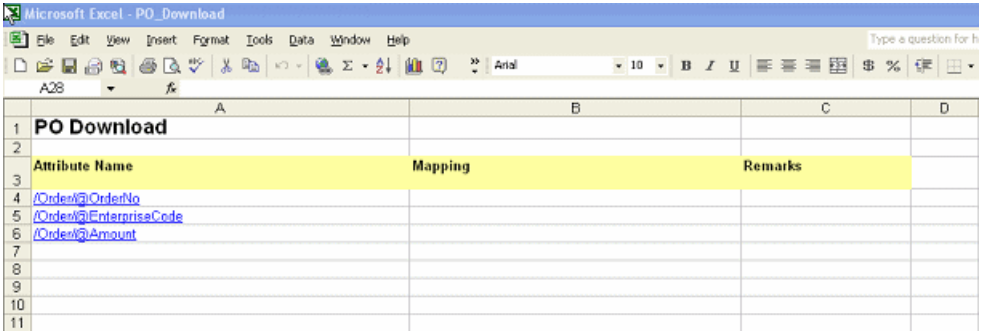
This tool is located in <INSTALL_DIR>/bin directory. This can also be used to generate XML spreadsheets for custom APIs.

13.1.1.2 Using Interface Field Mapping Template Documents

The XML spreadsheet generated using the command line tool can be opened and edited using Microsoft Excel (Versions 2002 and above).

The XML spreadsheet provides the Attribute Name, Mapping, and Remarks for each attribute.

Figure 13–1 A Sample XML Spreadsheet



Clicking on an attribute name launches the relevant datatype and description. These integration field mappings may be modified as applicable and saved.

13.2 Initial Data Loading

Selling and Fulfillment Foundation provides a initial data-loading tool for loading configuration data from legacy or ERP systems. The Initial Data

Loading (IDL) tool utilizes the bare minimum information required by the warehouse to be functional.

13.2.1 Initial Data-Loading Services

The Initial Data Loading (IDL) tool works based on the Service Definition Framework (SDF).

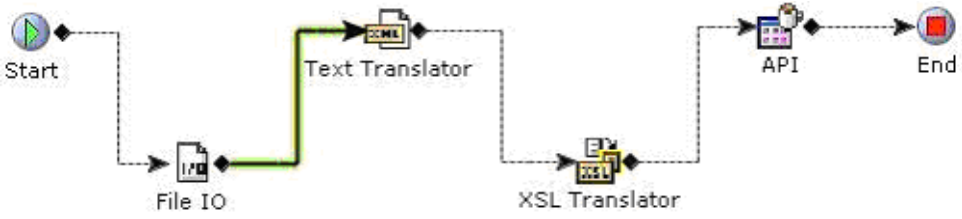
The IDL tool provides services to create the following configuration data:

- Items
- Shipping Cartons
- Locations
- SKU Dedications
- Location Inventory

To use the services provided for IDL, the configuration data to be loaded from the legacy or ERP systems should be made available in a comma delimited flat file.

The IDL tool uses services to convert the data into the XML format, required by the corresponding APIs to create or modify the relevant information in the warehouse.

Figure 13–2 Sample Service as displayed in the Applications Manager



To begin the initial data loading process, the integration server should be started by navigating `<runtime>/bin` folder and entering the following command:

```
<runtime>/bin/startIntegrationServer.sh <servername>
```

For more information about running the Selling and Fulfillment Foundation Integration Server, refer to the *Selling and Fulfillment Foundation: Installation Guide*.

The `RDTConfigDataFormat.xls` file located in the `<runtime>/repository/xapi/template/merged/RDTConfigSchemas` folder contains the data sequence and the headers required for the corresponding service provided in the IDL module of the RDT.

All Selling and Fulfillment Foundation services follow the predefined sequence specified in the `RDTConfigDataFormat.xls` file for calling the components:

- The File IO Receiver is used to read the data from the delimited flat file
- The Text Translator component is used to convert the delimited data to XML format
- The XSL Translator component is used to convert the XML into a format that is the input to an API, and
- The API component is used to call the business API for creating or modifying the data.

