Windchill Enterprise Systems Integration
Customization Guide

Windchill 7.0
December 2003
Copyright © 2000 Oracle Corporation. The Java™ Telnet Applet (StatusPeer.java, TelnetIO.java, TelnetWrapper.java, TimedOutException.java), Copyright © 1996, 97 Mattias L. Jugel, Marcus Meißner, is redistributed under the GNU General Public License. This license is from the original copyright holder and the Applet is provided WITHOUT WARRANTY OF ANY KIND. You may obtain a copy of the source code for the Applet at http://www.mud.de/se/jta (for a charge of no more than the cost of physically performing the source distribution), by sending e-mail to leo@mud.de or marcus@mud.de—you are allowed to choose either distribution method. The source code is likewise provided under the GNU General Public License. GTK+ The GIMP Toolkit are licensed under the GNU LPGL. You may obtain a copy of the source code at http://www.gtk.org/, which is likewise provided under the GNU LPGL. zlib software Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND
This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) (OCT'95) or DFARS 227.7202-1(a) and 227.7202-3(a) (JUN'95), is provided to the US Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 (OCT'88) or Commercial Computer Software-Restricted Rights at FAR 52.227-19(c)(1)-(2) (JUN'87), as applicable.
040103

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

1 Overview .......................................................................................................................... 1

1.1 About this Guide ........................................................................................................ 2

2 Customization Areas .................................................................................................... 3

2.1 Windchill ESI Services Customizations .................................................................. 3

2.1.1 Modify Windchill ESI Properties ...................................................................... 3

2.1.2 Modify Map Files .............................................................................................. 5

2.1.3 Create/Modify XML Schema .......................................................................... 12

2.1.4 Extend Windchill ESI Java Code .................................................................... 12

2.1.5 Create RPC Classes and Methods .................................................................. 19

2.1.6 Modify Difference APIs ................................................................................. 20

2.1.7 Transaction Management and Error Handling ............................................. 25

2.2 EAI Software Components Customizations ............................................................ 27

2.2.1 XML (eXtended Markup Language) schemas .............................................. 27

2.2.2 Common Business Object (CBO) .................................................................. 30

2.2.3 Business logic in BusinessWorks ................................................................. 31

2.2.4 SAP R/3 API descriptions and characterization .......................................... 34

2.2.5 SAP R/3 API Response Characterization .................................................. 42

2.2.6 Internationalization (I18N) & Localization (L10N) .................................... 45

2.3 Windchill Customizations ....................................................................................... 45

2.3.1 Attribute Length ............................................................................................ 45

3 Windchill ESI Services Modules .................................................................................. 46

3.1 Windchill ESI Mapper ............................................................................................. 46

3.1.1 Mapper Classes ............................................................................................. 46

3.1.2 Group and Map Information Classes .................................................................. 51

3.1.3 Customization Points .................................................................................... 53

3.2 VDB Builder ............................................................................................................ 53

3.2.1 Classes .......................................................................................................... 55

3.2.2 Customization Points .................................................................................... 64

3.3 Distribution Targets ............................................................................................... 65

3.3.1 Distribution Target Classes .......................................................................... 65

3.3.2 Establishing Distribution Target Assignment Classes ................................ 69

3.3.3 Distribution Target Assignment Determination Classes ................................ 73

3.3.4 GUI Support .................................................................................................. 75

3.3.5 Info*Engine Tasks ....................................................................................... 76

3.3.6 Customization Points .................................................................................... 77

3.4 Transaction Management ......................................................................................... 78

3.4.1 Creating Publishing History Objects ............................................................ 79

3.4.2 Classes ........................................................................................................... 82

3.4.3 GUI Support .................................................................................................. 93

3.4.4 GetHistory Support ...................................................................................... 94
<table>
<thead>
<tr>
<th>Figures</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windchill ESI Architecture</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Default Part XML Element Structure</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Example 1 Customized Part XML Element Structure</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Example 2 Customized Part XML Element Structure</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Post-CBO Filtering Layers</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>Master Process Flow</td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>Windchill ESI Mapper Classes</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>Using Windchill ESI Mapper</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>Windchill ESI Group and Map Information</td>
<td>51</td>
</tr>
<tr>
<td>10</td>
<td>Info*Engine VDB</td>
<td>53</td>
</tr>
<tr>
<td>11</td>
<td>VDB Builder Classes</td>
<td>55</td>
</tr>
<tr>
<td>12</td>
<td>Additional VDB Builder Classes</td>
<td>58</td>
</tr>
<tr>
<td>13</td>
<td>Windchill ESI Renderer Support</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>Using a Windchill ESI Renderer</td>
<td>62</td>
</tr>
<tr>
<td>15</td>
<td>ESITarget objects and Windchill Organizations</td>
<td>67</td>
</tr>
<tr>
<td>16</td>
<td>Distribution Target Assignment Objects</td>
<td>69</td>
</tr>
<tr>
<td>17</td>
<td>ESITarget Utility</td>
<td>70</td>
</tr>
<tr>
<td>18</td>
<td>Distribution Target Determination Strategies</td>
<td>74</td>
</tr>
<tr>
<td>19</td>
<td>Distribution Target User Interface Message Flow</td>
<td>76</td>
</tr>
<tr>
<td>20</td>
<td>Tracking Publishing History</td>
<td>79</td>
</tr>
<tr>
<td>21</td>
<td>Successful Transaction Status Values over Time</td>
<td>80</td>
</tr>
<tr>
<td>22</td>
<td>Failed Transaction Status Values over Time</td>
<td>81</td>
</tr>
<tr>
<td>23</td>
<td>Enumerated Types in Windchill ESI Transaction</td>
<td>82</td>
</tr>
<tr>
<td>24</td>
<td>Transaction Object Inheritance</td>
<td>83</td>
</tr>
<tr>
<td>25</td>
<td>Transaction Utility</td>
<td>86</td>
</tr>
<tr>
<td>26</td>
<td>Txn Package subclasses of ESITargetAssociation</td>
<td>92</td>
</tr>
<tr>
<td>27</td>
<td>Windchill ESI Transaction User Interface Message Flow</td>
<td>94</td>
</tr>
<tr>
<td>28</td>
<td>GetHistory API Support</td>
<td>95</td>
</tr>
<tr>
<td>29</td>
<td>ECN Structure</td>
<td>102</td>
</tr>
<tr>
<td>30</td>
<td>Relationships of Changeables</td>
<td>103</td>
</tr>
<tr>
<td>31</td>
<td>Windchill ESI ECN Release Support</td>
<td>105</td>
</tr>
<tr>
<td>32</td>
<td>Windchill ESI Change Order Renderer</td>
<td>108</td>
</tr>
<tr>
<td>33</td>
<td>Windchill ESI Change Header Renderer</td>
<td>109</td>
</tr>
<tr>
<td>34</td>
<td>Windchill ESI ECN Renderer Factory</td>
<td>111</td>
</tr>
<tr>
<td>35</td>
<td>Publishing an ECN</td>
<td>112</td>
</tr>
<tr>
<td>36</td>
<td>Rendering an ECN</td>
<td>113</td>
</tr>
<tr>
<td>37</td>
<td>Rendering the Changeables on an ECN</td>
<td>114</td>
</tr>
<tr>
<td>38</td>
<td>BOM Renderer Interfaces</td>
<td>117</td>
</tr>
<tr>
<td>39</td>
<td>BOM Renderer Classes</td>
<td>118</td>
</tr>
<tr>
<td>40</td>
<td>BOM Component, Substitute, and Reference Designator Renderers</td>
<td>122</td>
</tr>
<tr>
<td>41</td>
<td>BOM Renderer Factory</td>
<td>126</td>
</tr>
<tr>
<td>42</td>
<td>BOM Tree Navigator</td>
<td>128</td>
</tr>
<tr>
<td>43</td>
<td>BOM Utility</td>
<td>130</td>
</tr>
<tr>
<td>44</td>
<td>GetBOM RPC Processing</td>
<td>132</td>
</tr>
<tr>
<td>45</td>
<td>Windchill ESI Part Release Response Renderers</td>
<td>136</td>
</tr>
</tbody>
</table>
Figure 92  Response to CBO mapping........................................................................ 289
Figure 93  Response to CBO mapping – if Statements............................................... 289
### Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Windchill ESI Properties and Map Files</td>
<td>5</td>
</tr>
<tr>
<td>Table 2</td>
<td>Windchill ESI Renders</td>
<td>13</td>
</tr>
<tr>
<td>Table 3</td>
<td>Attribute Lengths</td>
<td>46</td>
</tr>
<tr>
<td>Table 4</td>
<td>Naming Standard - Phrase Definitions</td>
<td>191</td>
</tr>
<tr>
<td>Table 5</td>
<td>Business Object values</td>
<td>193</td>
</tr>
<tr>
<td>Table 6</td>
<td>Subject Standards</td>
<td>194</td>
</tr>
<tr>
<td>Table 7</td>
<td>JMS Queues Identifiers</td>
<td>194</td>
</tr>
<tr>
<td>Table 8</td>
<td>Configuration Types</td>
<td>195</td>
</tr>
<tr>
<td>Table 9</td>
<td>Primary Process Definition Fields</td>
<td>196</td>
</tr>
<tr>
<td>Table 10</td>
<td>Primary Process Definition Starter Fields</td>
<td>196</td>
</tr>
<tr>
<td>Table 11</td>
<td>Primary Process Definition Object Fields</td>
<td>197</td>
</tr>
<tr>
<td>Table 12</td>
<td>Secondary Process Definition Starter Fields</td>
<td>198</td>
</tr>
<tr>
<td>Table 13</td>
<td>Secondary Process Definition Object Fields</td>
<td>198</td>
</tr>
<tr>
<td>Table 14</td>
<td>Secondary Process Definition Action Fields</td>
<td>199</td>
</tr>
<tr>
<td>Table 15</td>
<td>Secondary Process Definition Purpose Fields</td>
<td>199</td>
</tr>
<tr>
<td>Table 16</td>
<td>Common Process Definitions Purposes</td>
<td>199</td>
</tr>
<tr>
<td>Table 17</td>
<td>Groups that Retry APIs Fields</td>
<td>200</td>
</tr>
<tr>
<td>Table 18</td>
<td>Groups that Error Handle Fields</td>
<td>200</td>
</tr>
<tr>
<td>Table 19</td>
<td>Object Fields for Groups that Iterate</td>
<td>201</td>
</tr>
<tr>
<td>Table 20</td>
<td>End Fields for Groups that Iterate</td>
<td>201</td>
</tr>
<tr>
<td>Table 21</td>
<td>Branch Description</td>
<td>201</td>
</tr>
<tr>
<td>Table 22</td>
<td>Variable Node names</td>
<td>202</td>
</tr>
<tr>
<td>Table 23</td>
<td>Method Fields</td>
<td>208</td>
</tr>
<tr>
<td>Table 24</td>
<td>Primary Build Fields</td>
<td>209</td>
</tr>
<tr>
<td>Table 25</td>
<td>Working Repository Fields</td>
<td>209</td>
</tr>
<tr>
<td>Table 26</td>
<td>ESISTUB Queue Permissions</td>
<td>216</td>
</tr>
<tr>
<td>Table 27</td>
<td>ESIEventUserA.xml Field Descriptions</td>
<td>217</td>
</tr>
<tr>
<td>Table 28</td>
<td>ESIResponseUserA.xml Field Descriptions</td>
<td>218</td>
</tr>
<tr>
<td>Table 29</td>
<td>Windchill Simulation module Global Variables</td>
<td>221</td>
</tr>
<tr>
<td>Table 30</td>
<td>Windchill Simulation Module JMS Configuration</td>
<td>224</td>
</tr>
<tr>
<td>Table 31</td>
<td>Generator_Validator Components</td>
<td>225</td>
</tr>
<tr>
<td>Table 32</td>
<td>Stub_ESIEvent_Publication_PD Components</td>
<td>226</td>
</tr>
<tr>
<td>Table 33</td>
<td>Stub_ESIRequest_Subscribe_ESIResponse_Publish_PD Components</td>
<td>226</td>
</tr>
<tr>
<td>Table 34</td>
<td>Stub_Result_ResultResponse_PD Components</td>
<td>227</td>
</tr>
<tr>
<td>Table 35</td>
<td>SAP R/3 APIs Used by Windchill Enterprise Systems Integration</td>
<td>257</td>
</tr>
<tr>
<td>Table 36</td>
<td>SAP R/3 Transaction Codes to Verify Business Object Creation</td>
<td>265</td>
</tr>
<tr>
<td>Table 37</td>
<td>SAP R/3 APIs for each Business Function Test Scenario</td>
<td>266</td>
</tr>
<tr>
<td>Table 38</td>
<td>Business Functions Supported by SAP R/3 Readiness Validator</td>
<td>270</td>
</tr>
<tr>
<td>Table 39</td>
<td>BusinessWorks Process Definitions in the SAP R/3 Readiness Validator</td>
<td>271</td>
</tr>
<tr>
<td>Table 40</td>
<td>Sample Source Test Data XML Files for SAP R/3 Readiness Validator</td>
<td>272</td>
</tr>
<tr>
<td>Table 41</td>
<td>SAP R/3 Readiness Validator Global Variables</td>
<td>273</td>
</tr>
<tr>
<td>Table 42</td>
<td>&quot;Create Part&quot; Process Definition Components</td>
<td>276</td>
</tr>
<tr>
<td>Table 43</td>
<td>Action]-[Object] Pairs for tibrvsend Trigger Command</td>
<td>276</td>
</tr>
<tr>
<td>Table 44</td>
<td>SAP R/3 Readiness Validator Rendezvous Configuration</td>
<td>278</td>
</tr>
<tr>
<td>Table 45</td>
<td>&quot;Create Part&quot; Process Definition Components</td>
<td>280</td>
</tr>
</tbody>
</table>
Table 46  Logging_Service Process Definition Components ........................................ 281
Table 47  Logging_Service Input Schema, APIReturnLog ........................................... 282
Table 48  ESI Support of Customized Windchill Objects ......................................... 303
1 Overview

The *Windchill Enterprise Systems Integration Customization Guide* describes how to customize the out-of-the-box implementation of Windchill Enterprise Systems Integration (Windchill ESI). It is intended for developers who are familiar with Windchill, TIBCO and Windchill ESI.

The following documents may be helpful as you Customize ESI:

- Windchill PDM User’s Guide
- Windchill PDMLink User’s Guide
- Windchill Customizer's Guide
- Windchill Enterprise Systems Integration User’s Guide
- Windchill Enterprise Systems Integration Open Application Programming Interface Guide
- TIBCO BusinessWorks Documentation
- TIBCO Enterprise for JMS Documentation
- TIBCO Adapter for SAP R/3 Documentation

This document provides high-level guidance to those considering customizing Windchill ESI. It is not intended to be prescriptive or extensive. Customization should not be undertaken lightly. Customization can have a major impact on technical support, maintenance, and future upgrades. The costs and benefits of a customization should be carefully analyzed with an experienced systems integrator prior to moving forward.

Also, you should discuss the Windchill ESI product roadmap with your PTC representative to determine if future product releases will include the desired functionality.

Customizing Windchill ESI must be viewed from an end-to-end system perspective. Windchill ESI consists of many components organized in a layered architecture, as illustrated in the diagram, below.

Customizations generally cannot be contained to a single component or module, due to the many interrelationships and dependencies between components. In other words, you should carefully consider the downstream effects of any modifications to the standard Windchill ESI product.
1.1 About this Guide

This guide is composed of the following chapters:

- **Section 2, Customization Areas**, describes the detailed steps required to customize the three different areas (Windchill ESI services, EAI software components and Windchill).
- **Section 3, Windchill ESI Services Modules**, describes the java classes and Info*Engine tasks that implement Windchill ESI services.
- **Section 4, EAI Software Components Naming Standards**, describes the standards that have been followed in the EAI software components. It is strongly suggested to follow these standards when customizing the code.
- **Section 5, Windchill ESI EAI Windchill Simulation module**, describes the utilities provided out-of-the-box with the EAI software components that allow users to simulate Windchill when testing the EAI software components by themselves.
- **Section 6, EAI Automated Test Fixture**, describes the utilities provided out-of-the-box with the EAI software components that allow users a mechanism for automating testing using the EAI Windchill Simulation module to mimic the Windchill Open API.
- **Section 7, SAP R/3 Readiness Validation Process**, describes the utilities provided out-of-the-box with the EAI software components that helps users
confirm that a given SAP R/3 distribution target system is correctly configured per documented assumptions and API characterizations.

- Section 8, Windchill ESI Best Practices, provides some guidelines to help minimize problems during customizations.
- Section 9, Examples – Windchill to SAP R/3, provides two examples of end-to-end customizations than can be done in Windchill ESI.
- Section 10, Glossary, provides definitions of terms used throughout the guide
- Section 11, Appendices, describes how customized Windchill Objects are supported in ESI Services

2 Customization Areas
This section describes the points of potential customization in Windchill ESI.

2.1 Windchill ESI Services Customizations
You can add new features to Windchill ESI services. In addition, some Windchill customizations will require you to customize Windchill ESI services. You will find a list of the required Windchill ESI services customization steps in the appendix, Windchill ESI Services Customization Scenarios. This section describes the points of potential customization in Windchill ESI services.

2.1.1 Modify Windchill ESI Properties

2.1.1.1 Background
Windchill ESI properties are read from disk when the Windchill Method Server initializes. They are not reread until the next time the Method Server restarts. The properties define two major types of capabilities: configurable option settings and class extension specification.

A complete description of all Windchill ESI properties can be found in the Windchill Enterprise Systems Integration Installation and Configuration Guide.

Descriptions of the Windchill ESI properties that must be changed when you extend Windchill ESI java class can be found in the Windchill ESI Services Modules section of this document. The procedures to follow when extending a Windchill ESI java class are described in the Extend Windchill ESI Java Code section of this document.

2.1.1.2 Procedure
Windchill ESI properties are defined using the Windchill XCONF feature. See the Windchill Business Administrator's Guide for details about the XCONF files and how to create them.
2.1.1.3 Example: Property File Change

Suppose that you wish to activate the sample map file change which is defined in the Example: Adding an Attribute to <Part> section of this document. In order to activate the new map file, you must:

1. Consult the Windchill ESI Properties and Map Files table below, to determine the name of the property you wish to change.

2. Create an XCONF file that contains the appropriate new value for the property.

3. Use the XCONF Manager to update Windchill ESI properties.

The following XML document is an XCONF command to activate the new part map:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/com/ptc/windchill/esi/esi.properties">
  <Property default="$CODEBASE$/examples/Example1myPart.map" name="com.ptc.windchill.esi.part.partMapFile"/>
</Configuration>
```

XCONF file is located at <Windchill>/codebase/com/ptc/windchill/esi/examples/Example1.properties.xconf.

The following command saves the changes in the file to Windchill ESI properties:

```
xconfmanager -i codebase/com/ptc/windchill/esi/examples/Example1.properties.xconf -p
```

**Caution:** The preceding command causes Windchill ESI services to use the example map file when rendering parts. This changes the structure of the RPC response for GetPart, GetBOM and GetECN. Changing the structure causes the XML schemas in Windchill and the Windchill ESI software components to become out of synch. If you change one of the map files you must modify the XML schema as described in the Create/Modify XML Schema section of this document. You must also modify the EAI software components to use the new schema. See the XML schemas section of this document for details.
2.1.2 Modify Map Files

2.1.2.1 Background

The features of the Windchill ESI mapper are defined in the [ESIMapper section](#) of this document. The Windchill ESI mapper controls the structure of the following RPC response XML elements:

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Map File Location¹</th>
<th>Defining Windchill ESI Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateLink</td>
<td>esipart/ESIAltLink.map</td>
<td>com.ptc.windchill.esi.part.alternateLinkMapFile</td>
</tr>
<tr>
<td>BOMComponent</td>
<td>bom/BOMCom.map</td>
<td>com.ptc.windchill.esi.BOM.component.mapFile</td>
</tr>
<tr>
<td>BOMHeader</td>
<td>bom/BOMH.map</td>
<td>com.ptc.windchill.esi.BOM.header.mapFile</td>
</tr>
<tr>
<td>Document</td>
<td>esidoc/Doc.map</td>
<td>com.ptc.windchill.esi.esidoc.document.mapFile</td>
</tr>
<tr>
<td>DocumentLink</td>
<td>esidoc/DocLink.map</td>
<td>com.ptc.windchill.esi.esidoc.documentLink.mapFile</td>
</tr>
<tr>
<td>ECNHeader</td>
<td>ecn/ChHeader.map</td>
<td>com.ptc.windchill.esi.ECN.changeHeader.mapFile</td>
</tr>
<tr>
<td>Part</td>
<td>esipart/ESIPart.map</td>
<td>com.ptc.windchill.esi.part.partMapFile</td>
</tr>
<tr>
<td>ReferenceLink</td>
<td>bom/RefDes.map</td>
<td>com.ptc.windchill.esi.BOM.refDes.mapFile</td>
</tr>
<tr>
<td>Substitute</td>
<td>bom/Sub.map</td>
<td>com.ptc.windchill.esi.BOM.substitute.mapFile</td>
</tr>
<tr>
<td>N/A²</td>
<td>tgt/ESITarget.map</td>
<td>com.ptc.windchill.esi.tgt.mapFile</td>
</tr>
</tbody>
</table>

Table 1  Windchill ESI Properties and Map Files

All Windchill ESI RPC responses are XML documents. The XML documents are generated by Info*Engine from the contents of an Info*Engine virtual database (VDB). See the [Info*Engine User's Guide](#) for more information about the Info*Engine VDB. Info*Engine generates the XML output in the order that objects were added to the VDB.

Windchill ESI map files define the structure and much of the content of Windchill ESI RPC response output. They do so by telling the Windchill ESI mapper where to obtain data and the order in which to add the obtained data to the Info*Engine element object that is created by the mapper.

¹ All locations are relative to the directory in which ESI Services was installed. ESI Services are, in turn, installed in a directory relative to the directory where Windchill was installed. For example, if Windchill was installed on a Windows system in the directory `c:\ptc`, then ESI Services is installed in the directory, `c:\ptc\Windchill\codebase\com\ptc\windchill\esi`. Therefore, the map file that defines the structure of the `<AlternateLink>` element is `c:\ptc\Windchill\codebase\com\ptc\windchill\esi\esipart\ESIAltLink.map`.

² This map file is used to define the output of the ESI Services StandardESI Service method, `getTargetAssociations()`, which supports the ESI Distribution Targets GUI.
The map files are ASCII text files. The structure of Windchill ESI map files is defined in the Windchill ESI Map section of this document.

The names of the map files used by Windchill ESI services are defined in Windchill ESI properties. You may modify the output of the mapper by changing the format of the map being used.

2.1.2.2 Procedure

When changing a map file, you should copy the existing map file to some other directory that is accessible to the Windchill Method Server. Use the editor of your choice to modify the files. Each entry in the file, as described in the Windchill ESI Map section of this document, generates an XML element in the output group.

Caution: Do not modify the map files that were provided when you installed Windchill ESI. These files may be replaced by future software releases. If you wish to modify them, copy them to a new location.

Define the new map file to Windchill ESI services by changing the appropriate Windchill ESI property. (See the table, above, for a list of properties that define map file locations.)

Use the following command to define the new map file in Windchill ESI properties:

```
xconfmanager -s <property name>=<new map file location> -t codebase/com/ptc/windchill/esi/esi.properties –p
```

Replace `<property name>` with the name of the appropriate property from the table. Replace `<new map file location>` with the location of the map file. You should use the slash (/), not the backslash (\) as the directory separator character in any path name you specify as the value for `<new map file location>`.

Caution: Changing a map file can change the structure of an RPC response. Changing the structure causes the XML schemas in Windchill and the Windchill ESI software components to become out of synch. If you change one of the map files you must modify the XML schema as described in the Create/Modify XML Schema section of this document. You must also modify the EAI software components to use the new schema. See the XML schemas section of this document for details.

2.1.2.3 Example: Adding an Attribute to <Part>

ESIPart.map is the Windchill ESI map file that controls the output for the <Part> element. These are the map file contents:

```
#ESI Part Mapping
```
oid=ObjectID

# force class name
xxx=Class|com.ptc.windchill.esi.Part
modifier=LastChangedBy
number=Number
xxx=StartEffectivity
xxx=EndEffectivity
defaultUnit=DefaultUnit
name=Name
partType=PartType
source=Source
versionInfo.identifier.versionId=Version
iterationInfo.identifier.iterationId=Iteration

The map file is located in <Windchill>/codebase/com/ptc/windchill/esi/esipart.
The map generates an XML element with this structure:

![Default Part XML Element Structure Diagram]

**Figure 2** Default Part XML Element Structure
If the map file is modified to add an additional entry that contains the lifecycle state of the part the map file looks like this:

#Example 1: My Part Mapping

obid=ObjectID

# force class name
xxxx=Class|com.ptc.windchill.esi.Part
modifier=LastChangedBy
number=Number
xxxx=StartEffectivity
xxxx=EndEffectivity
defaultUnit=DefaultUnit
name=Name
partType=PartType
source=Source
versionInfo.identifier.versionId=Version
iterationInfo.identifier.iterationId=Iteration
state.state=State

This map file is located in
<Windchill>/codebase/com/ptc/windchill/esi/examples/Example1myPart.map.

Use the following command to define the map file in Windchill ESI properties:

```
xconfmanager -s com.ptc.windchill.esi.part.partMapFile=<Windchill>/codebase/com/ptc/windchill/esi/examples/Example1myPart.map -t codebase/com/ptc/windchill/esi/esi.properties -p
```

**Caution:** Changing a map file can change the structure of an RPC response. Changing the structure causes the XML schemas in Windchill and the Windchill ESI software components to become out of synch. If you change one of the map files you must modify the XML schema as described in the [Create/Modify XML Schema section](#) of this document. You must also modify the EAI software components to use the new schema. See the [XML schemas section](#) of this document for details.
The Windchill ESI mapper generates the following output if `Example1/myPart.map` is used:

![Figure 3](image)

**Figure 3  Example 1 Customized Part XML Element Structure**

The XML schema definition for the Windchill ESI RPC response that is generated when the customized map file is used is located in `<Windchill>/codebase/com/ptc/windchill/esi/examples/Example1.xsd`.

The Windchill ESI RPC response that is generated when the customized map file is used is located in `<Windchill>/codebase/com/ptc/windchill/esi/examples/Example1.xml`. 
2.1.2.4 Ad Hoc Instance Based Attributes

Windchill PDM allows you to add attributes to modeled classes like wt.part.WTPart without changing the model. You can do this using the Windchill Type Manager. See the *Windchill Business Administrator’s Guide* (or section *Publish Costing Information From Windchill to SAP R/3* of this document) for details. If you extend a modeled class by creating a soft type, or add an attribute to a modeled class, Windchill ESI services obtains the added attribute when you change the appropriate Windchill ESI map file as described in the *previous section of this document*.

Windchill PDM also allows you to add instance based attributes (IBAs) to a modeled class that is an IBAHolder, such as wt.part.WTPart without using the Windchill Type Manager. IBAs added in this way are called ad hoc attributes.

See the *Windchill Application Developer’s Guide* for details about IBAHolder objects.

See the *Windchill User’s Guide* for details about adding IBAs to a wt.part.WTPart.

If you wish to publish ad hoc attributes of wt.part.WTPart objects, you must change the Windchill ESI map and you must modify the Windchill ESI services Info*Engine task that obtains the attributes of WTPart objects. The task must be modified to that it explicitly lists the names of the ad hoc attributes that you want to obtain.

A sample task is distributed with Windchill ESI Services. The task obtains three ad hoc IBAs: StandardPrice, PriceUnit, and PlantStatus. The task, QueryIterationWithIBAandMaster.xml, is located in `<Windchill>/tasks/com/ptc/windchill/esi/examples`.

The recommended procedure to change the task is:

- Copy the sample task to a directory subordinate to the Windchill task root, `<Windchill>/tasks`
- Using your text editor, modify the copied sample task so that the list of attributes includes your ad hoc attribute or attributes. See the comments in the task for instructions on how to make the modification.
- Modify Windchill ESI properties so that Windchill ESI invokes your task instead of the default Windchill ESI services task. Use the following command to modify Windchill ESI properties:

  ```
  xconfmanager -s com.ptc.windchill.esi.part.queryPartTask=com/ptc/windchill/esi/QueryIterationAndMaster.xml -t codebase/com/ptc/windchill/esi/esi.properties -p
  ```

  **Caution:** The sample `xconfmanager` command sets the Windchill ESI properties to the default value. Replace "com/ptc/windchill/esi/QueryIterationAndMaster.xml" with the location and name of your task. The location is relative to the Windchill task root, `<Windchill>/tasks`. 

2.1.3 Create/Modify XML Schema

The Windchill ESI Services RPC output is automatically rendered by a combination of the ESI Mapper, VDB Builder, and Info*Engine SOAP Renderer. The details of this rendering process are described in the Windchill Enterprise Systems Integration Open Application Programming Interface Guide. The function of the XML schema is to describe the format of this automatically generated output to the EAI software components. You would typically need to modify the XML schema if you had performed customizations such as adding an attribute to a Windchill ESI service or adding a new service. See the XML schemas section of this document for details on how to modify an XML schema.

2.1.4 Extend Windchill ESI Java Code

2.1.4.1 Background

Windchill ESI services features are implemented using Info*Engine tasks and java classes. Details about these tasks and classes can be found in the Windchill ESI Services Modules section of this document. These services can be extended using the Windchill Information Modeler, which is described in the Windchill Customizer’s Guide.

Several key features of the Windchill services can also be quickly customized without using the Windchill Information Modeler. The following customization mechanisms are available:

2.1.4.1.1 VDB builder

Windchill ESI RPC responses are XML documents. The documents are generated by Info*Engine from the contents of an Info*Engine virtual database (VDB). See the Info*Engine User's Guide for more information about the Info*Engine VDB. Windchill ESI services provides a set of classes that structure the VDB so that the RPC response output XML is formatted correctly. Details about the VDB builder are located in the VDB Builder section of this document. That section contains examples of things about the VDB builder that you might want to customize.
2.1.4.1.2 Multiple renderers

Renderers are objects that process Windchill objects and convert those objects into Info*Engine Element objects that are managed and formatted by the Windchill ESI VDB builder. (See the VDB Builder section of this document.) Windchill ESI services provide the following renderers:

<table>
<thead>
<tr>
<th>Renderer Interface</th>
<th>Objects Rendered</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECNRenderer</td>
<td>WTChangeOrder2 and all of the changeable objects</td>
</tr>
<tr>
<td>ECNHeaderRenderer</td>
<td>WTChangeOrder2</td>
</tr>
<tr>
<td>BOMRenderer</td>
<td>Windchill product structure</td>
</tr>
<tr>
<td>BOMHeaderRenderer</td>
<td>Windchill part</td>
</tr>
<tr>
<td>BOMComponentRenderer</td>
<td>PartUsageInfo object (which describes a WTPartUsageLink object)</td>
</tr>
<tr>
<td>SubstituteRenderer</td>
<td>PartUsageInfo object (which also describes a WTPartSubstituteLink object)</td>
</tr>
<tr>
<td>ReferenceDesignatorRenderer</td>
<td>PartUsageInfo object (which also describes a UsesOccurrence object)</td>
</tr>
<tr>
<td>PartRenderer</td>
<td>WTPart and any associated WTPartMaster and WTPartAlternateLink objects</td>
</tr>
<tr>
<td>DocumentRenderer</td>
<td>PartDocInfo object (which describes WTDocument WTDocumentMaster, WTPartReferenceLink, WTPartDescribeLink, and EPMDocument objects)</td>
</tr>
</tbody>
</table>

Table 2 Windchill ESI Renders

ECNRenderer is the interface for the GetECN RPC response generator. See the Windchill Enterprise Systems Integration Open Application Programming Interface Guide and the Publishing ECNs section of this document for details about the GetECN RPC. BOMRenderer is the interface for the GetBOM RPC response generator. See the ESI Open Application Programming Interface Guide and the Publishing BOMs section of this document for details about the GetBOM RPC. PartRenderer is the interface for the GetPart RPC response generator. See the Windchill Enterprise Systems Integration Open Application Programming Interface Guide and the Publishing Parts section of this document for details about the GetPart RPC.
2.1.4.1.3 Programming to an interface

Windchill ESI services RPC requests are generated by renderer classes. Each renderer implements an interface that defines the expected behavior of any class that performs the renderer function. For example, any class that is able to generate the output for the GetPart RPC must implement the PartRenderer interface. Windchill ESI services provides a default implementation for each interface. The default implementation classes all permit extensions by customizers.

2.1.4.1.4 Obtaining the interface implementation from a factory

Each interface implementation class is defined in Windchill ESI properties. Windchill ESI services determines at runtime which interface implementation to use. The determination is made by a set of Windchill ESI services java classes called factories. The factory classes are not intended to be customized, but they are intended to facilitate customization.

As someone customizing Windchill Enterprise Systems Integration, you may choose to extend a provided default implementation class or they may choose to write a new implementation class. Either kind of extension can be made with or without the Windchill Information Modeler. These extensions are activated by changing Windchill ESI properties to point to the extension class. See the Modify Windchill ESI Properties section of this document for information about changing Windchill ESI properties. See the Windchill ESI Services Modules section of this document for a description of the property to change when extending a default renderer implementation or providing a new renderer interface implementation class.

2.1.4.1.5 Providing a hook

Each of the default renderer implementation classes produces Info*Engine Element objects. These Element objects are passed to the VDB builder. Each default renderer provides a method, adjustElement(). The method is always called by the default renderer immediately before passing the element to the builder. The default implementation of the method is to do nothing.

The Windchill ESI mapper facility provides access only to the getter methods of the objects that are mapped. The mapper does not provide complex logic capabilities. The adjustElement() method provides a simple way to add or modify the renderer output in cases where map file changes are insufficient to achieve the desired result.
2.1.4.2 Procedure

Procedures for extending Windchill ESI classes using the Windchill Information Modeler are defined in the *Windchill Customizer’s Guide*. For simple extensions of the default interface implementation use this process:

1. Make any map file changes that may be appropriate.
2. Define the java source file using the editor or Integrated Development Environment (IDE) of your choice.

**Caution**: Do not modify or replace any classes that were provided when Windchill ESI was installed. These files may be replaced by future software releases.

3. Compile the java source file into the Windchill codebase.
4. Change the appropriate Windchill ESI property to point to your new renderer implementation.
5. Modify the XML schema and possibly the EAI software components, if your change or add any new elements to the output.

2.1.4.3 Example: Adding an Attribute to <Part> by Extending the Renderer

The default output for the Part XML element is shown in Figure 2. The default part map file is shown in the *Example: Adding an Attribute to <Part>* section of this document.

The new <Part> element now has an additional child element, <Team>.

The modified map file looks like this:

```plaintext
# Example 2: My Part Mapping
obid=ObjectID

# force class name
xxxx=Class|com.ptc.windchill.esi.Part
modifier=LastChangedBy
number=Number
xxxx=StartEffectivity
xxxx=EndEffectivity
defaultUnit=DefaultUnit
name=Name
partType=PartType
source=Source
versionInfo.identifier.versionId=Version
iterationInfo.identifier.iterationId=Iteration
team=Team
```
This map file is located in
<Windchill>/codebase/com/ptc/windchill/esi/examples/Exa2mple2myPart.map.
The Windchill ESI mapper generates the following output if Example1myPart.map is used:

![Diagram of Example 2 Customized Part XML Element Structure]

The mapper may not be able to obtain the required data since a WTPart object does not implement a getTeam() method. As a result, activating the map file results in an empty <Team> element in the RPC response. This could be fixed by either extending WTPart to add a getTeam method, or by extending Windchill ESI’s PartRenderer to fill in the empty team value. The class, Example2MyRenderer, shows how to extend the PartRenderer. The source code for the class is located in
<Windchill>/codebase/com/ptc/windchill/esi/examples/Example2MyRenderer.java.
These are the contents of the sample file:

```java
package com.ptc.windchill.esi.examples;

import com.infoengine.object.factory.Att;
import com.infoengine.object.factory.Element;
import com.ptc.windchill.esi.esipart.ESIPartRenderer;
import com.ptc.windchill.esi.rnd.ESIRendererException;
import java.util.Collection;
import wt.eff.Eff;
import wt.part.WTPart;

public class Example2MyRenderer extends ESIPartRenderer {

    // This example method will adjust the mapper to return a value for the attribute "team".
    // If the attribute "team" is found
    // and the value is null or empty then this method will update the value.

    protected Element adjustPartElement( Element elem,
                                              String group,
                                              WTPart part,
                                              Eff eff,
                                              Collection targets )
                                                  
            throws ESIRendererException {

                Att att = elem.getAtt("Team"); // get the attribute "Team" from the mapper output

                if ( att != null ) { // verify that the attribute exists in the map.
                    // Check to see if the value of "Team" is null or is empty.
                    if ( att.getValue() == null || att.getValue().toString().trim().equals"")
                        {
                            // Get the value for "team" from part object and
                            // add it the value for "Team" in the mapper output.
                            att.setValue(part.getTeamName());
                        }
                }

                return elem;

            }

} // End Example2MyRenderer Class
```

Consider the following:

- The package statement ensures that the output of the java compiler is written to the correct directory.
- The java import statements are required because of the inheritance of the class and the arguments to the `adjustElement()` method.
• The if statement, "if ( att != null )" allows the mapper to control the presence of the team attribute. It does so by attempting to obtain the Att object whose name is "Team" from the mapper output. If there is no map file entry with the name "Team" the result of the attempt to obtain the Att object will be null.

• The if statement, "if ( att.getValue() == null || att.getValue().toString().trim().equals(""))" checks to see if the mapper was able to obtain the desired data. If the data was obtained, nothing is done. This is the recommended way to override the behavior of the mapper.

You can compile the example by doing the following:

• Launch the Windchill shell, from within the <Windchill> directory that was defined when you installed Windchill with this command:

  \textit{bin/windchill shell}

• Use the java compiler to compile the class

  \textit{javac -d ./codebase ./codebase/com/ptc/windchill/esi/examples/*.java}

The following XML document is an XCONF command to activate the new part map and renderer class:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/com/ptc/windchill/esi/esi.properties">
  <!-- override esi.properties.xconf setting to point to example 2 map file -->
  <Property default="$CODEBASE$/examples/Example2myPart.map" name="com.ptc.windchill.esi.part.partMapFile"/>

  <!-- override esi.properties.xconf setting to point to example 2 part renderer -->
  <Property default="com.ptc.windchill.esi.examples.Example2MyRenderer" name="com.ptc.windchill.esi.part.class.PartRenderer"/>
</Configuration>
```

The file is located in <Windchill>/codebase/com/ptc/windchill/esi/examples/Example2.properties.xconf.

The following command saves the changes in the file to Windchill ESI properties:

xconfmanager -i codebase/com/ptc/windchill/esi/examples/Example2.properties.xconf -p

\textbf{Caution:} The preceding command causes Windchill ESI services to use the example map file when rendering parts. This changes the structure of the RPC response for GetPart, GetBOM and GetECN. Changing the structure causes the XML schemas in Windchill and the Windchill ESI software components to become out of synch. If you change one of
the map files you must modify the XML schema as described in the Create/Modify XML Schema section of this document. You must also modify the EAI software components to use the new schema. See the XML schemas section of this document for details.

The XML schema definition for the Windchill ESI RPC response that is generated when the customized map file is used is located in <Windchill>/codebase/com/ptc/windchill/esi/examples/Example2.xsd.

The Windchill ESI RPC response that is generated when the customized map file is used is located in <Windchill>/codebase/com/ptc/windchill/esi/examples/Example2.xml.

2.1.5 Create RPC Classes and Methods

2.1.5.1 Background

Windchill ESI RPC classes and methods are described in detail in the Windchill Enterprise Systems Integration Open Application Programming Interface Guide. The RPC classes use the globally safe namespace, "com.ptc.windchill.esi". Customizers should not modify the RPC classes in this namespace or the methods of those classes. Such modifications are subject to being overlaid by future releases of Windchill ESI. Customizers may modify the output of the RPC classes and methods using the procedures defined in the preceding sections of this document.

2.1.5.2 Procedure

Perform the following steps to define a custom RPC:

1. Define the RPC to Info*Engine using the Delegate Administrator. Details are located in the Info*Engine Administration and Implementation Guide.

2. Create the Info*Engine task that you specified in the previous step that is to be executed when your RPC is requested.

2.1.5.3 Example: Defining an RPC to Mark an Object Published

The StandardESIService has a method, recordExport(), which flags an object as published. The method has a void return type. Exposing the method as a Windchill ESI SOAP RPC requires the following:

- Define an appropriate delegate type identifier. This is the name of the RPC class. The example type identifier is com.ptc.windchill.esi.Example3RPC.

- Create a delegate definition. The example delegate definition is Example3RecordExport.
Windchill ESI services provides an LDIF file which contains the delegate type identifier and delegate definitions. You can import using the Delegate Administrator

\(<\text{Windchill}>/\text{codebase/com/ptc/windchill/esi/examples/Example3.ldif}\).

- Write the task that was defined in the delegate. The example delegate defines the task to be \(\text{com/ptc/windchill/esi/examples/Example3RecordExport.xml}\). When Windchill ESI is installed, the task is stored in the appropriate directory for Info*Engine to execute it. In a typical installation, the directory is \(<\text{Windchill}>/\text{tasks/com/ptc/windchill/esi/examples}\). The task is also stored in \(<\text{Windchill}>/\text{codebase/com/ptc/windchill/esi/examples/Example3RecordExport.xml}\).

### 2.1.6 Modify Difference APIs

Windchill ESI uses the features of Windchill APIs to determine what changes have been made to product structures. Additionally, it uses Windchill Difference APIs to determine what changes have been made to the structure of documents attached to parts. If you want to modify the way changes are identified, you can extend the default Difference APIs.

#### 2.1.6.1 Product Structure Differences

Windchill ESI relies upon the product structure difference logic to determine the changes to the product structure since the part was last published. This difference information identifies what information must be published in order to synchronize the data with your target enterprise systems.

The differences are returned as java enumeration objects pointing to \(\text{wt.part.PartUsageInfo}\) objects. The \(\text{PartUsageInfo}\) class is a non-persistent class that represents the inclusion of a part by another part. In addition to the included part, the object provides access to the substitutes and reference designators specified for the included part.

#### 2.1.6.1.1 Procedure

To customize, extend the default implementation and override specific methods.

Procedures for extending classes using the Windchill Information Modeler are defined in the \(\text{Windchill Customizer’s Guide}\). For simple extensions of the default interface implementation use this process:

1. Define the java source file using the editor or Integrated Development Environment (IDE) of your choice.

   **Caution:** Do not modify or replace any classes that were provided when Windchill ESI was installed. These files may be replaced by future software releases.

2. Compile the java source file into the Windchill codebase.

3. Change the appropriate property to point to the new implementation.
(For more details, see "Customizing service.properties" in the 
Windchill Customization Guide.)

2.1.6.1.2 Example

The following example illustrates how to extend the `wt.part.PartUsageInfo` class and 
modify the default behavior for comparison. The example compares line numbers in 
addition to the default comparisons.

The source code is located at 

These are the contents of the sample file:

```java
package com.ptc.windchill.esi.examples
import wt.part.LineNumber;
import wt.part.PartUsageInfo;
import wt.part.WTPartUsageLink;
import wt.util.WTException;

public class Example4MyPartUsageInfo extends PartUsageInfo
{
    protected boolean isDifferent(PartUsageInfo info)
    {
        boolean result = super.isDifferent(info);

        if (result == false)
        {
            Example4MyPartUsageInfo other = (Example4MyPartUsageInfo)info;
            WTPartUsageLink thisLink = this.getPartUsageLink();
            WTPartUsageLink otherLink = other.getPartUsageLink();
            LineNumber thisLine = thisLink.getLineNumber();
            LineNumber otherLine = otherLink.getLineNumber();

            if (thisLine == null)
            {
                if (otherLine != null)
                {
                    // different line number
                    result = true;
                }
            }
            else
            {
                if (otherLine == null ||
                    thisLine.equals(otherLine) == false)
                {
                    // different line number
                    result = true;
                }
            }
        }
    }
}
```

Windchill Enterprise Systems Integration Customization Guide
Consider the following:

- The package statement ensures that the output of the java compiler is written to the correct directory.
- The java import statements are required by the class’ inheritance and implementation specifics.

You can compile the example by doing the following:

- Launch the Windchill shell, from within the `<Windchill>` directory that was defined when you installed Windchill PDM or Windchill PDMLink with this command:

  ```bash
  bin/windchill shell
  ```

- Use the java compiler to compile the class.

  ```bash
  javac -d ./codebase ./codebase/com/ptc/windchill/esi/examples/*.java
  ```

To activate the new implementation, modify `wt.properties` to specify a property file that specifies the service implementation for `wt.part.PartDocInfo`.

```properties
wt.services.applicationcontext.WTServiceProviderFromProperties.
customPropertyFiles
```

Append the name of a property file that contains the specification for the new implementation, using a comma to separate the file names. There is an example property file (`Example4.properties`) that contains the specification for the example implementation.

```properties
duplicate
```

### 2.1.6.2 Document Attachment Differences

Windchill ESI relies upon the Document Attachment Difference logic to determine the changes to the document structure since a part was last published. This difference information identifies what information must be sent in order to synchronize the data with your ERP systems.

The differences are returned as enumerations of `com.ptc.windchill.esi.PartDocInfo` objects. The `PartDocInfo` class is a transient class that represents the attachment of a document to a part.
There are several different associations that represent documentation for a part. These document associations include the following:

- `wt.doc.WTDocument` associated with the `wt.part.WTPart` via a "Described By" association
- `wt.doc.WTDocumentMaster` associated with the `wt.part.WTPart` via a "References" association
- `wt.epm.EPMDocument` associated with `wt.part.WTPart` via a "Build History" association
- `wt.epm.EPMDocument` association with `wt.part.WTPartMaster` via a "Described By" association

### 2.1.6.2.1 Procedure

Procedures for extending classes using the Windchill Information Modeler are defined in the *Windchill Customizer’s Guide*. For simple extensions of the default interface implementation use this process:

1. Define the java source file using the editor or Integrated Development Environment (IDE) of your choice.

   **Caution:** Do not modify or replace any classes that were provided when Windchill ESI was installed. These files may be replaced by future software releases.

2. Compile the java source file into the Windchill codebase.

3. Change the appropriate property to point to the new implementation.

   (For more details, see "Customizing service.properties" in the *Windchill Customizer’s Guide").

### 2.1.6.2.2 Example

The following example illustrates how to extend the `com.ptc.windchill.esi.esidoc.PartDocInfo` class and modify the default behavior for generating document URLs. Instead of generating a URL to the document properties page, this example generates a URL to the document content.

The source code is located at `<Windchill>/codebase/com/ptc/windchill/esi/examples/Example5MyPartDocInfo.java`.

These are the contents of the sample file:

```java
package com.ptc.windchill.esi.examples;

import com.ptc.windchill.esi.esidoc.PartDocInfo;

import java.util.HashMap;

import java.util.HashMap;

import wt.doc.Document;
```
import wt.fc.ObjectIdentifier;
import wt.fc.Persistable;
import wt.fc.PersistenceHelper;

import wt.httpgw.GatewayServletHelper;
import wt.httpgw.URLFactory;

import wt.util.WTException;

public class Example5MyPartDocInfo extends PartDocInfo {
    public String getDocumentURL()
    {
        String url = null;
        try{
            // Generate a URL to download the document content.
            URLFactory urlFactory = new URLFactory();
            HashMap map = new HashMap();
            map.put("action", "DownloadContent");
            map.put("oid", getDocumentOID());

            url = GatewayServletHelper.buildAuthenticatedHREF(
                    urlFactory, "wt.enterprise.URLProcessor", "URLTemplateAction", map);
        }
        catch (WTException e){
            // Use the default logic which generates a URL to display
            // the document properties.
            url = super.getDocumentURL();
        }

        return url;
    }

    private String getDocumentOID()
            throws WTException
    {
        Persistable p = (Persistable) getDocument();
        ObjectIdentifier oid = PersistenceHelper.getObjectIdentifier(p);

        return oid.getStringValue();
    }
}
Consider the following:

- The package statement ensures that the output of the java compiler is written to the correct directory.
- The java import statements are required by the class’ inheritance and implementation specifics.

You can compile the example by doing the following:

- Launch the Windchill shell, from within the `<Windchill>` directory that was defined when you installed Windchill with this command:
  
  ```
  bin/windchill shell
  ```

- Use the java compiler to compile the class.
  
  ```
  javac -d ./codebase ./codebase/com/ptc/windchill/esi/examples/*.java
  ```

To activate the new implementation, modify `wt.properties` to specify a property file that specifies the service implementation for `com.ptc.windchill.esidoc.PartDocInfo`.

```
wt.services.applicationcontext.WTServiceProviderFromProperties.
customPropertyFiles
```

Append the name of a property file that contains the specification for the new implementation, using a comma to separate the file names. There is an example property file (`Example5.properties`) that contains the specification for the example implementation.

```
wt.services/svc/default/com.ptc.windchill.esi.esidoc.PartDocInfo/null
/java.lang.Object=0=com.ptc.windchill.esi.examples.
Example5MyPartDocInfo/duplicate
```

### 2.1.7 Transaction Management and Error Handling

#### 2.1.7.1 Background

When a request to publish a Windchill object is received, many objects may be created in multiple distribution targets. The purpose of the Windchill ESI transaction management components in Windchill is to:

- Perform automated validation of packages to be published
- Identify what objects to publish, and to where they should be published, based on a single request
- Accept and record status messages on the success or failure of each attempted object publication
- Manage user interaction with the publication process, including manual error handling
• Ensure the overall publication process is handled in a closed-loop fashion so that all objects are successfully published
Windchill ESI services transaction management comprises the following:

- Windchill ESI workflow templates
  
  Instructions for customizing workflows can be found in the *Windchill Business Administrator's Guide*.

- Windchill ESI RPCs
  
  Instruction is located in the [Create RPC Classes and Methods section](#) of this document

- Windchill ESI services java code
  
  Instructions for modifying Windchill ESI services java code can be found in the [Extend Windchill ESI Java Code section](#) of this document

### 2.2 EAI Software Components Customizations

#### 2.2.1 XML (eXtended Markup Language) schemas

This section describes how to customize XML schemas. Windchill ESI defines all Windchill ESI schemas via XSDs (eXtensible Schema Definition). XSDs were chosen over DTD (Document Type Definition), due to their semantic richness, scalability, and extensibility advantages. Customizers may want to customize XML schemas to add or change attributes that are being sent to SAP R/3 or to modify existing PRC calls.

**EAI Software Components Schemas:**

**Data Model Schemas** (CBO and ESIResponse): Change these schemas to add or change attributes coming from Windchill PDM or Windchill PDMLink.

**Internal EAI Schemas** (ESIResultService, ESIErrorHandling, ESILogging): Should only be changed if the core architecture of the solution is being changed. Otherwise, these would not be modified as part of a standard implementation.

**Windchill Communication Schemas** (ESIEvent, ESIGetPart, ESIGetBOM, ESIGetECN, ESIPostResult, ESIPostEvent, ESIResponse): Modify these schemas if the communication between Windchill PDM and the EAI software components needs to be modified. This includes updating an RPC or adding a new one.

Here are some example scenarios:

- **Scenario 1: Add a new attribute to an object**
  
  - **Description:** Update the GetPart RPC to return a new attribute. Thus, the request to Windchill from the EAI software components looks the same. However, the data coming back from Windchill contains an extra attribute for Part.
  
  - **Customization Steps for EAI software components:** Modify the ESIResponse schema to include this new attribute (The CBO would also need to be modified if this parameter is to be passed to the ERP system)
• Scenario 2: Add a new flag to the RPC
  - Description: Update the GetPart RPC to include a new flag called Iteration to send iteration information if this flag is set to true. Out-of-the-box the Getpart RPC only has two flags: Alternates and Documents.
  - Customization Steps for EAI software components: Modify the ESIGetPart schema to include the new flag. If more data may be returned from Windchill, the ESIResponse and CBO schemas may also need to be modified.

• Scenario 3: Adding a new RPC
  - Description: Add a new RPC called GetDocuments to be able to query for document only releases.
  - Customization Steps for EAI software components: A new schema would need to be created, for example ESIGetDocument. Again, if more data should be returned from Windchill, the ESIResponse and CBO Schemas may also need to be modified.

**Note:** All of the above would require updating process definitions in TIBCO BusinessWorks.

### Flow of Windchill Interaction Schemas

An interaction with Windchill includes several schemas that are sent back and forth over JMS. These schemas are utilized in the following order:

<table>
<thead>
<tr>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESIEvent</td>
<td>This schema is the initial communication between Windchill and EAI software components. It contains some basic information, such as a TransactionID.</td>
</tr>
<tr>
<td>Request (ESIGetPart, ESIGetBOM, ESIGetECN)</td>
<td>When the EAI software components receive the Event, it returns a Request schema to Windchill that asks for a release of the business data to be published to SAP R/3.</td>
</tr>
<tr>
<td>ESIResponse</td>
<td>When Windchill receives the EAI software components' Request, it sends back a Response schema. This contains the data that is published to SAP R/3 by the EAI software components.</td>
</tr>
<tr>
<td>ESIPostResult</td>
<td>When the EAI software components process data, it sends a PostResult message back to Windchill for each business object in each organization that was processed with a success or failure flag. This allows Windchill to track which objects were successfully published and which require resubmit.</td>
</tr>
<tr>
<td>ESIPostEvent</td>
<td>This is the summary of all the PostResult messages that were sent back to Windchill. If all PostResult messages were successful, then this message contains success. Otherwise, it contains false.</td>
</tr>
<tr>
<td>ESIResultResponse</td>
<td>This is Windchill’s confirmation to the EAI software components.</td>
</tr>
</tbody>
</table>
components that the transaction has completed.