Each service reads the input data line by line from the delimited flat files. Thus, all the details required for a configuration should be provided in a single line, separated by commas, and in a fixed sequence. The first item in each line is the header, and it is fixed for each service. If the first item is anything other than the header then that row is not considered for processing.

Error Handling in Initial Data Loading (IDL) Tool:

The error handing for Initial Data Loading services is undertaken at two levels:

a. When there is an error in translating the flat file into an xml file as per the defined schema, the file is pushed to the working directory and an error file indicating the error is added to the error directory. The error may now be fixed and the modified flat file reprocessed.

b. When the API throws an exception for a record, it is sent to the default exception queue where it can be viewed in the exception console by searching for exceptions in initial status. The input xml

may now be modified by providing the right input, and reprocessed using the reprocess button.

13.2.1.1 Item Configuration Data-Loading

This service enables you to create an item or modify the attributes of an existing item for which inventory is stored in the warehouse. It calls the `manageItem()` API.

Table 13–1, "Format for Item Configuration Data Loading Service" explains the format of the headers and the sequence of items to be provided for this service. For more information, see the *Selling and Fulfillment Foundation: Javadocs*.

Table 13–1 Format for Item Configuration Data Loading Service

Attribute	Description	Sequence	Data Type	Size
ITEMHEDR	The item header identifier	1	String	6
ItemID	The unique identifier for an item that belongs to a catalog organization	2	String	40
OrganizationCode	The code of the organization whose product information is being stored	3	String	24
UnitOfMeasure	The unit of measure for item quantity	4	String	40
GlobalItemID	The unique global identifier used to cross reference an item with another catalog organization	5	String	128
Description	A localized description	6	String	200
ProductLine	The product line of an item	7	String	100
KitCode	The kit code of an item. Value 'LK' indicates a logical kit, while PK indicates a physical kit	8		
ItemGroupCode	The code of the item group. This is used to identify whether the item is a Product, Provided Service, Provided Service Option, Delivery Service, or Delivery Service Option	9	String	20
UnitCost	The cost of one unit of the item	10	Decimal	19
CostCurrency	The currency in which the item's cost is defined	11	String	20

Table 13–1 Format for Item Configuration Data Loading Service

Attribute	Description	Sequence	Data Type	Size
CountryOfOrigin	The item's country of origin or manufacture	12	String	40
ItemType	The generic type of the item	13	String	40
UnitWeight	The weight of one unit of the item	14	Decimal	14
WeightUOM	The unit of measure in which the item's weight is defined	15	Decimal	14
UnitHeight	The height of one unit of the item	16	Decimal	14
UnitLength	The length of one unit of the item	17	Decimal	14
UnitWidth	The width of one unit of the item	18	Decimal	14
SerializedFlag	This indicates whether the item is serialized	19	Boolean	1
TagControlFlag	This indicates whether the item is tag controlled	20	Boolean	1
TimeSensitive	This indicates whether the item is time sensitive	21	Boolean	1
IsFifoTracked	This indicates whether the item is FIFO tracked	22	Boolean	1
IsSerialTracked	This indicates whether the item is serial tracked	23	Boolean	1
HarmonizedCode	The harmonized code of the item	24	String	40
NMFCCode	The NMFC code of the item	25	String	40
VelocityCode	The velocity code of the item	26	String	40
ECCNNo	The ECCN number of the item	27	String	40
HazmatClass	The hazardous material classification of the item	28	String	40
CommodityCode	The commodity code of the item	29	String	40
StorageType	The storage type of the item	30	String	40
AddName1	The name of the first additional attribute	31	String	20
AddValue1	The value of the first additional attribute	32	String	2000