**Tag Naming Conventions**

Tags within Windchill ESI XML follow leading-capitalization notation, in which words are concatenated together and the first letter of each is capitalized, for example, PartNumber.

It is recommended for maximum clarity that XML tag names be given names that are as descriptive as possible. However, shorter names may be given to tags when the parent node or context of the tag is clear. For example, a <Number> tag within a Part structure should be taken as the Part Number. If the Part Number was a child of another object, such as Document Link, then the field would require at a minimum the string "DocumentNumber." It is recommended that in this case a more descriptive name be given to the field, such as "AssociatedDocumentNumber."

For more information see Section 4 – Naming Standards

**SOAP Guidelines**

SOAP is the XML RPC protocol that is used to communicate between the EAI software components and Windchill. All JMS communications are SOAP-encoded.

**ESIEvent, ESIPostResult, ESIPostEvent, ESIGetPart, ESIGetBOM, ESIGetECN**

All XML in the document outside the <arguments> tags must remain static. The XML structure inside these tags may be modified if the RPC in Windchill is modified equivalently.

In the case of the Request, the <GetResponse> tag varies according to the type of object being processed. The pairs are:

- `<GetBOMResponse></GetBOMResponse>`
- `<GetPartResponse></GetPartResponse>`
- `<GetChangeResponse></GetChangeResponse>`

**ESIResponse, ESIResultResponse**

All XML in the document outside the <COLLECTION> tags must remain static. The XML structure inside these tags may be modified if the RPC in Windchill is modified equivalently.

The COLLECTION object in the Response and all its child structures use HTML notation for their "<" and ">" – "&lt;" and "&gt;" respectively. Info*Engine creates well-formed XML documents; however, for security purposes, the JAX-M SOAP classes assume that any XML-reserved characters (" " < > & ) are to be converted to the
corresponding XML entities ("\&quot; "\&lt;" etc.). Thus the body of the Response contains the HTML notation for these characters.

**How to edit and import schemas**

Schemas may be edited using any text editor, such as Windows Notepad, or XML editor, such as TIBCO TurboXML.

To view the main CBO schema, the schemas need to be extracted from BusinessWorks. CBO schemas are located in the /SharedConfigurations/ESISchemas/EAIMiddlewareSchemas folder. Since the CBO is made up of many referencing schemas, all of the referenced schemas must be loaded in order to view the high level objects.

1. Click on the Source Tab of each schema, copy the contents and paste them into a text editor. Save the file as the schema name that appears in BusinessWorks with a .xsd extension.
2. Each of the structures can be opened in Turbo XML.
3. When viewing a higher level object - one that references other schemas - Turbo XML prompts for the location of the referenced schemas. Browse to the location and click OK. This prompting continues until all the referenced schemas have been located.

Consider the following:

- Some of the schemas reference another schema more than once. For example, the BOM schema references the ObjectHeader and Effectivity schemas more than once. In this case, Turbo will generate a warning message that says "This element type has been defined more than once. The declaration can be ignored"
- When looking at a schema that references another schema, the referenced schema will appear in BW with warning icon next to the element. This simply indicates that the element is locked and thus, you cannot modify the element. You can unlock the element by right clicking on the lock and selecting unlock. This will now allow you to modify the referenced schema as well. Refer to the TurboXML documentation for more information.
- It is strongly recommended that the CBO be modified using the UserArea schemas that have been created for each object as explained in the following section. Modifying the main CBO schema can lead to fatal errors.

To import a schema into BusinessWorks, add a "Schema Definition" activity to the repository. In the Source tab of the activity, choose "Load Schema." TIBCO BusinessWorks validates the XML at the time it is imported to the repository.

**2.2.2 Common Business Object (CBO)**

The CBO leverages key concepts from the Open Application Group Interface Standard (OAGIS). OAGIS is an XSD-based XML schema specifically designed
for enterprise system communication. It is a self-describing data model that is used to pass enterprise data between ERP, CRM, and other enterprise-level systems.

The CBO is a self-describing XSD-based XML schema containing separate, self-describing business objects - an OAGIS concept. The Windchill Response model describes each object, relationship, and action separately. There is no organization of objects and their relationships. For example, for BOMS, there are four elements for BOM header information - one for each action on BOMs: UnchangedBOMs, ChangedBOMs, etcetera - and separate elements for each relationship type - Children, Substitutes, and Reference Designators (each with separate elements per action: AddedSubstitutes, DeletedSubstitutes, etc.)

The CBO is extensible to create new objects and new attributes on existing objects via customization. Each business object contains a UserArea element that supports customization and cleanly separates user-defined data from the Windchill ESI out-of-the-box data model.

To create a schema for a given UserArea perform the following steps:
1. Determine the schema for the UserArea
2. Create the .XSD file. Schemas may be created using any text editor, such as Windows Notepad, or XML editor, such as TIBCO TurboXML
3. In BusinessWorks, select the UserArea that has been modified. In the TIBCO Designer GUI, the CBO schemas are located in SharedConfigurations/ESISchemas/EAIMiddlewareSchemas
4. In the Configuration tab, enter the appropriate Location and choose Load Schema. Once the new schema has been loaded, click Apply.

2.2.3 Business logic in BusinessWorks

Before making changes to the Windchill ESI BusinessWorks application, customizers should be familiar with how the code is structured. This helps ensure that customizations are consistent with the overall Windchill ESI design and efficiently implemented. To assist in localizing problems during technical support activities, customizations should be compartmentalized as much as possible.

Functionality
The Windchill ESI BusinessWorks application performs several primary functions:

- Exchange messages across the JMS interface with Windchill
- Transport business data from Windchill to the SAP R/3 distribution targets
- Transform the data as required by the SAP R/3 distribution targets
- Apply formulas and functions, such as padding object numbers or concatenating values for keys required by SAP R/3
- Supplement missing data with pre-defined default values
- Covert list-of-value fields via cross-referencing
- Stage and transmit data for the required sequence of API calls to the SAP R/3 distribution targets
- Translate the data from XML schemas to SAP R/3’s native format (performed by the Adapter for R/3)
- Transaction management: Maintaining the integrity of the overall release transaction and its component subtransactions
- Error-handling
- Logging
- Return results and error messages to Windchill

**Mapping and Filtering**
Data mapping between the systems is accomplished in two main steps:

- Mapping from the Windchill ESI data model to the Common Business Object (CBO) schema
- Mapping from the CBO schema to the SAP R/3 API schemas

The latter mapping requires several stages of filtering, as illustrated below:

- The Master Process Flow process determines the correct overall order of business object processing for the publishing transaction
- Each business object sub-process determines the correct sequence of delete, add, and change operations and iterates through the data for each
- Each business object sub-process invokes the required SAP R/3 APIs in the correct sequence for each object iteration
ESI BusinessWorks Application
Post-CBO Filtering Layers

Master Process Flow (Object Processing Order)

<table>
<thead>
<tr>
<th>ECN</th>
<th>Part</th>
<th>BOM</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Add</td>
<td>Change</td>
<td>Delete</td>
</tr>
<tr>
<td>R/3 API</td>
<td>R/3 API</td>
<td>R/3 API</td>
<td>R/3 API</td>
</tr>
</tbody>
</table>

Figure 5  Post-CBO Filtering Layers

Code Structure
Overall, the Windchill ESI BusinessWorks application consists of a "Master Process Flow" and several subprocesses oriented around each Windchill ESI business object: ECN, Part, BOM, and Document. The Master Process Flow controls most JMS communications with Windchill, the overall sequence of object processing, and transaction management. Within each sub-process, some functionality is stubbed for future use or customization, such as object deletion and change ECN operations.

Note: After the four main object sub-processes execute, the Master Process Flow calls on a special, fifth sub-process to update temporary ECN effectivity dates established during the initial ECN creation. All processes can make inline calls to shared service routines for defaults, crossreferencing, error handling, and logging wherever they are needed.
Technical Architecture

Files used by the EAI software components (FilesToRead.properties, ESIDefaults.properties, ESIEmailMessageLookups.properties, ESIErrorHandlingCodes.properties, ESILookups.properties, ESIMessageLookups.properties) must be stored in UTF-8 encoded format. Systems integrators or customizers must use care when configuring Windchill ESI to save these files as UTF-8 files, rather than standard ASCII files, which is the default format used by many text editors. Microsoft Windows users are advised to use Microsoft Notepad rather than Microsoft WordPad. UNIX users should investigate the current status of UTF-8 support in their text editors, as emacs and vi have been slow to adopt UTF-8 standards.

The header of each file contains detailed information as to its proper usage and how to update it.

2.2.4 SAP R/3 API descriptions and characterization

The SAP R/3 APIs used by Windchill ESI were selected after an exhaustive analysis of business requirements and available SAP R/3 functionality (see list below). The use and characterization of these APIs are critical Windchill ESI implementation assumptions. Any updates to APIs require re-characterization, that is, re-testing to ensure that the appropriate values are entered for each field. Some examples of customizations to APIs include, but are not limited to:

a. Increase Data / different data passed to the out-of-the-box APIs
b. Utilizing a different API than the out-of-the-box APIs
c. Modify API in ERP
d. Stringing APIs together to perform a particular function
Examples of increasing data passed to an out-of-the-box API can be seen in the Examples - Windchill to SAP R/3 section of this document.

Windchill ESI utilizes the following SAP R/3 APIs (grouped by Windchill ESI function):

### Create/Change Part

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAPI_MATERIAL_SAVEDATA</td>
<td>This API is used to create/change all material data in SAP R/3. The API needs to be called for each material to be created in each plant. The API does not process multiple materials in multiple organizations with one API invocation</td>
</tr>
<tr>
<td>CCAP_REV_LEVEL_MAINTAIN</td>
<td>This API creates a new revision level for the material created with BAPI_MATERIAL_SAVEDATA. The revision level creation is tracked with a change number defined from Windchill. The change master record for the change number must exist in SAP R/3 prior to running this API.</td>
</tr>
<tr>
<td>BAPI_TRANSACTION_COMMIT</td>
<td>This API is used to commit a transaction to the SAP R/3 system. If the return message generated from the BAPI_MATERIAL_SAVEDATA execution is a success then this API is executed. Upon execution of the commit API all part data sent from Windchill is committed to the MARA (and MARC if plant data sent) SAP R/3 database tables.</td>
</tr>
<tr>
<td>BAPI_TRANSACTION_ROLLBACK</td>
<td>This API is used to rollback a transaction to the SAP R/3 system. If the return message generated from the BAPI_MATERIAL_SAVEDATA execution is not a success then this API is executed. Upon execution if the rollback API all part data sent from Windchill is rolled back and not committed to the MARA (and MARC if plant data sent) SAP R/3 database tables.</td>
</tr>
</tbody>
</table>
Create Part API Flow

Change Part API Flow
Create BOM

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSAP_MAT_BOM_CREATE</td>
<td>This API is invoked to create BOMs in SAP R/3. This API cannot be called for multiple BOMs in multiple organizations.</td>
</tr>
<tr>
<td>CALO_INIT_API</td>
<td>It is necessary to call this API before the CSAP_MAT_BOM_CREATE is called. It is not necessary to pass any data into this API. The defaults written into the API upon execution meet the Windchill ESI requirements.</td>
</tr>
<tr>
<td>NOTE:</td>
<td>All values are defaulted in upon execution of the API. It is not necessary to pass data to this API from default tables</td>
</tr>
</tbody>
</table>

Create BOM API Flow

![Create BOM API Flow Diagram](image)

Change BOM

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSAP_MAT_BOM_MAINTAIN</td>
<td>This API is invoked to change BOMs in SAP R/3. This API cannot be called for multiple BOMs in multiple organizations.</td>
</tr>
<tr>
<td>CALO_INIT_API</td>
<td>It is necessary to call this API before the CSAP_MAT_BOM_MAINTAIN is called to change a BOM component. It is not necessary to pass any data into this API. The defaults written into the API upon execution meet the Windchill ESI requirements.</td>
</tr>
<tr>
<td>NOTE:</td>
<td>All values are defaulted in</td>
</tr>
</tbody>
</table>
Change BOM API Flow

Create Document Info Record

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAPI_DOCUMENT_CREATE2</td>
<td>This API is used to create all document data in SAP R/3. The API needs to be called for each document to be created. The API does not require plant information.</td>
</tr>
<tr>
<td>BAPI_DOCUMENT_CHANGE2</td>
<td>This API is used to create and delete the links between documents and parts in SAP R/3. This API does not require plant information.</td>
</tr>
<tr>
<td>BAPI_TRANSACTION_COMMIT</td>
<td>This API is used to commit a transaction to the SAP R/3 system. This API is called if the return message generated from the BAPI_DOCUMENT_CREATE2 execution is a success and also if the return message generated from the BAPI_DOCUMENT_CHANGE2 execution is a success. Upon execution of the commit API all document data</td>
</tr>
</tbody>
</table>
sent from Windchill is committed to the DRAW, DRAK and DRAT SAP R/3 database tables

| BAPI_TRANSACTION_ROLLBACK | This API is used to rollback a transaction to the SAP R/3 system. This API is called if the return message generated from the BAPI_DOCUMENT_CREATE2 execution is not a success and also if the return message generated from the BAPI_DOCUMENT_CHANGE2 execution is not a success. Upon execution of the rollback API all part data sent from Windchill is rolled back and not committed to the DRAW, DRAK, and DRAT SAP R/3 database tables |

Create Document Info Record API Flow

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAPI_DOCUMENT_CHANGE2</td>
<td>This API is used to change all document data in SAP R/3. The API needs to be called for each document to be changed. The API does not require plant information. The API also creates and deleted the links between documents and other objects in SAP R/3.</td>
</tr>
<tr>
<td>BAPI_DOCUMENT_CREATENEWVRS2</td>
<td>This API is necessary to create a new document version</td>
</tr>
<tr>
<td>BAPI_TRANSACTION_COMMIT</td>
<td>This API is used to commit a</td>
</tr>
</tbody>
</table>
transaction to the SAP R/3 system. This API is called if the return message generated from the BAPI_DOCUMENT_CREATENEW VRS2 execution is a success and also if the return message generated from the BAPI_DOCUMENT_CHANGE2 execution is a success. Upon execution of the commit API all document data sent from Windchill is committed to the DRAW, DRAK and DRAT SAP R/3 database tables.

**BAPI_TRANSACTION_ROLLBACK**

This API is used to rollback a transaction to the SAP R/3 system. This API is called if the return message generated from the BAPI_DOCUMENT_CREATENEW VRS2 execution is not a success and also if the return message generated from the BAPI_DOCUMENT_CHANGE2 execution is not a success. Upon execution of the rollback API all part data sent from Windchill is rolled back and not committed to the DRAW, DRAK, and DRAT SAP R/3 database tables.

**Change Document Info Record API Flow**

*Change Document Info Record API Flow*

<table>
<thead>
<tr>
<th>Create ECN</th>
<th>CCAP_ECN_CREATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>CCAP_ECN_CREATE</td>
<td>This API is used to create an ECN in SAP R/3 Release 4.6C. The API is characterized to create an ECN that is</td>
</tr>
</tbody>
</table>
valid for the create or change functions of Documents, BOMs and Materials. It is also characterized to list Material Objects on the ECN with Alternate Effective Dates set to the current date. The Header Effective is initially set to a future Effective Date and the ECN status is set to Active. **Note:** BOMs and Documents are not listed on the ECN as part of the ECN create process. They are listed on the ECN automatically when they are created or changed with the ECN. The process then goes to update the ECN (by calling `CCAP_ECN_MAINTAIN`) and assign the correct effectivity dates.

| **CCAP_ECN_MAINTAIN** | This API is used to change an ECN in SAP R/3 Release 4.6C. This API is called after the ECN is created and used for the creations or change functions of BOMs or Documents. The API is used to Create Alternate Dates and to assign those Alternate Dates to Document and BOMs objects that are listed on the ECN. Additionally, this API is used to set the ECN Header Effective Date to the Current Date and Set the ECN Status to Inactive |

---

**Create ECN API Flow**
2.2.5 SAP R/3 API Response Characterization

This section describes SAP R/3’s response and the interpretation by EAI software components.

<table>
<thead>
<tr>
<th>API</th>
<th>Response Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAPI_MATERIAL_SAVEDATA</td>
<td>Windchill ESI interprets the transaction as being <strong>successful</strong> when the following</td>
</tr>
<tr>
<td></td>
<td>conditions exist for the last return message table entry (values are case-sensitive):</td>
</tr>
<tr>
<td></td>
<td>RETURN-ID = ‘MM’</td>
</tr>
<tr>
<td></td>
<td>and</td>
</tr>
<tr>
<td></td>
<td>RETURN-NUMBER = ‘356’</td>
</tr>
<tr>
<td></td>
<td>If the conditions for a successful transaction – as specified above - are not met, the</td>
</tr>
<tr>
<td></td>
<td>application will interpret the transaction as being <strong>unsuccessful</strong>.</td>
</tr>
<tr>
<td>CCAP_REV_LEVEL_MAINTAIN</td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is</td>
</tr>
<tr>
<td></td>
<td>generated. See Unsuccessful Condition Below.</td>
</tr>
<tr>
<td></td>
<td>Unsuccessful Condition: An Exception Error is generated. A Business Works error is</td>
</tr>
<tr>
<td></td>
<td>generated and has the effect that a &quot;_error&quot; variable is created and it generates an</td>
</tr>
<tr>
<td></td>
<td>error.</td>
</tr>
</tbody>
</table>
transition out of the adapter invocation task.

<table>
<thead>
<tr>
<th>Function</th>
<th>Successful Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAPI_TRANSACTION_COMMIT</td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
<tr>
<td>BAPI_TRANSACTION_ROLLBACK</td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
<tr>
<td>CSAP_MAT_BOM_CREATE</td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
<tr>
<td>CSAP_MAT_BOM_MAINTAIN</td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
<tr>
<td>CALO_INIT_API</td>
<td>Always assumed successful</td>
</tr>
</tbody>
</table>
| BAPI_DOCUMENT_CREATE2           | Windchill ESI interprets the transaction as being **successful** when the following condition exist for the last return message table entry (values are case-sensitive):  

\[
\text{RETURN-TYPE} \neq \text{\\'E\\'} \\
\text{and} \\
\text{RETURN-TYPE} \neq \text{\\'A\\'}
\]

If the conditions for a successful transaction – as specified above - are not met, the application interprets the transaction as being **unsuccessful**. |
| BAPI_DOCUMENT_CHANGE2           | Windchill ESI interprets the transaction as being **successful** when the following condition exist for the last return message table entry (values are case-sensitive):  

\[
\text{RETURN-TYPE} \neq \text{\\'E\\'} \\
\text{and} \\
\text{RETURN-TYPE} \neq \text{\\'A\\'}
\]

If the conditions for a successful transaction – as specified above - are not met, the application interprets the transaction as being **unsuccessful**. |
| BAPI_DOCUMENT_CREATENEW VRS2 | Windchill ESI interprets the transaction as being **successful** when the following condition exist for the last return message table entry (values are case-sensitive):

\[
\text{RETURN-TYPE} \neq 'E' \\
\text{and} \\
\text{RETURN-TYPE} \neq 'A'
\]

If the conditions for a successful transaction – as specified above - are not met, the application interprets the transaction as being **unsuccessful**. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CCAP_ECN_CREATE</strong></td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
<tr>
<td><strong>CCAP_ECN_MAINTAIN</strong></td>
<td>Successful Condition: A successful condition is assumed unless an Exception Error is generated.</td>
</tr>
</tbody>
</table>
2.2.6 Internationalization (I18N) & Localization (L10N)

Windchill ESI supports 8 languages and 10 locales. It supports UTF-8 encoding for Unicode and multi-byte characters. Built-in localization (L10N) features are included for the following locales:

- English (U.S.)
- English (U.K.)
- French
- German
- Italian
- Spanish
- Japanese
- Chinese (Simplified)
- Chinese (Traditional)
- Korean

For more information on how to localize the out-of-the-box Windchill ESI solution, refer to the Enterprise Systems Integration Installation and Configuration Guide.

If you must support a different locale, in addition to the steps described in the Enterprise Systems Integration Installation and Configuration Guide, you may have to address:

- SAP R/3 code pages
- Windchill support
- Operating System support (for logging)
- The com_infoengine_locale attribute in the JMS header of Windchill ESI messages
- The ESIR3Locale global variable
- Locale encoding in the Adapter for R/3 configuration
- TIBCO product support (if developing in a language other than English).

2.3 Windchill Customizations

2.3.1 Attribute Length

Several important business object attributes have different standard lengths in Windchill and SAP R/3 (see Attribute Lengths table). Windchill ESI assumes that your business process limits the Windchill attributes to the lengths allowed by SAP R/3. If the attributes sent are longer allowed by SAP R/3, BusinessWorks throws an exception. To enforce the length assumptions systematically, systems integrators/customizers can add validation programming logic to Windchill or modify SAP R/3 to extend the attribute lengths.
<table>
<thead>
<tr>
<th>Windchill Attribute</th>
<th>Windchill Attribute Length</th>
<th>SAP R/3 Attribute</th>
<th>SAP R/3 Attribute Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Part) Number</td>
<td>32</td>
<td>Material Number</td>
<td>18</td>
</tr>
<tr>
<td>(Part) Name</td>
<td>60</td>
<td>(Material) Description</td>
<td>40</td>
</tr>
<tr>
<td>Doc Number</td>
<td>32</td>
<td>Document Number</td>
<td>25</td>
</tr>
<tr>
<td>(Doc) Name</td>
<td>2000</td>
<td>(Document) Description</td>
<td>40</td>
</tr>
<tr>
<td>Number (Change Number)</td>
<td>20</td>
<td>Change Number</td>
<td>12</td>
</tr>
<tr>
<td>(ECN) Name</td>
<td>70</td>
<td>(Change Number) Description</td>
<td>40</td>
</tr>
<tr>
<td>Description (Change Number Description)</td>
<td>2000</td>
<td>Reason for Change</td>
<td>40</td>
</tr>
</tbody>
</table>

**Table 3   Attribute Lengths**

For more information on how to limit Attribute Lengths in Windchill, refer to the "Customizing Column Lengths" section of the *Windchill Customizer’s Guide*.

## 3 Windchill ESI Services Modules

This section describes the objects that comprise Windchill ESI services. The section includes information about the objects' interface and how the objects collaborate to provide the capabilities of Windchill ESI services. In addition, the section defines the customization points and configuration options that are available in Windchill ESI Services.

### 3.1 Windchill ESI Mapper

#### 3.1.1 Mapper Classes

Windchill ESI mapping capabilities are provided by the classes shown in Figure 7. None of the classes are serializable. Therefore, they are not available to Windchill GUI clients.
3.1.1.1 ESIMapManager

ESIMapManager maintains a collection of ESIMap objects. The map manager maintains the ESIMap objects in a cache so that they only have to be retrieved once from external storage. Once retrieved, the map files are not re-read from disk until the java virtual machine is restarted.

The class follows the Singleton design pattern to ensure that only one instance exists in a java virtual machine. All users of ESIMap objects are expected to obtain them from the map manager.

3.1.1.1.1 <<Static>> instance() : ESIMapManager

Gets the value of the _instance attribute. If it's null, create an instance and store it in the attribute before returning the result.

3.1.1.1.2 <<final>> getMap(name : String) : ESIMap

Returns the ESIMap identified by the name argument. If there is an entry in the maps Hashtable, return it. Otherwise, create a new ESIMap, store it in the table, and return the result.
3.1.1.2 ESIMap

Windchill ESI maps are objects that represent the contents of Windchill ESI map files. Windchill ESI Map files are ASCII text files. Each entry in the file is expected to have the following format:

Source_Name=Target_Name|Default

- **Source_Name** is an instruction describing where to find the data that is to be extracted. This section of the entry is required.
  - If the object being mapped is an Info*Engine element object, **Source_Name** is the name of an Info*Engine Att object on that element object.
  - If the object being mapped is not an Info*Engine element, **Source_Name** is the name of a "getter method" to invoke on the object. The value supplied for **Source_Name** is pre-pended with the characters, "get", and the resulting string is used as a method name. The mapper only invokes methods that take no arguments and have a name that begins with "get".
  - The mapper is able to navigate object structures. If **Source_Name** contains periods, the periods are interpreted as method delimiters. The mapper invokes the "getter method" identified first segment of the **Source_Name** on the object being mapped. If that method returns a result, the mapper invokes the second segment of the **Source_Name** on the result. The mapper continues to obtain the result for each section of **Source_Name**.

  Example: This is the **Source_Name** value from one of the map files used by the Windchill ESI BOM renderer: Quantity.Amount. The mapper executes the getQuantity() method on the object being mapped. If the result is not null, the mapper executes the getAmount() method on the object obtained by the getQuantity() method. If getAmount() returns a value, the mapper assigns that value to the output.

- The character "=" separates **Source_Name** from **Target_Name**. It must always be provided.

- **Target_Name** is the name to give to the output Att object that is created by the mapper. This section of the entry is required.

- The character "|", is the separator between **Target_Name** and **Default**. **Default** is an optional parameter. If **Default** is not specified, the "|" character need not be specified. If **Default** is to be specified, the "|" is required.

- **Default** is the value to assign to the output if the value is not available from the source. This section of the entry is optional.

Blank entries or entries that begin with the hash character ("#") are ignored.
These are the available methods of the class:

3.1.1.2.1 **ESIMap(fileName : String)**

Loads the map file identified by the `fileName` argument. This is the only constructor for the class, thus insuring that every instance is properly initialized.

3.1.1.2.2 **getSourceAttributes() : Collection**

Returns a collection of Strings that represent the names of the source attributes.

3.1.1.2.3 **getTargetAttributes() : Collection**

Returns a collection of Strings that represent the names of the target attributes.

3.1.1.2.4 **iterator() : java.util.Iterator**

Returns an iterator for the Collection in the entries attribute.

3.1.1.3 **ESIMapper**

ESIMapper objects contain the logic to interpret ESIMap object contents. They also create output. ESIMapper objects provide three public methods. The methods are:

3.1.1.3.1 **map(object : Object, map : ESIMap) : Element**

This method creates a mapped Info*Engine element object from the Object argument.

3.1.1.3.2 **map(inputElement : Element, map : ESIMap) : Element**

This method creates a mapped Info*Engine element object from another Info*Engine element object.

3.1.1.3.3 **map(inputElement : Element, persistedObject : Persistable, map : ESIMap) : Element**

This method creates a mapped Info*Engine element object from another Info*Engine element object. It the source element object does not have the value specified in Source_Name of the map file entry, the mapper attempts to obtain the value from the Persistable object in the second method argument.
3.1.1.4 Mapper Class Collaboration

The following sequence diagram shows how an object uses the Windchill ESI Mapper:

Using ESI Mapper

1. The requesting object obtains an ESIMap object from the Windchill ESI map manager. The name of the map file is known to the object. When Windchill ESI objects invoke the mapper logic, they obtain the map file name from Windchill ESI properties.
2. The map manager examines its cache and returns the appropriate ESIMap object. If the ESIMap object does not exist in the cache, the map manager creates it and adds it to the cache.
3. The requesting object creates an ESIMapper object and calls the appropriate map method.
4. The ESIMapper object obtains an iterator from the Windchill ESI map. For each entry identified by the iterator:
5. The ESIMapper obtains the Source_Name
6. The ESIMapper obtains the value for that attribute from the source and adds it to the output.

Figure 8 Using Windchill ESI Mapper
### 3.1.2 Group and Map Information Classes

#### 3.1.2.1 ESIGroupInfo

Several Windchill ESI services RPC responses are Info*Engine virtual database objects. The VDB objects are composed of Info*Engine group objects. ESIGroupInfo objects define characteristics of the group. The attributes are:

- **logicalName : String**
  
  The logical name to be used by the clients of this object when obtaining the object from the Group Information Manager.

- **physicalName : String**
  
  The name to give the Info*Engine group this object describes.

- **releaseActivityNeeded : boolean**
  
  If the value is true pending ReleaseActivity objects are created when the group's contents are processed by the VDB builder. The existence of the ReleaseActivity objects prevents Windchill ESI services from attempting multiple concurrent attempts to publish the same object.

---

**Figure 9  Windchill ESI Group and Map Information**

- **ESI Group Information and Map Information**
- **ESIGroupInfoManager**
  
  - getInstance : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
  - ESIGroupInfoManager() : ESIGroupInfoManager
  
- **ESIGroupInfo**
  
  - logicalName : String
  
  - physicalName : String
  
  - releaseActivityNeeded : boolean

- **ESIMapInfo**
  
  - mapFileName : String
  
  - elementMetaName : String
  
  - keyAttributes : String[]
  
  - ESIMapInfo() : ESIMapInfo
  
  - ESIMapInfo() : ESIMapInfo

- **ESIMap**
  
  - entries : java.util.List
  
  - ESIMap() : ESIMap

- **ESIMapManager**
  
  - instance : ESIMapManager
  
  - ESIMapManager() : ESIMapManager

---
These attributes of the class have public getter methods. Additional public methods of the class:

3.1.2.1.4 ESIGroupInfo(logicalName : String, physicalName : String, mapInfo :
ESIMapInfo, isReleaseActivityNeeded : boolean)

It is the only constructor of the class; insuring that the object is properly initialized. This method is package-protected, forcing objects outside of this package, to use the ESIGroupInfoManager to obtain these objects.

3.1.2.1.5 getMapSourceAttributes() : Collection

Returns a collection of Strings that represent the names of the source attributes.

3.1.2.2 ESIGroupInfoManager

The ESIGroupInfoManager is a factory which creates ESIGroupInfo objects and makes them available to other objects. Implements the Singleton pattern to ensure that only one instance exists in a VM.

3.1.2.2.1 ESIGroupInfoManager()

The default constructor is declared as protected. This hides the default constructor to force access via the class' instance method.

3.1.2.2.2 getGroupInfo(groupName : String) : ESIGroupInfo

Returns the ESIGroupInfo for the supplied group name.

3.1.2.2.3 getGroupInfos() : Collection

Returns an ordered collection of all ESIGroupInfo objects.

3.1.2.2.4 <<Static>> instance() : ESIGroupInfoManager

Gets the value of the _instance attribute. If the attribute is null, create an instance and store it in the attribute before returning its value.

3.1.2.3 ESIMapInfo

Each ESIGroupInfo object contains an ESIMapInfo object. The ESIMapInfo object manages the relationship between the Info*Engine Group and the ESIMap that is used to format the contents of the Group. The attributes are:

3.1.2.3.1 mapFileName : String

The location of the ESIMap source file.

3.1.2.3.2 elementMetaName : String

When mapping the object, the output Info*Engine Element metadata element "com.infoengine.xml.esi" is to be assigned the value of this attribute. This causes the I*E SOAP servlet to render the output XML element name equal to the value of this attribute.

3.1.2.3.3 keyAttributes : String[]

The names of the key attributes used to filter out duplicates. If null, the elements are not filtered.
These attributes of the class have public getter methods. Addition public methods of the class:

3.1.2.3.4 \texttt{ESIMapInfo(mapFileName: String, elementMetaName: String, keyAttributes: String[])}

Only constructor for the object, ensures that the object is initialized properly.

3.1.2.3.5 \texttt{getMap(): ESI\texttt{Map}}

Returns the ESI\texttt{Map} identified by the value of the mapFileName attribute.

3.1.3 Customization Points

The Windchill ESI Mapper classes are fundamental building blocks for Windchill ESI output generation. The Windchill ESI Mapper classes are not intended to be customized. Map files may be customized.

3.2 VDB Builder

Windchill ESI services are invoked using SOAP-formatted RPC requests using JMS protocol. The output from such a request is a SOAP-formatted RPC response. Windchill ESI takes advantage of Info*Engine’s SOAP JMS processor. Info*Engine’s SOAP JMS processor creates XML from an Info*Engine virtual database (VDB). Figure 10 shows the objects of interest to Windchill ESI services.

A VDB contains zero or more group objects.

The Info*Engine SOAP JMS processor converts a VDB to XML using the following rules:

- The VDB is used to create an XML document fragment. The root element of the fragment is always \texttt{<COLLECTION>}.  
- Each Group object generates an XML element that is an immediate child of the root element. The name given to the XML element is the value of the className attribute of the group object.
• The group objects appear in the XML document in the order that they were added to the VDB.

• Each Info*Engine element object creates an XML element that is an immediate child of the XML element for the group object that contains the Info*Engine element object.

• Info*Engine objects allow you to set metadata information. If the element object metadata name, "com.infoengine.xml.esi" has a value, that is the name given to the XML element that represents the Info*Engine element object. If the metadata value was not set, the XML element is always <INSTANCE>.

• Each Att object can have zero or more values. Each value causes an XML element to be added as an immediate child of the XML element that describes the Info*Engine element object. The XML element for the att value has the same name as the Att object’s name attribute.

The Windchill ESI VDB Builder is a set of classes that collaborate to enable ESI to build complex VDB structures that would be otherwise unwieldy to build.

These are some things you might decide to customize:

• Apply filtering logic to the objects that are passed as arguments to the various addElement() methods of the VdbBuilder implementation

• Provide additional processing to modify the values that are passed to the various addElement() methods of the VdbBuilder implementation

Such customizations could be made by creating a subclass of VdbBuilderImpl and overriding the desired methods. The subclass need not be modeled using the Windchill Information Modeler.
3.2.1 Classes

Windchill ESI VDB Builder support is a set of classes that build Info*Engine VDB with a defined structure: These are the classes:

**VDB Builder**

![Class Diagram]

---

3.2.1.1 VdbBuilder

This interface declares the methods that all VDB builders must implement. These are the methods of the interface:

![VDB Builder Classes]

**Figure 11** VDB Builder Classes
3.2.1.1  initialize(transactionNumber : String)
   Makes sure the object is ready for requests. It is assumed that the implementation
   class has a transaction number and methods that use that transaction number.

3.2.1.2  getTransactionNumber() : String
   Returns the value of the transaction number that was passed to the initialize()
   method.

3.2.1.3  addElement(groupName : String, element : Element, persistable : Persistable,
            targets : Collection)
   Adds an element to a collection of elements. The collection is logically identified
   by the groupName argument.

3.2.1.4  addElement(groupName : String, element :
            com.infoengine.object.factory.Element, persistable : Persistable, dependents :
            Persistable[], targets : Collection)
   Adds an element to a collection of elements. The collection is logically
   identified by the groupName argument.

3.2.1.5  asVDB() : IeCollection
   Returns the results of the calculations that were performed on the elements passed
   into addElement() method calls.

3.2.1.6  isEmpty() : Boolean
   Returns true if all contained groups are empty.

3.2.1.2  VdbBuilderFactory
   Creates an instance of the correct implementation of the VdbBuilder interface.
   The class is not intended to be extended. This is the public interface of the class:

3.2.1.2.1 newVdbBuilder(transactionNumber : String) : VdbBuilder
   Creates a new object from the appropriate VdbBuilder implementation class.
   Obtains the name of the implementation class from Windchill ESI properties.

3.2.1.3  VdbBuilderImpl
   This class is the default Windchill ESI VdbBuilder implementation. It is
   extendable. These are the available methods of the class:

3.2.1.3.1 initialize(transactionNumber : String)
   Makes sure the object is ready for requests. Stores the argument in the
   transactionNumber attribute.

3.2.1.3.2 getGroup(groupName : String) : VdbGroup
   Returns the VdbGroup identified by the groupName attribute. Obtains the result
   from the associated VdbContainer. This is a protected method.
3.2.1.3.3 addElement(groupName : String, element : Element, persistable : Persistable, targets : Collection)

Creates a VdbElement containing an I*E element and a persistable. Adds the VdbElement to a VdbGroup. The VdbGroup is logically identified by the groupName argument.

3.2.1.3.4 addElement(groupName : String, element : com.infoengine.object.factory.Element, persistable : Persistable, dependents : Persistable[], targets : Collection)

Creates a VdbElement containing an I*E element and a persistable. Adds the VdbElement to a VdbGroup. The VdbGroup is logically identified by the groupName argument.