Table 13–1 Format for Item Configuration Data Loading Service

Attribute	Description	Sequence	Data Type	Size
AddName2	The name of the second additional attribute	33	String	20
AddValue2	The value of the second additional attribute	34	String	2000
LotNumber	The lot number of the item. This indicates whether this attribute can be used as a Tag Identifier or a Tag Descriptor. Valid values are: 01 - Use as Tag Descriptor 02 - Use as Tag Identifier 03 - Not used	35	String	2
LotAttribute1	The lot attribute of the item. This indicates whether this attribute can be used as a Tag Descriptor. Valid values are: 01 - Use as Tag Descriptor 03 - Not Use	36	String	2
LotAttribute2	The lot attribute of the item. This indicates whether this attribute can be used as a Tag Descriptor. Valid values are: 01 - Use as Tag Descriptor, 03 - Not used.	37	String	2
CaseQuantity	The quantity of one case of the item	38	Decimal	14
CaseWeight	The weight of one case of the item	39	Decimal	14
CaseLength	The length of one case of the item	40	Decimal	14
CaseWidth	The width of one case of the item	41	Decimal	14
CaseHeight	The height of one case of the item	42	Decimal	14
PalletQuantity	The quantity of one pallet of the item	43	Decimal	14
PalletWeight	The weight of one pallet of the item	44	Decimal	14
PalletLength	The length of one pallet of the item	45	Decimal	14
PalletWidth	The width of one pallet of the item	46	Decimal	14
PalletHeight	The height of one pallet of the item	47	Decimal	14
DimensionUOM	The unit of measure that define the dimensions of the item	48	String	40

The schema files used by each component of the service and the API called by the service are as follows:

- **Service Name:** Items
- **Service Group:** InitialDataLoad
- **Text Translator:** ModifyItemSchema
- **XSL Translator:** ModifyItem
- **API:** manageItem
- **Server Name:** ItemLoader

13.2.1.2 Shipping Carton Data-Loading

This service creates shipping cartons (modelled as items) that are stored in the warehouse. It calls the `createItem()` API.

Table 13–2, "Format for Shipping Carton Data Loading Service" explains the format of the headers and the sequence of items to be provided for this service. For more information, see the *Selling and Fulfillment Foundation: Javadocs*.

Table 13–2 Format for Shipping Carton Data Loading Service

Attribute	Description	Sequence	Data Type	Size
ITEMHEDR	The item header identifier	1	String	6
ItemID	The unique identifier for an item that belongs to a catalog organization	2	String	40
OrganizationCode	The code of the organization whose product information is being stored	3	String	24
UnitOfMeasure	The unit of measure for item quantity	4	String	40
UnitWeight	The weight of one unit of the item	5	Decimal	14
UnitHeight	The height of one unit of the item	6	Decimal	14
UnitLength	The length of one unit of the item	7	Decimal	14
UnitWidth	The width of one unit of the item	8	Decimal	14
MaxCntrWeight	The maximum weight of the carton	9		

The schema files used by each component of the service and the API called by the service are as follows:

- **Service Name:** ShippingCartons
- **Service Group:** InitialDataLoad
- **Text Translator:** ShippingCartonSchema
- **XSL Translator:** ShippingCarton
- **API:** createItem
- **Server Name:** ShippingCartonLoader

13.2.1.3 Location Data-Loading

This service creates locations in a zone within a node in the warehouse. These locations specify the physical space where inventory is stored. It calls the `manageLocation()` API.

Table 13–3, "Format for Location Data Loading Service" explains the format of the headers and the sequence of items to be provided for this service. For more information, see the *Selling and Fulfillment Foundation: Javadocs*.

Table 13–3 Format for Location Data Loading Service

Attribute	Description	Sequence	Data Type	Size
LOCAHEDR	The location header identifier	1	String	8
LocationId	The unique identifier for the location. This in conjunction with NODE_KEY identifies a unique location in the node	2	String	40
Node	The node to which the location belongs.	3	Key	24