3.2.1.3.5 asVDB() : IeCollection

Creates the VDB from the contents of the VdbGroup objects that were built. If transactionID is not null, identifies the objects to be marked as pending and asks the transaction utility to mark them. Returns the result of VdbContain.asVDB().

The class has the following accessible attributes:

3.2.1.3.6 transactionNumber : String

The attribute is modeled. The class provides a public getter method, but does not allow the attribute to be set, except via the initialize() method.
3.2.1.4 VdbContainer

Manages a collection of VdbGroup objects and renders them as an Info*Engine IeCollection. This class is not intended to be extended by customizers. These are the available methods of the class:

3.2.1.4.1 addGroup(name : String, group : VdbGroup)

Adds a VdbGroup to the groups collection. Uses the logical name of the group that is passed in the name argument.

 Throws a MissingParameterException if either argument is null.

3.2.1.4.2 removeGroup(name : String)

Removes a VdbGroup from the groups collection if one exists in the collection with a logical name that is equal to the argument.
3.2.1.4.3 getGroup(name : String) : VdbGroup
   Returns a VdbGroup from the groups collection. Returns null if the group with the supplied name is not found in the collection.

3.2.1.4.4 getGroups() : Collection
   Returns the contents of the groups collection.

3.2.1.4.5 asVDB() : IeCollection
   Returns the contents of the groups collection as an Info*Engine IeCollection.

3.2.1.5 VdbGroup
   Manages a collection of VdbElements and renders the contents as an Info*Engine group. This class is not intended to be extended. These are the available methods of the class:

3.2.1.5.1 VdbGroup(name : String)
   This is the only constructor for the class. It ensures that the object is ready to process method calls by forcing the name attribute to be set.

3.2.1.5.2 getElement(key : Object) : VdbElement
   Returns the element whose key is equal to the argument. If no matching key, returns null.

3.2.1.5.3 getElements() : Collection
   Returns the values of the elements collection.

3.2.1.5.4 addElement(key : Object, element : VdbElement)
   Adds a VdbElement to the collection at the location identified by key. If there is already an element at that location, it is replaced.

3.2.1.5.5 removeElement(key : Object)
   Removes an element from the elements collection if there is one with a key equal to the argument.

3.2.1.5.6 asGroup() : Group
   Renders the VdbGroup as an Info*engine group.

   The class has one attribute that is accessible.

3.2.1.5.7 name : String
   The generated getter is public. The setter is private, since the class constructor method sets the attribute.

3.2.1.6 VdbElement
   Manages the contents of an Info*Engine element.

3.2.1.6.1 VdbElement(element : Element, persistable : Persistable)
   Only constructor for the class. Ensures that the object always contains an Info*Engine element and an optional persistable object.
3.2.1.6.2 **addTarget(target : ESITarget)**

Adds the argument to the distribution targets collection.

3.2.1.6.3 **addTargets(targets : Collection)**

Given a collection of distribution target objects, adds each entry to the targets collection.

3.2.1.6.4 **asElement() : com.infoengine.object.factory.Element**

Renders the element attribute as an Info*Engine element.

The attributes of the objects of the class have been generated with public getter methods. Unless otherwise noted, the attributes are initialized by the constructor and do not provide setter methods. The class has the following attributes:

3.2.1.6.5 **element : com.infoengine.object.factory.Element**

This is the Info*Engine element object that this object manages.

3.2.1.6.6 **persistable : Persistable**

The Persistable object associated with this VdbElement.

3.2.1.6.7 **dependents : Persistable[]**

A set of objects that the Info*Engine element depends upon. This attribute has a setter method.

3.2.1.6.8 **targets : Collection**

The distribution targets associated with this element.
ESI Response Renderers

Windchill ESI services invokes the VDB builder using classes called renderers. The renderers are described in the following sections of this document:

- **Publishing ECNs**
- **Publishing BOMs**
- **Publishing Parts**
- **Publishing Documents**

Figure 13  Windchill ESI Renderer Support
The following UML sequence diagram shows the message flow involved in use of a Windchill ESI services renderer:

- the client creates a VdbBuilder object for rendering.
- the client asks the renderer to render itself using the supplied VdbBuilder.
- if the Renderer is derived from ResponseRendererImpl, the default implementation is a template method that calls validate() and then buildGroups().
- within buildGroups(), the renderer will obtain the data to render, and will add each object element or object to VdbBuilder.
- the client obtains the generated VDB by calling asVDB() on the builder.

**Figure 14  Using a Windchill ESI Renderer**

Windchill ESI Services provides an abstract class and an interface that defines common capabilities of all Windchill ESI services renderers. (See Figure 14, above.) These are the methods provided:

### 3.2.1.7 ResponseRenderer

This interface defines the methods available on all Windchill ESI renderers. The interface is intended to be extended. These are the methods:

#### 3.2.1.7.1 \texttt{render(builder : VdbBuilder)}

Given an instance of VdbBuilder, creates the appropriate output.

### 3.2.1.8 ESIRendererException

Subclass of ESIException (and therefore of WTException) which allows clients to determine that the exception was thrown by an ESIRenderer.
3.2.1.9 ResponseRendererImpl

This is an abstract base class containing generally useful behavior for all Windchill ESI XML response renderers. The class is abstract and, therefore, must be extended. These are the available methods:

3.2.1.9.1 validate()

This is an abstract method that forces all subclasses to implement this method. The concrete method is expected to examine the attributes of the object and throw an exception if they are not set properly.

3.2.1.9.2 buildGroups()

This is an abstract method that forces subclasses to implement buildGroups. The concrete method is expected to navigate the object of interest and to create Info*Engine groups of elements and atts. When completed with the construction of the groups, the concrete class is expected to add the constructed groups into the IeCollection that this object created. Windchill ESI renderers use the services of a VdbBuilder to create and render the output.

3.2.1.9.3 logException(e : Exception)

Provides consistent error logging for subclasses.

3.2.1.9.4 getAttsFromMapFile(grpName : String) : String

Returns a list of attributes from the mapfile. The strings are "~" delimited.

3.2.1.9.5 render(builder : VdbBuilder)

This is a Template Method for Windchill ESI services renderers. It saves the argument and then calls validate() and buildGroups().

3.2.1.9.6 buildUfidString(objects : Collection, completeUFIDString : String, objectType : Class) : String

This method builds the UFID string of object ids for the I*E task. The UFIDs will be returned delimited by "~" character.

3.2.1.9.7 invokeQueryTask(taskName : String, className : String, ufids : String, auth : String, wcAdapter : String) : IeCollection

This method calls the Info*Engine task specified by the task name. The Info*Engine collection from the task is returned.

3.2.1.9.8 getEffectivityHandler() : ESIEffectivityHandler

Getter for the effectivityHandler attribute. Does lazy initialization. Throws an ESIRendererException if the lazy initialization fails.

3.2.1.9.9 setEffectivityHandler(handler : ESIEffectivityHandler)

Sets the effectivityHandler attribute. If the argument is null or the attribute is not null, the method does nothing.
3.2.1.9.10 mapElement(groupName : String, element : Element) : Element
Calls the Windchill ESI mapper to create an Info*Engine element using a
different Info*Engine element as input. Obtains the map information using the
logical group name in the groupName argument.

3.2.1.9.11 mapObject(groupName : String, object : Object) : Element
Calls the Windchill ESI mapper to create an Info*Engine element using an object
as input. Obtains the map information using the logical group name in the
groupName argument.

3.2.1.9.12 adjustObjectID(element : Element, objectID : String)
If the Info*Engine element argument has an "objectID" attribute, this method
changes the attribute’s value to the value of the objectID argument.

3.2.1.9.13 adjustTargets(element : Element, targets : Collection)
For each entry in the targets argument, creates a new Info*Engine Att object and
add it to the element argument.

3.2.1.9.14 adjustEffectivities(element : Element, eff : Eff)
Adds effectivity information to an element based on the effectivity of the WTPart.
This does the following:

1. Reads Windchill ESI properties to find the name for the effectivity end
date att of the argument element and the name of the effectivity start date
Att of the argument Element. If either of those atts are missing, or contain
a value they are ignored.
2. Gets an ESIEffectivityHandler from the ESIEffectivityHandlerFactory.
3. Gets a DateEff from the effectivity handler. If the result is not null:
   • Obtains the appropriate format from Windchill ESI properties
   • Formats and adds the start date to the appropriate Att
   • If the end date is present in the DateEff, formats it and adds it to
     the appropriate Att

3.2.2 Customization Points

3.2.2.1 Extendable classes
Some extendable Windchill ESI classes described in section VDB Builder of
this document must be defined in Windchill ESI properties if they are
extended. The following classes are designed to permit extensions by
customizers:

• ESIEffectivityHandlerImpl

  Property name:
  com.ptc.windchill.esi.rnd.class.ESIEffectivityHandler

  Default value:
  com.ptc.windchill.esi.rnd.ESIEffectivityHandlerImpl
• VdbBuilder
  Property name: com.ptc.windchill.esi.rnd.class.VdbBuilder
  Default value: com.ptc.windchill.esi.rnd.VdbBuilderImpl

3.2.2.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI renderers and the Windchill ESI VDB builder. The options are:

• Define the format of the data that is generated when effectivity dates are created. The value is used by a java.text.SimpleDateFormat and uses the syntax defined by that class.
  Property name: com.ptc.windchill.esi.rnd.effectivityFormat
  Default value: yyyy/MM/dd HH:mm:ss

• Define the XML element name that contains effectivity start date.
  Property name: com.ptc.windchill.esi.rnd.StartEffectivity
  Default value: StartEffectivity

• Define the XML element name that contains effectivity end date.
  Property name: com.ptc.windchill.esi.rnd.EndEffectivity
  Default value: EndEffectivity

• Declare the name of the class that defines the effectivity that is supported by Windchill ESI renderers.
  Property name: com.ptc.windchill.esi.rnd.class.Effectivity
  Default value: wt.effectivity.WTDatedEffectivity

3.3 Distribution Targets

3.3.1 Distribution Target Classes

3.3.1.1 ESITarget

Distribution targets represent a unique destination. Each destination is persisted in the database. ESITarget objects represent the persisted destinations. ESITarget is a subclass of wt.fc.Item and inherits the Windchill persistence and access control capabilities of the Item class (see Figure 15, below).

ESITarget objects have three attributes. The name attribute and the description attribute are provided for the benefit of the Windchill ESI organization assignment user interface. The attributes can contain any value up to 2048 characters in length. The name attribute is required. The description attribute may be null. The system attribute is the identifier used by the EAI software components to determine where to route Windchill ESI messages. The EAI software components define a structure for
the contents of the system attribute. Windchill ESI services requires the system attribute to be specified and to be 2048 characters or less. Additionally, each ESITarget must have a unique system attribute value in the database.
Windchill ESI does not provide a user interface for defining ESITarget objects. Windchill ESI services does provide methods to:

- Define ESITarget objects
- Delete ESITarget objects
- Associate ESITarget objects to WTOrganization objects
- Remove the association between ESITarget objects and WTOrganization objects

These are the available methods of the class:

3.3.1.1 `getObjectID() : String`
Returns the UFID of this ESITarget object.

3.3.1.2 `equals(obj : Object) : boolean`
Returns true if the argument is an instance of ESITarget and if the argument's system attribute equals this object's system attribute.

3.3.1.2 ESITargetOrganizationLink
An ESITarget object may be associated to zero or more WTOrganization objects. WTOrganization objects may be associated to zero or more ESITarget. ESITargetOrganizationLink objects represent those relationships. ESITargetOrganizationLink is a subclass of wc.fc.ObjectToObjectLink and inherits the persistent relationship capabilities of that class (see Figure 15).

ESITarget Relationships to Organizations

![Figure 15  ESITarget objects and Windchill Organizations](image)

**Note:** The ESITargetOrganizationLink object is provided for the benefit of customizers. Although the class and corresponding database table are defined, the current release of Windchill ESI services does not use the features of the class.
3.3.1.3 ESITargetAssignmentLink
Distribution target assignment is accomplished by creating a persisted relationship between a Windchill persistable object and an ESITarget. ESITargetAssignmentLink objects represent those relationships. ESITargetAssignmentLink is a subclass of wc.fc.ObjectToObjectLink and inherits the persistent relationship capabilities of that class (see Figure 16).

3.3.1.4 ESITargetAssociation
Windchill ESI services methods often need to maintain the relationship between a Windchill Persistable object and an ESITarget temporarily. The relationship may be persisted in the database, but it need not be. The ESITargetAssociation interface provides the relationship management capability for Windchill ESI services methods. The methods do not need to know whether or not the relationship is persisted. Several of the publicly available methods of Windchill ESI services accept arguments which are collections of ESITargetAssociation objects. The methods often return a collection of ESITargetAssociation objects.

When relationships are known to be persisted, ESITargetAssignmentLink objects are created or obtained from the Windchill Persistent Object Manager (see Figure 16).
3.3.1.5 ESIQueryTargetAssociation

When relationships are not known to be persistent, ESIQueryTargetAssociation objects are created by the caller of Windchill ESI services (see Figure 16). Several methods of the Windchill ESI Services API require arguments that are instances of ESITargetAssociation. The ESIQueryTargetAssociation class is provided as a convenience to objects that need to provide such arguments. The objects using ESIQueryTargetAssociation are not required to create any objects in the database.

ESITarget Association Diagram

```
ESITargetAssociation
    + getTarget()
    + getPersistableObject()

<<Abstract>>
ObjectToObjectLink
(from fc)

ESIQueryTargetAssociation
    target : ESITarget
    persistableObject : Persistable

Persistable
(from fc)

ESITarget
+ target
0..n

ESITargetAssignmentLink
+ initialize()

WTException
(from wt.util)
```

Figure 16  Distribution Target Assignment Objects

3.3.2 Establishing Distribution Target Assignment Classes

Distribution target assignments are established by creating an ESITargetAssignmentLink object and persisting that object in the database. Windchill ESI services class, StandardESIService, provides a public method, assignTarget(), that creates and saves the appropriate object.
3.3.2.1 ESITargetUtility

The distribution target package is intended to be self-contained and "private", except for the classes, ESITarget, ESITargetAssociation, ESIQueryTargetAssociation, and ESITargetUtility. ESITargetUtility is the public interface to the package. The services of the package should be invoked via the public methods of the ESITargetUtility class. The methods of the class are not thread-safe. Each thread is responsible for creating a new instance of the distribution target utility. The following methods are public services to the ESITargetUtility class:

3.3.2.1.1 getTargetAssociations(object : Persistable) : Collection

Returns a Collection of ESITargetAssociation objects that are associated with the argument. This method is used by ESISimplePartTargetFinder objects.
3.3.2.1.2  
\texttt{getTargetAssociations(object : Persistable, base : Persistable) : Collection}

Returns a Collection of ESITargetAssociation objects that are associated with the argument. The method uses an ESITargetFinder to obtain the result.

3.3.2.1.3  
\texttt{getTargetAssnsFromObjects(objects : Collection) : Collection}

Returns a Collection of ESITargetAssociation objects that are associated with the objects in the argument Collection. This method is used by ESISimplePartTargetFinder objects.

3.3.2.1.4  
\texttt{getTargetAssnsFromObjects(objects : Collection, base : Persistable) : Collection}

Returns a Collection of ESITargetAssociation objects that are associated with the objects in the argument collection. The method uses an ESITargetFinder to obtain the result.

3.3.2.1.5  
\texttt{getTargetAssnsFromAssns(associations : Collection) : Collection}

Given a collection of ESITargetAssociation objects, returns a collection of ESITargetAssociation objects that exist in the database for the persistable objects in the argument collection. This method is used by ESISimplePartTargetFinder objects.

3.3.2.1.6  
\texttt{getTargetAssnsFromAssns(associations : Collection, base : Persistable) : Collection}

Given a collection of ESITargetAssociation objects, returns a Collection of ESITargetAssociation objects that exist in the database for the persistable objects in the argument collection. The method uses an ESITargetFinder to obtain the result.

3.3.2.1.7  
\texttt{getTarget(id : String) : ESITarget}

Returns the ESITarget object that has a system attribute equal to the argument. If no ESITarget has that system attribute value, a ESINoTargetAssignmentException is thrown.

3.3.2.1.8  
\texttt{connectTargetToOrganization(tgt : ESITarget, org : WTOrganization)}

Creates an ESITargetAssociationLink between a WTOrganization and an ESITarget in the arguments.

3.3.2.1.9  
\texttt{disconnectTargetFromOrganization(tgt : ESITarget, org : WTOrganization)}

Deletes all ESITargetOrganizationLink objects that connect the ESITarget and the WTOrganization arguments.

3.3.2.1.10  
\texttt{getOrganizationLinksForTargets(targets : Collection) : Collection}

Given a collection of ESITarget objects, return a collection of ESITargetOrganizationLink objects that have one of the ESITarget objects as a roleA object.
3.3.2.11  getOrganizationLinksForOrganizations(organizations : Collection) : Collection
Given a collection of WTOrganization objects, return a collection of
ESITargetOrganizationLink objects that have one of the WTOrganization objects
as a roleB object.

3.3.2.12  assignTargets(targetAssociations : Collection)
Given a collection of ESITargetAssociation objects, create an
ESITargetAssignmentLink object for each entry in the collection that is not
already persisted in the database. Save the results in the database. Uses a
database transaction to ensure that all are saved or none are saved.

3.3.2.13  assignTargets(object : Persistable, targets : Group)
Given a Persistable object and an Info*Engine group of ESITarget objects, create
an ESITargetAssignmentLink for each unassigned ESITarget object in the group.
Save the results. Use a database transaction to insure that all are saved or none
are saved. Throw an exception if one or more ESITarget object in the group is
already assigned.

3.3.2.14  removeTargetAssociations(associations : Collection)
Deletes the ESITargetAssignments that are related to the persistable objects in the
argument Collection. Uses a database transaction to ensure that all are deleted or
none are deleted.

3.3.2.15  deleteTargetAssociation(link : ESITargetAssignmentLink)
Deletes the argument from the database.

3.3.2.16  deleteESITarget(tgt : ESITarget)
Deletes the argument from the database.