Table 13–3 Format for Location Data Loading Service

Attribute	Description	Sequence	Data Type	Size
LocationType	The system defined classification of location to aid association of locations of certain types for certain other operations with WMS. The supported types are: INTRANSIT (Mobile locations), STAGING, VIRTUAL, REGULAR and DOCK. For example, all equipment locations should be of type INTRANSIT. If LocationType is passed blank or passed unallowed values then default LocationType is taken as REGULAR	4	String	40
ZoneId	The zone to which the location belongs. This in conjunction with the node key identifies a unique zone within the node.	5	String	40
AisleNumber	The aisle number of the location. Locations belong to zones, which have travel aisle's between them. A zone could belong to multiple aisles and multiple zones could belong to an aisle. But a location in a zone belongs to one and only one aisle.	6	Integer	9
LevelLocation	The level number of the location. This indicates the height of the location (y-co-ordinate of the location from the floor) classified as levels. Level attribute of the location is used in arriving at locations nearest to the dedicated locations algorithm used in put away. Typically, the level attribute is contained within the location ID.	7	Integer	9
BayNumber	The bay number of the location. Typically, the aisle, level and bay put together gives the physical location of the location in the node if they are based on coordinate system. Bay attribute of the location (x-coordinate from the beginning of the aisle) is used in arriving at locations nearest to the dedicated locations algorithm used in put away. Typically, the bay attribute is contained within the location ID.	8	Integer	9

Table 13–3 Format for Location Data Loading Service

Attribute	Description	Sequence	Data Type	Size
MoveInSequenceNumber	The move in sequence number of the location. This is used by task management for location suggestion while moving in inventory (put away). The put away location selection algorithm uses this information to select locations amongst a list of locations based on its move in sequence.	9	Integer	9
MoveOutSequenceNumber	The move out sequence number of the location. This is used by task management for location suggestion while moving out inventory (picking). The pick location selection algorithm uses this information to select locations amongst a list of locations based on its move in sequence.	10	Integer	9
InStagingLocationId	The in staging location id indicates the Drop off location (For moves coming into a location, they may be dropped here)	11	String	40
OutStagingLocationId	The out staging location id indicates the Out Drop off Location (Location where moves originated at this location, may be dropped).	12	String	40

Table 13–3 Format for Location Data Loading Service

Attribute	Description	Sequence	Data Type	Size
VelocityCode	The velocity code of the location classifies items as A, B or C class items based on whether they are fast selling, not so fast selling and low selling item. These item classifications are typically followed by all enterprises to optimize certain operations such as sourcing and stocking. The reason we have locations preferring certain velocity codes is that, we could have locations closer to dock stocking A class items, and locations furthest away from the dock stocking C class items. Velocity code is a preference on the location and not a constraint. If A class items fill up all locations meant for A class items, then they can go in to B and then C. Similarly C can go to B and then A for lack of space in the respective locations preferred for a specific velocity code. B class items go into C and then into A. If VelocityCode is passed blank or passed unallowed values then default VelocityCode is taken Last VelocityCode in the alphabetic sequence in common code of type VELOCITY_CODE.	13	String	40
LocationSizeCode	The location size code defines the capacity of a location. All locations having the same size (dimensions and ability to hold the same weight) are classified under the same size code. This maps to the primary key attribute of the YFS_LOCATION_SIZE_CODE table.	14	String	40

Table 13–3 Format for Location Data Loading Service

Attribute	Description	Sequence	Data Type	Size
StorageCode	Storage code is an attribute of the location that allows the warehouse to store items that have the same storage profile as that of the location. For example, hazardous inflammable items need locations close to fire extinguishers. In this case the locations are marked as having a storage code, which is suitable for storing Inflammable items. This ensures that only inflammable items get to these locations.	15	String	40
X Co-ordinate	X Co-ordinate for a location in the warehouse	16	Number	14
Y Co-ordinate	Y Co-ordinate for a location in the warehouse	17	Number	14
Z Co-ordinate	Z Co-ordinate for a location in the warehouse	18	Number	14

The schema files used by each component of the service and the API called by the service are as follows:

- **Service Name:** Locations
- **Service Group:** InitialDataLoad
- **Text Translator:** LocationSchema
- **XSL Translator:** Location
- **API:** manageLocation
- **Server Name:** LocationLoader

13.2.1.4 SKU Dedication Data-Loading

This service modifies the attributes of a location, and is basically used to dedicate a location as a dedicated location. A dedicated location is one that stores inventory for a particular item only. It calls the `modifyLocation()` API.