3.3.2.17  newESITarget(name : String, description : String, system : String) : ESITarget
Creates and saves ESITarget object. If required parameters are not provided or if
something goes wrong in the database, a WTException is thrown.

Caution: Do not use the Windchill Adapter Create-Objects
webject to create ESITarget objects. The webject does not
perform proper validation of the ESITarget object. Use the
newESITarget() method of the ESITransaction Utility to
create such objects.

3.3.2.18  newESITargetAssignmentLink(persistable : Persistable, target : ESITarget) : ESITargetAssignmentLink
Creates and saves an instance of ESITargetAssignmentLink. If Windchill ESI
properties specify a subclass of ESITargetAssignmentLink, an instance of that
class is returned. This method attempts to invoke the
newESITargetAssignmentLink(Persistable, ESITarget) method on the class
identified in Windchill ESI properties.
3.3.2.19  \textit{newESITargetOrganizationLink}(org : WTOrganization, target : ESITarget) : ESITargetOrganizationLink

Creates and saves an instance of ESITargetOrganizationLink. If Windchill ESI properties specify a subclass of ESITargetOrganizationLink, an instance of that class is to be returned. This method attempts to invoke the \textit{newESITargetOrganizationLink}(ESITarget, WTOrganization) method on the class identified in Windchill ESI properties.

3.3.2.20  \textit{validateDelete}(tgt : ESITarget)

Throws an exception if the argument is not eligible to be deleted.

- Examines Windchill ESI properties to determine whether or not to check ESITargetAssignmentLink objects. If the check is to be made, the method throws a WTException if the argument is associated to one or more ESITargetAssignmentLink (or a subclass designated by Windchill ESI properties) objects.

- Examines Windchill ESI properties to determine whether or not to check ESITargetOrganizationLink objects. If the check is to be made, the method throws a WTException if the argument is associated to one or more ESITargetOrganizationLink (or a subclass designated by Windchill ESI properties) objects.

3.3.3 Distribution Target Assignment Determination Classes

Distribution target assignments can be determined either explicitly or implicitly. Explicit assignments are the persisted relationships that are maintained by ESITargetAssignmentLink objects. Implicit assignments are relationships that are inferred from some other object’s persisted relationships.

Windchill ESI provides support for multiple assignment determination strategies by using ESITargetFinder objects. Windchill ESI only supports one strategy at a time in any Windchill instance. The strategy is implemented by writing a java class that implements the com.ptc.windchill.esi.tgt.ESITargetFinder interface. The ESITargetUtility obtains the appropriate ESITargetFinder from the ESITargetFinderFactory. The ESITargetFinderFactory obtains the name of the implementation class from Windchill ESI properties.
3.3.3.1 ESITargetFinderFactory
Provides one static synchronized method, \texttt{newESITargetFinder(object : Persistable, utility : ESITargetUtility) : ESITargetFinder}. The method obtains the name of the implementation class from Windchill ESI properties. If the implementation class is a subclass of ESITargetFinderImpl, the method invokes setBase() and setUtility() on the finder before returning the finder.

3.3.3.2 ESITargetFinderImpl
This is an abstract base class for the out-of-the-box distribution target finders. It has an attribute, named "base", of class \texttt{wt.fc.Persistable}. The class also has an association with an ESITargetUtility.

3.3.3.3 ESISimplePartTargetFinder
This class implements explicit assignment. It delegates all of its work to the ESITargetUtility. These are the available methods of the class:

3.3.3.3.1 \texttt{getTargetAssnsFromObjects(objects : Collection) : Collection}
Returns the result of the method with the same signature on the ESITargetUtility.

3.3.3.3.2 \texttt{getTargetAssnsFromAssns(assns : Collection) : Collection}
Returns the result of the method with the same signature on the ESITargetUtility.

3.3.3.3.3 \texttt{getTargetAssociations(object : Persistable) : Collection}
Returns the result of the method with the same signature on the ESITargetUtility.
3.3.3.4 ESIRootInheritTargetFinder

This class implements one form of implicit assignment. If the objects have no explicit distribution target assignments, they are assumed to be assigned to the same distribution targets as the base object.

3.3.3.4.1 getTargetAssnsFromObjects(objects : Collection) : Collection

Returns a collection of ESITargetAssociations. For each Persistable in the argument, obtain the ESITarget Assignments from the ESITargetUtility. If there are assignments, adds them to the result. If there are no assignments:

- Obtains the base object’s distribution target assignments
- For each base object assignment, creates a new ESIQueryTargetAssociation object. Uses the base assignment distribution target and the object from the argument collection.
- Adds the ESIQueryTargetAssociation to the result.

3.3.3.4.2 getTargetAssnsFromAssns(assns : Collection) : Collection

Obtains the persistable objects from the argument collection. Invokes getTargetAssnsFromObjects() using those persistable objects. Returns the result of getTargetAssnsFromObjects().

3.3.3.4.3 getTargetAssociations(object : Persistable) : Collection

Obtains distribution target associations of the argument from the ESITargetUtility. If there are assignments, returns the result. If there are no assignments:

- Obtains the base objects distribution target assignments
- For each base object assignment, creates a new ESIQueryTargetAssociation object. Uses the base assignment distribution target and the object from the argument.
- Adds the ESIQueryTargetAssociation to the result.

3.3.4 GUI Support

Distribution target assignment user interface is described in the Windchill Enterprise Systems Integration User's Guide. Windchill ESI distribution target UI is developed using PTC’s Dynamic Client Architecture (DCA). DCA requests data and services by invoking Info*Engine tasks. The Windchill ESI UI invokes standard DCA tasks and Windchill ESI tasks. Windchill ESI tasks invoke the services of the Windchill Adapter via the Apply-Service webject. (See the Windchill Adapter Guide for details.) The Apply-Service webject calls the appropriate method on the StandardESIService class. (See the Windchill ESI Windchill section of this document for details.) StandardESIService delegates all work to an instance of ESITargetUtility. (See the ESITransactionUtility section of this document for details).
3.3.5 Info*Engine Tasks

3.3.5.1 AssignTarget
Given the UFID of a wt.fc.Persistable object and the UFID of an ESITarget, the task calls the assignTarget method of StandardESIService via the Windchill Adapter Apply-Service Webject.

3.3.5.2 RemoveTargetAssignment
Given the UFID of an ESITarget ESITargetAssignmentLink Link, the task calls the removeTargetAssignment() method of StandardESIService via the Windchill Adapter Apply-Service Webject. The StandardESIService deletes the link object from the database.
3.3.6 Customization Points

3.3.6.1 Extendable classes

Extendable Windchill ESI classes, described in the Distribution Targets section of this document, must be defined in Windchill ESI properties if they are extended. The following classes are designed to permit custom extensions:

- ESITarget
  
  Property name: com.ptc.windchill.esi.tgt.class.ESITarget
  Default value: com.ptc.windchill.esi.tgt.ESITarget

- ESITargetAssignmentLink
  
  Property name: com.ptc.windchill.esi.tgt.class.ESITargetAssignmentLink
  Default value: com.ptc.windchill.esi.tgt.ESITargetAssignmentLink

- ESITargetOrganizationLink
  
  Property name: com.ptc.windchill.esi.tgt.class.ESITargetOrganizationLink
  Default value: com.ptc.windchill.esi.tgt.ESITargetOrganizationLink

- ESITargetFinder
  
  Property name: com.ptc.windchill.esi.tgt.class.ESITargetFinder
  Default value: com.ptc.windchill.esi.tgt.class.ESISimplePartTargetFinder

3.3.6.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI Transaction Management. The options are:

- Allow deletion of ESITarget objects that have associated distribution target assignments.
  
  Property name: com.ptc.windchill.esi.tgt.allowDelete.ESITargetAssignmentLink
  Default value: false

- Allow deletion of ESITarget objects that have associated WTOrganization objects.
  
  Property name: com.ptc.windchill.esi.tgt.allowDelete.ESITargetOrganizationLink
  Default value: false
• Define the name of the map file to use when displaying ESITarget objects.
  
  Property name:  com.ptc.windchill.esi.tgt.mapFile
  Default value: $CODEBASE$/tgt/ESITarget.map

• Define the delimiter to use when finding a single UFID within a string of concatenated UFIDs. This property is used internally by Windchill ESI services; it does not affect the behavior of other Windchill components.
  
  Property name:  com.ptc.windchill.esi.uti.UfidSeparator
  Default value:|

• Define the ReleaseActivity releaseClass attribute value that identifies a BOM. This value is required because a single Windchill part can be published twice to the same distribution target. The first publishing defines a part. The second publishing defines a BOM. Some Windchill ESI APIs are required to distinguish between the two types of publishing activity. This property tells those APIs what to look for.
  
  Property name:  com.ptc.windchill.esi.txn.class.BOM
  Default value: com.ptc.windchill.esi.BOMHeader

3.4 Transaction Management

Publishing history is a relationship between a publishable object and a distribution target. These are the publishable object types in Windchill ESI:

• Change Orders
• Parts (WTPart objects and information from their associated WTPartMaster objects)
• Part Usage Links (BOM components)
• Part Substitute Links
• Part Alternate Links
• WTDoucument objects
• Document Reference Links
• Document Describes Links

The UML diagram in Figure 20 shows the relationship between the publishable object types and the Windchill ESI objects that track publishing history.
3.4.1 Creating Publishing History Objects

The Windchill ESI event emitter creates ESITransaction objects immediately prior to emitting an event. The initial status of the transaction is "PENDING".

Windchill ESI RPCs GetPart, GetBOM, and GetECN create ReleaseActivity objects for every object that is expected to be published. The status of the ReleaseActivity objects is set to pending.

Windchill ESI adds ReleaseActivity objects to a transaction whenever a PostResult (subtransaction=true) RPC request executes. The first ReleaseActivity that is added to the transaction causes its state to be set to "PROCESSING".

When a PostResult (subtransaction=false) RPC request executes, Windchill ESI sets the transaction status to "SUCCEEDED" or "FAILED". The RPC also causes all "PENDING" ReleaseActivity objects to be deleted.

The Windchill ESI Windchill Service processes the PostResult request. The service determines whether or not a subtransaction is being processed. The service then passes the request parameters to the appropriate method of ESITransactionUtility. ESITransactionUtility creates a ReleaseActivity or updates the ESITransaction.
The Windchill ESI Windchill Service’s RPC support methods are not available for execution on remote clients.

The level of logging detail is controlled by these two Windchill ESI configuration properties:

- `com.ptc.windchill.esi.result.logFailSubTran` If this property is "false", causes the PostResult to ignore failed subtransaction messages. The default for this property is "true".

- `com.ptc.windchill.esi.result.logSuccessSubTran` If this property is "false", all successful subtransaction messages are immediately flagged as deleted. This suppresses their display on the user interface, but records the object as having been released. The default value for this property is "true".

Figure 21, below, shows the status values of a successful transaction.
Figure 22 shows the status values of a failed transaction.

Windchill ESI Services
Failed Transaction

Figure 22  Failed Transaction Status Values over Time
3.4.2 Classes

Enumerated Types

- **ReleaseStatusType**
  - FAILED : ReleaseStatusType = toReleaseStatusType("failed")
  - SUCCEEDED : ReleaseStatusType = toReleaseStatusType("succeeded")
  - PENDING : ReleaseStatusType = toReleaseStatusType("pending")

- **ESITransactionStatusType**
  - PROCESSING : ESITransactionStatusType = toESITransactionStatusType("processing")
  - PENDING : ESITransactionStatusType = toESITransactionStatusType("pending")
  - FAILED : ESITransactionStatusType = toESITransactionStatusType("failed")
  - SUCCEEDED : ESITransactionStatusType = toESITransactionStatusType("succeeded")

- **ReleaseActivityAction**
  - ADD : ReleaseActivityAction = toReleaseActivityAction("add")
  - CHANGE : ReleaseActivityAction = toReleaseActivityAction("change")
  - DELETE : ReleaseActivityAction = toReleaseActivityAction("delete")

Figure 23  Enumerated Types in Windchill ESI Transaction

3.4.2.1 ESITransaction

Any Windchill ESI publication attempt generates a Windchill ESI transaction, which is represented by a persistable ESITransaction object. The transaction has an initial status of "PENDING." Other transaction status values are: "PROCESSING", "SUCCEEDED," or "FAILED." The values are localized using standard Windchill enumerated types. The ESITransactionStatusType class defines the range of legal values for ESITransaction status. See Figure 23, for the definition of Windchill ESI transaction enumerated types.

The transaction has a unique number and a flag that indicates whether the user tried to delete the transaction (that is, permanently hide it from the user interface). The system generates a unique number by finding the highest value in the database and adding one to it. In order to prevent duplicate numbers, the system serializes the creation of transactions.

ESITransaction is a "public" class. The class inherits from wt.fc.Item so that it can take advantage of standard Windchill access control functionality. Customizers may extend this class.
3.4.2.2 ESITransactionObjectLink

ESITransaction objects may be related to wt.fc.WTObject objects. The relationship is persistent. ESITransactionObjectLink objects manage the relationship.

ESITransactionObjectLink is a "public" class. The class may be extended by customizers.

3.4.2.3 ReleaseActivity

Each transaction includes zero or more release activities, which are represented by persistable ReleaseActivity objects as shown in Figure 24, above. ReleaseActivity objects are subclasses of Windchill ObjectToObjectLink, as shown in Figure 24.

ReleaseActivity objects may have one of three statuses, FAILED, SUCCEEDED, or PENDING. The status values are defined by the enumerated type class, ESITransactionStatusType.

ReleaseActivity object have an action attribute. The values are defined by enumerated type class, ReleaseActivityAction. See Figure 23, above.

ReleaseActivity objects may have one associated ReleaseActivityMessage. See Figure 20, above.
A release activity is associated with one releasable object, which is an instance of wt.fcPersistable, and is associated with one ESITarget object. The ReleaseActivity objects have the following additional attributes:

- **creator**, which is the user who created the ReleaseActivity. Often the value is the session principal that was associated with the function that created the ReleaseActivity. The attribute is a java.lang.String because the value may not be a WTPrincipal.

- **description**, which is useful text that was supplied by the function that created the ReleaseActivity.

- **releaseClass**, which is a Windchill ESI logical name, provided on all Windchill ESI RPC responses.

- **deleted**, which is a flag that determines whether the user wished to delete the entry (that is, permanently hide it from the user interface).

- **releaseStamp**, which is the timestamp provided by the function that created the ReleaseActivity. If the value was not provided, releaseStamp is set to the time when the ReleaseActivity object was created. When displaying transactions and their associated activities, the activities are sorted in order of ascending releaseStamp.

ReleaseActivity objects that have a status of "SUCCEEDED" are used for determining whether or not an object has been previously published. These may not be deleted from the database, since that information is required for determining the appropriate request to make (update or create) when releasing an object. Administrators with appropriate SQL privileges may be delete Windchill ESI transactions and failed publishing activities (with their associated messages). Windchill ESI does not provide a tool to do that.

ReleaseActivity is a "public" class. The class may be extended by customizers. Javadoc is provided.

### 3.4.2.4 ReleaseActivityMessage

ReleaseActivityMessage objects are persistable objects that carry information that may be helpful to the Windchill ESI user. Each ReleaseActivity may have one ReleaseActivityMessage associated to it.

ReleaseActivity is a "public" class. The class is not intended to be extended by customizers.

### 3.4.2.5 ESITransactionRelease

ESITransaction objects may be related ReleaseActivity objects. The relationship is persistent. ESITransactionRelease objects manage the relationship.

ESITransactionRelease is a "public" class. The class may be extended by customizers.
3.4.2.6 ESIRelatedTransaction

ESIRelatedTransaction is an object-to-object link between two Windchill ESI transactions. It is provided in anticipation of a requirement to couple multiple publication requests. The object is modeled in Windchill ESI, but there are no services to create or maintain ESIRelatedTransaction objects.

ESIRelatedTransaction is a "public" class. The class may be extended by customizers.
### 3.4.2.7 ESITransactionUtility

<table>
<thead>
<tr>
<th>ESITransactionUtility</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;final&gt;&gt; getTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; addTransactionMessage()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; removeTransactionMessage()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getTransactionActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getBOMActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getRelatedTransactions()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; updateTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; recordExport()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; deleteTargetAssociation()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; deleteTargetAssociation()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getMaxTransactionNumber()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; createTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; createTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; hideTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; createReleaseActivity()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; postResult()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; postEvent()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; latestExport()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; latestPartExport()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; latestEffExport()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getLatestUnreleasedDocuments()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; markPending()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; isTxnPending()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; isPartReleaseNeeded()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; removePending()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getTargetsWithHistory()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getTargetsWithHistory()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; isPartNewerThan()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getReleaseActivities()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getReleaseActivitiesMessage()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getTransactionLogMessage()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; validateDelete()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; validateDelete()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; validateDelete()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; getTransactions()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; abortTransaction()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; findTransactions()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; findTransactions()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; isTxPending()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; isTxPending()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; hasTxSucceed()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; hasTxFailed()</td>
</tr>
<tr>
<td>&lt;&lt;final&gt;&gt; hasReleased()</td>
</tr>
<tr>
<td>&lt;&lt;Static&gt;&gt; performStartupProcess()</td>
</tr>
</tbody>
</table>

#### Figure 25  Transaction Utility
The Windchill ESI transaction package is intended to be self-contained and "private", except for the classes, ESITransaction, ReleaseActivity, and ESITransactionUtility. ESITransactionUtility is a public interface to the package. The services of the package should be invoked via the public methods of the ESITransactionUtility class. The methods of the class are not thread-safe. Each thread is responsible for creating a new instance of the transaction utility. The following methods are public services to the ESITransactionUtility class:

3.4.2.7.1 \texttt{getTransaction(number : String) : ESITransaction} \\
Returns the ESITransaction that is has an idNumber equal to the argument.

3.4.2.7.2 \texttt{addTransactionMessage(txn : ESITransaction, text : String, txnRequired : boolean)} \\
Adds a message to an ESITransaction object. If the Windchill ESI transaction already had a message, the old one is replaced by the new message. This results in multiple database activities which should be wrapped in a database transaction. If the caller has started a database transaction, pass false in the txnRequired argument, otherwise pass true in the argument.