Note: This service require 9 attributes. If you are giving 8 commas to separate these 9 attributes, you have to make sure that the last attribute is non-blank. If it is blank, you have to close it with an extra comma, which means the 9th comma. In this case, 9 commas does not mean that there are 10 attributes.

Table 13–4, "Format for SKU Dedication Data Loading Service" explains the format of the headers and the sequence of items to be provided for this service. For more information, see the *Selling and Fulfillment Foundation: Javadocs*.

Table 13–4 Format for SKU Dedication Data Loading Service

Attribute	Description	Sequence	Data Type	Size
SKUDEDIC	The SKU Dedication header identifier	1	String	8
LocationId	The identifier for the location. This in conjunction with NODE_KEY identifies a unique location in the node	2	String	40
Node	The node to which the location belongs to	3	Key	24
EnterpriseCode	The code of the enterprise to which the location is dedicated	4	String	40
ItemId	The item identifier of the SKU	5	String	40
UnitOfMeasure	The unit of measure of the SKU	6	String	40
ProductClass	The product class of the SKU	7	String	40
SegmentType	SKUs are sometimes custom made. This field stores the customization details.	8	String	40
Segment	SKUs are sometimes custom made. This field stores the customization details. When inventory is customized for a specific order, it needs to be tracked separately so that it can be allocated to that order	9	String	40

The schema files used by each component of the service and the API called by the service are as follows:

- **Service Name:** SkuDedications
- **Service Group:** InitialDataLoad
- **Text Translator:** SkuDedicationSchema
- **XSL Translator:** SkuDedication
- **API:** modifyLocation
- **Server Name:** SkuDedicationLoader

13.2.1.5 Location Inventory Data-Loading

This service adds the inventory for the previously created items and locations in the warehouse. It calls the `adjustLocationInventory()` API.

Table 13–5, "Format for Inventory Data Loading Service" explains the format of the headers and the sequence of items to be provided for this service. For more information, see the *Selling and Fulfillment Foundation: Javadocs*.

Table 13–5 Format for Inventory Data Loading Service

Attribute	Description	Sequence	Data Type	Size
ALOCINVN	The inventory header identifier	1	String	8
EnterpriseCode	The Inventory Organization Code. This indicates the Enterprise whose product information is being stored.	2	String	24
Node	The Business key or unique identifier for a ship node.	3	String	24
CaseId	The identifier for a case. This gives the LPN information for adjustment.	4	String	40
LocationId	The identifier for a location. This forms unique key of this table in conjunction with NODE_KEY. Indicates the location from where the inventory is being adjusted. LocationId becomes mandatory, if CaseId/PalletId is not passed.	5	String	40
PalletId	The identifier for a pallet. This gives the LPN information for adjustment.	6	String	40

Table 13–5 Format for Inventory Data Loading Service

Attribute	Description	Sequence	Data Type	Size
InventoryStatus	The inventory status gives the status of the inventory. Only one level InventoryStatus transitions happen for the inventory for positive adjustments. Negative adjustments do not take care of InventoryStatus transitions. If not passed, the status is taken as blank.	7	String	10
SegmentType	The segment type for particular enterprise or organization. SegmentType becomes mandatory if Segment is passed.	8	String	40
Segment	The segment for particular enterprise or organization. Segment becomes mandatory if SegmentType is passed.	9	String	40
Quantity	This gives the adjustment quantity for the inventory. The negative quantity specifies negative adjustment and positive quantity denotes positive adjustment. Quantity becomes mandatory if SerialDetail does not provide quantity for adjustment.	10	Decimal	14
ItemID	The item identity for the inventory	11	String	40
UnitOfMeasure	The unit of measure for the item	12	String	40
ProductClass	The product class for the item	13	String	40
LotNumber	The lot number for the inventory	14	String	40
LotAttribute1	The lot attribute for the inventory	15	String	40
LotAttribute2	The lot attribute for the inventory	16	String	40

Table 13–5 Format for Inventory Data Loading Service

Attribute	Description	Sequence	Data Type	Size
ShipByDate	The date by which the inventory has to be shipped	17	Date	10
SerialNo	The unique identifier for each serial	18	String	40
ReasonCode	The reason code for the inventory transaction. The business significance of this reason code is that inventory bins are tied to this reason code, which is used to adjust inventory (for global inventory visibility purposes) on host systems. This is mandatory if inventory is getting updated. Some Selling and Fulfillment Foundation APIs doing inventory adjustments expect some adjustment reason codes to be configured in the system. These are RECEIPT used by Receiving, PACK used by Packing functions and SHIP used by Shipment. PACK should have a bin associated while RECEIPT and SHIP should not have bin location associations.	19	String	40

The schema files used by each component of the service and the API called by the service are as follows:

- **Service Name:** Inventory
- **Service Group:** InitialDataLoad
- **Text Translator:** AdjustLocationInventorySchema
- **XSL Translator:** AdjustLocationInventory
- **API:** adjustLocationInventory
- **Server Name:** InventoryLoader

13.2.1.6 Hazmat Data-Loading

Selling and Fulfillment Foundation supports the Hazmat Data-Load tool, which works based on the Service Definition Framework (SDF) to load the Hazmat data to the YFS_HAZMAT_COMPLIANCE table.

Use this tool to load, modify, or delete the Hazmat data as specified by the Department Of Transportation (DOT). Based on the action passed, the tool appropriately loads, modifies, or deletes the Hazmat data from the YFS_HAZMAT_COMPLIANCE table.

13.2.1.6.1 Initially Loading the Hazmat Data

To initially load the Hazmat data:

1. Using any standard Web browser, download the CFR49 Hazmat data file 172101ascii.zip from http://hazmat.dot.gov/enforce/forms/ohmforms.htm#101.
2. Extract the cfr.dat file that is stored in the 172101ascii.zip file to the <INSTALL_DIR>/bin directory.
3. The Hazmat Data-Load tool requires the data file to be in a readable format in order to be processed by the Selling and Fulfillment Foundation SDF. To convert the downloaded data file into this format, run the prepareHazmatData.xml script located in the <INSTALL_DIR>/bin directory. This script takes three input parameters: runtime, datafile, and operation (the valid values for which are CHG and DEL. Use the CHG operation to load or modify the Hazmat data, and the DEL operation to delete data). For example, in UNIX or Linux, run this command:

   ```
   sci_ant.sh -f prepareHazmatData.xml -Druntime=<INSTALL_DIR>
   -Ddatafile=cfr.dat -Doperation=CHG
   ```

   For Windows, run this command:

   ```
   sci_ant.cmd -f prepareHazmatData.xml -Druntime=<INSTALL_
   DIR> -Ddatafile=cfr.dat -Doperation=CHG
   ```

4. After running the prepareHazmatData.xml script, each record in the data file is appended with the HAZMATDATA string and the operation passed, which can now be processed by the Selling and Fulfillment Foundation SDF.
5. Start the integration server by passing the <servername> as HazmatDataLoader.

   For more information about starting the integration server, see the *Selling and Fulfillment Foundation: Installation Guide*.

13.2.1.6.2 Maintaining the Hazmat Data

To insert, modify, or delete the Hazmat data as specified by DOT:

1. Using any standard Web browser, the Hazmat data details can be found at: http://hazmat.dot.gov/regs/hmtentries.htm
2. The Hazmat data listed in the `hmtentries.htm` file needs to be in a readable format in order to be processed by the Selling and Fulfillment Foundation SDF. Therefore, create two different Hazmat data files, one for additions or modifications, and another file for deletions. Ensure that the Hazmat data file format that you created exist in the `<INSTALL_DIR>/bin` folder and is of the same format as the `cfr.dat` file that is stored in the `172101ascii.zip` file.
3. Repeat Step 3 through Step 5.

Index

F

G

H

I

J

L

M

O

P

R

S

T

U

W

X

Y