3.4.2.7.3 \texttt{removeTransactionMessage(txn : ESITransaction)} \\
Deletes from the database any ESITransactionMessage that is associated with the ESITransaction object in the argument.

3.4.2.7.4 \texttt{getTransactionActivities(id : String) : Collection} \\
Returns all of the ReleaseActivity objects that are associated with the ESITransaction whose idNumber is equal to the argument.

3.4.2.7.5 \texttt{getActivities(object : Persistable) : Collection} \\
Returns all of the ReleaseActivity objects that are associated with the argument. Only objects that have a status of Pending or Successful are returned.

3.4.2.7.6 \texttt{getActivities(ufid : String) : Collection} \\
Returns the ReleaseActivity objects that are associated with the ESITransaction object that is identified by the UFID in the argument.

3.4.2.7.7 \texttt{getAllActivities(object : Persistable) : Collection} \\
Returns a Collection of ReleaseActivities that are associated with a persisable object argument.

3.4.2.7.8 \texttt{getBOMActivities(parts : Collection) : Collection} \\
Given a collection of parts, return all ReleaseActivity objects that indicate one or more versions of the parts have been published as a BOM. The method:

- Finds the part master objects associated with the parts in the argument
- Finds all iterations of the master
- Finds all successful ReleaseActivity objects that are associated with the obtained part iterations
• Adds it to the result any ReleaseActivity object that has a releaseClass indicating that it is a BOM

3.4.2.7.9 getRelatedTransactions(txn: ESITransaction): Collection

Returns all of the ESITransaction objects that are related to the argument. This method is not implemented for the current release.

3.4.2.7.10 recordExport(obj: Persistable, org: ESITarget, releaseClass: String, action:String)

Creates a ReleaseActivity with a status of success for the object and the identified distribution target. This method is intended as a convenience for System Integrators who need to with synchronize the Windchill ESI release history and the state of the objects in the target systems.

3.4.2.7.11 deleteTargetAssociation(link: ESITargetAssignmentLink)

Removes the argument from the database. The method insures that the persistable object associated to the argument has not already been published to the distribution target associated to the argument before allowing the deletion to occur.

3.4.2.7.12 createTransaction(object: WToObject)

Creates a new ESITransaction that is associated to the supplied Windchill object. Persists the transaction and the association (creates and saves and ESITransactionObjectLink object).

The supplied Windchill object reference may be null, in which case this method does not create an ESITransactionObjectLink object.

3.4.2.7.13 createTransaction(object: WToObject, String userName)

Creates a new ESITransaction that is associated to the supplied Windchill object. Persists the transaction and the association (creates and saves and ESITransactionObjectLink object). The supplied user name will be used to set the creator attribute of the ESITransaction object.

The supplied Windchill object reference may be null, in which case this method does not create an ESITransactionObjectLink object.

3.4.2.7.14 hideTransaction(ufid: String)

Hide an ESITransaction from the user interface. The method sets the value of the deleted attribute to "true" and saves the object to the database.

3.4.2.7.15 createReleaseActivity(persistable: Persistable, description: String, transaction: ESITransaction, successful: boolean, target: ESITarget, message: String, timestamp: String, action: String, cls: String, user: String)

Creates and saves a ReleaseActivity object.
3.4.2.7.16 postResult(objectID : String, description : String, txnID : String, isSuccessful : boolean, isSubtran : boolean, targetSystem : String, message : String, timestamp : String, action : String, cls : String, user : String)
This method is the implementation for the Windchill ESI PostResult RPC. See the Windchill Enterprise Systems Integration Open API Guide for details.

3.4.2.7.17 postEvent(objectID : String, transaction : String, isSuccessful : boolean, message : String)
This method is the implementation for the Windchill ESI PostEvent RPC. See the Windchill Enterprise Systems Integration Open API Guide for details.

3.4.2.7.18 latestPartExport(parts : Collection) : Collection
Given a collection of WTPart objects, finds the latest released iteration in every ESITarget. Returns a collection of ESITargetAssociation objects.

3.4.2.7.19 latestEffExport(part : EffManagedVersion, target : ESITarget) : Eff
Returns the last published wt.effectivity.Eff object that is associated with the supplied part and target. The last publish attempt may have a release status of pending or successful.

3.4.2.7.20 getLatestUnreleasedDocuments (docs : Collection) : Collection
Given a collection of WTDocument objects and a collection of ESITarget objects, this method returns a collection of ESITargetAssociation objects for documents that have not been published to the provided distribution targets. The returned ESITargetAssociation objects are the highest version/iteration that is associated with an ESITarget.

The objects in the request class must all be members of the same class.

3.4.2.7.21 markPending(tgtAssns : Collection, txnID : String)
Creates a ReleaseActivity with status = PENDING for each of the object - distribution target pairs (ESITargetAssociation objects) in the tgtAssns argument. Associate the ReleaseActivity objects with the ESITransaction whose idNumber attribute is equal to the value of the txnID argument.

Encloses all database requests in a database transaction so that if anything fails the entire request is rolled back.

3.4.2.7.22 isTxnPending(object : Persistable) : boolean
Returns true if the argument is associated with an ESITransaction whose status is either PENDING or PROCESSING.

3.4.2.7.23 isPartReleaseNeeded(assns : Collection) : boolean
Given a collection of ESITargetAssociations, returns true if any of the persistable objects have not been published to the ESI target identified in the target association.
3.4.2.7.24 `removePending(txn : ESITransaction)`
Deletes all ReleaseActivity objects that are associated with the argument and whose state is PENDING.

3.4.2.7.25 `getTargetsWithHistory(objects : Collection) : Group`
Given a collection of wt.fcPersistable objects, gets a collection of the ESITargetAssignmentLink objects that are associated with the persistable objects in the argument. Creates ESITargetHistory objects and attaches any related ReleaseActivity objects. Passes the resulting ESITargetHistory object to the ESIMapper to create an output group of Info*Engine elements.

3.4.2.7.26 `getTargetsWithHistory(object : Persistable) : Collection`
Given a wt.fcPersistable object, gets a collection of the ESITargetAssignmentLink objects that are associated with the argument. Creates ESITargetHistory objects and attaches any related ReleaseActivity objects. Passes the resulting ESITargetHistory objects to the ESIMapper to create an output group of Info*Engine elements.

3.4.2.7.27 `isPartNewerThan(current : WTPart, other : WTPart) : boolean`
Determines whether one WTPart has been published more recently than another WTPart.

3.4.2.7.28 `getReleaseActivities(ufid : String) : Group`
Returns an Info*Engine Group of mapped ReleaseActivity objects given the UFID of an ESITransaction object.

3.4.2.7.29 `getReleaseActivitiesMessage(ufid : String) : Group`
Returns an Info*Engine Group of mapped ESITransactionMessage objects given the UFID of the ESITransaction object.

3.4.2.7.30 `getTransactionLogMessage(ufid : String) : Group`
Returns an Info*Engine Group of mapped ESITransactionMessage objects given the UFID of an ESITransaction object.

3.4.2.7.31 `validateDelete(tgt : ESITarget)`
Throws an exception if the argument is not eligible to be deleted. The method is called by the StandardESIService whenever an ESITarget object is about to be deleted. The method examines Windchill ESI properties to determine whether or not to check for the existence of ReleaseActivity objects. If the check is to be made, the method throws a WTException if the argument is associated to one or more ReleaseActivity objects. If no exception is thrown in the first check, the method calls ESITargetUtility to perform additional checks.

The method is designed to work with subclasses of ReleaseActivity.

3.4.2.7.32 `validateDelete(txn : ESITransaction)`
Throws a WTException if the argument is not eligible for deletion. The method is called by the StandardESIService whenever an ESITransaction object is about
to be deleted. The method throws a WTException if the argument has a status of
PENDING or PROCESSING.

3.4.2.7.33 validateDelete(part : WTPart)
Throws an exception if the argument is not eligible to be deleted. The method is
called by the StandardESIService whenever an ESITransaction object is about to
be deleted.

1. Examines Windchill ESI properties to determine whether or not to check
ReleaseActivity objects. If the check is to be made, the method throws a
WTException if the argument is associated to one or more ReleaseActivity
objects. The method is designed to work with subclasses of
ReleaseActivity.

2. Examines Windchill ESI properties to determine whether or not to check
ESITransactionObjectLink objects. If the check is to be made, the method
throws a WTException of the argument is associated to one or more
ESITransactionObjectLink objects. The method is designed to work with
subclasses of ESITransactionObjectLink.

3.4.2.7.34 getTransactions(ufids : Group) : Group
Returns an Info*Engine Group of ESITransaction objects that are associated with
the objects in the argument. The argument is an Info*Engine Group containing
Elements that have been created by (or resemble the output of) a Query-Objects
webject.

3.4.2.7.35 abortTransaction(obj : Persistable)
Remove an ESITransaction from the database

Note: This method can only be called by an ESI workflow if any system errors
(for example, JMS server is not available) occur during a publish attempt.

3.4.2.7.36 findTransactions(obj : Persistable) : QueryResult
Returns all ESITransaction objects that are associated with the given object.

3.4.2.7.37 findTransactions(obj : Persistable, status : String) : QueryResult
Returns all ESITransactions that are associated with the given object and that
have the given status.

3.4.2.7.38 isTxPending(obj : Persistable) : boolean
Returns true if an ESITransaction object is associated with the given object and
the ESITransaction has a status of "pending".

3.4.2.7.39 isTxProcessing(obj : Persistable) : boolean
Returns true if an ESITransaction object is associated with the given object and
the ESITransaction has a status of "processing".

3.4.2.7.40 hasTxSucceed(obj : Persistable) : boolean
Returns true if an ESITransaction object is associated with the given object and
the ESITransaction has a status of "succeeded".
3.4.2.7.41 hasTxFailed(obj : Persistable) : boolean
Returns true if an ESITransaction object is associated with the given object and
the ESITransaction has a status of "failed".

3.4.2.7.42 hasReleased(obj : Persistable, target : ESITarget) : boolean
Returns true if the object has been published successfully to the given distribution
target.

3.4.2.7.43 performStartupProcess()
This static method Performs the necessary startup logic for the Transaction
pUtility. It is called at method server startup by the StandardESIService.

Figure 26   Txn Package subclasses of ESITargetAssociation

3.4.2.8 ESITargetHistory
ESITargetHistory objects manage the relationship between distribution target
assignments and ReleaseActivity objects. ESITargetHistory objects are not
persistent. The objects are created to provide the information required by the
Windchill ESI Distribution List GUI.
ESITargetHistory is a "public" class. The class is not designed to be extended by customizers. Javadoc is provided. These are the public methods of the class:

3.4.2.8.1  getLinkObid() : String
Returns the UFID of the ESITargetAssignmentLink that is held in the targetAssociation attribute.

3.4.2.8.2  getStatus() : String
Returns history.getStatus().

3.4.2.8.3  getTarget() : ESITarget
Returns targetAssociation.getTarget().

3.4.2.8.4  getPersistableObject() : Persistable
Returns targetAssociation.getPersistableObject().

3.4.2.9  ESIPendingTxnRequest
ESIPendingTxnRequest objects the ESITargetAssociation by providing two attributes that contain information that is required by ESITransactionUtility. markPending(). The objects are not persistent. ESIPendingTxnRequest is a "public" class. The class is not designed to be extended by customizers. Javadoc is provided. These are the public methods of the class:

3.4.2.9.1  ESIPendingTxnRequest()
This is the default constructor.

3.4.2.9.2  ESIPendingTxnRequest(assn:ESITargetAssociation, cls:String, action:String)
Constructor that establishes the class in the correct state.

3.4.2.9.3  getTarget() : ESITarget
Returns targetAssociation.getTarget().

3.4.2.9.4  getPersistableObject() : Persistable
Returns targetAssociation.getPersistableObject()

The class has attributes which have public getter methods. The attributes do not have setter methods because the attributes values are presumed to be established by the class constructor. These are the attributes:

3.4.2.9.5  releaseClass : String
The releaseClass value to establish when creating a pending ReleaseActivity.

3.4.2.9.6  action : String
The action value to establish when creating a pending ReleaseActivity.

3.4.3  GUI Support
The Windchill ESI transaction management user interface (Windchill ESI Transaction Log) is described in the Windchill Enterprise Systems Integration User's
Guide. The Windchill ESI transaction user interface is developed using PTC’s Dynamic Client Architecture (DCA). The DCA requests data and services by invoking Info*Engine tasks. The Windchill ESI user interface invokes standard DCA tasks and Windchill ESI tasks.

The Windchill ESI tasks invoke the services of the Windchill Adapter via the Apply-Service webject. (See the Windchill Adapter Guide for details.) The Apply-Service webject calls the appropriate method on the StandardESIService class. (See the Windchill Adapter Guide for a detailed description of the Apply-Service webject.) The StandardESIService delegates all work to an instance of ESITransactionUtility. (See the ESITransactionUtility section of this document for details).

![Windchill ESI Transaction User Interface Message Flow](image)

### 3.4.4 GetHistory Support

Windchill ESI provides an API that provides the publishing status of Windchill ESI distribution targets that are associated with Windchill part objects. The Windchill ESI services getHistory() API allows customizers obtain similar publishing status information for Windchill persistable objects that are not Windchill parts. Several
classes in the java package, com.ptc.windchill.esi.rnd implement the API. Figure 28, below, shows the classes that support the API.

Object History Renderers

![Diagram of Object History Renderers]

**Figure 28** GetHistory API Support

### 3.4.4.1 ESIObjectHistoryRenderer

This interface defines the public methods of Windchill ESI object history renderers. These are the methods of the interface:

#### 3.4.4.1.1 `setData(ufid : String)`

Establishes the state of the object so that it is ready to process requests.

#### 3.4.4.1.2 `render(builder : ESIObjectHistoryBuilder)`

Generate output using the help of the argument.
3.4.4.2 ESIObjectHistoryRendererImpl

This class is the default implementation of the ESIObjectHistoryRenderer interface.

The class has two attributes that are accessible to subclasses:

3.4.4.2.1 ufid : String

This attribute contains a String which is presumed to be a valid UFID of a Windchill Persistable object. Although the attribute has public setters and getters for the convenience of customizers, the value of this attribute is normally established using the setData() method.

3.4.4.2.2 builder : ESIObjectHistoryBuilder

This attribute is defined as an association in the Windchill ESI model. The association is defined so that the Windchill code generation provides a protected getter method and a protected setter method. The attribute value is set when the render() method is invoked.

These are the available methods of the class:

3.4.4.2.3 setData(ufid : String)

Saves the argument in the ufid attribute.

3.4.4.2.4 render(builder : ESIObjectHistoryBuilder)

This is a template method for object history renderers. It saves the argument and then calls validate() and buildGroups(). If subclasses override this method, the overriding implementation should either call this method or save the argument in the builder attribute.

3.4.4.2.5 validate()

This protected method insure that the object is ready to process. If attributes, builder or ufid are null, the method throws an exception.

3.4.4.2.6 buildGroups()

The protected method processes the ReleaseActivity objects that are associated with the object that is identified by the ufid attribute. The method obtains a collection of ReleaseActivity objects from an ESITransactionUtility object. Passes each of the ReleaseActivity objects to the builder’s addElement() method.

3.4.4.3 ESIObjectHistoryBuilder

This interface defines the public methods of Windchill ESI object history renderers.

These are the methods of the interface:

3.4.4.3.1 setData(groupOutName : String)

Establishes the state of the object so that it is ready to process requests.

3.4.4.3.2 addElement(ra : ReleaseActivity)

Map the argument and add it to the output.
3.4.4.3.3 \textit{getGroup()} : \textit{Group}  
Return the result of the build process.

3.4.4.4 \textit{ESIObjectHistoryBuilderImpl}  
This class is the default implementation of the ESIObjectHistoryRenderer interface.
The class has two attributes that are accessible to subclasses:

3.4.4.4.1  groupOutName : String
The name of the Info*Engine Group object to create on output. Although the attribute has protected setters and getters for the convenience of customizers, the value of this attribute is normally established using the setData() method.

3.4.4.4.2  groupOut : Group
This attribute is the Info*Engine group that is created on output. The attribute getter method performs lazy initialization of the attribute. A protected setter method is provided for the convenience of customizers.

These are the available methods of the class:

3.4.4.4.3  setData(groupOutName : String)
Stores the argument in the groupOutName attribute.

3.4.4.4  addElement(ra : ReleaseActivity)
Maps the argument. The method calls the Windchill ESI mapper to create an Info*Engine element from the argument. The method calls adjustElement(), then adds the result to the output Group object in the groupOut attribute.

3.4.4.5  getGroup() : Group
Returns the groupOut attribute. The method returns the result of getGroupOut(), which is a lazy initializer.

3.4.4.6  adjustElement(element :.Element, object : Persistable)
Modifies the contents of an element before adding it to the output group. This method is provided as a convenience to subclasses. Default behavior is to do nothing.

3.4.4.5 ESIObjectHistoryRendererFactory
This class provides methods to obtain instances of the appropriate implementation classes for ESIObjectHistoryRenderer and ESIObjectHistoryBuilder.

3.4.4.5.1  newESIObjectHistoryRenderer(ufid : String) : ESIObjectHistoryRenderer
Creates a new object from the appropriate ESIObjectHistoryRenderer implementation class. Obtains the name of the implementation class from ESI properties. Invokes setData() on the object. If the class cannot be instantiated or the setData() method throws an exception, this method allows the exception to pass through.

3.4.4.5.2  newESIObjectHistoryBuilder(groupOut : String) : ESIObjectHistoryBuilder
Creates a new object from the appropriate ESIObjectHistoryBuilder implementation class. Obtains the name of the implementation class from Windchill ESI properties. Invokes setData() on the object. If the class cannot be instantiated or the setData() method throws an exception, this method allows the exception to pass through.
3.4.5 Info*Engine Tasks

3.4.5.1 GetActivities
Given the UFID of an ESITransaction object, the task returns all of the ReleaseActivity objects that are associated with that transaction.

3.4.5.2 GetTargetsWithHistory
Given the UFID of a wt.fc.Persistable object, the task returns all of the ESITarget objects that are associated with that object. Additionally, for each ESITarget that is associated, if a ReleaseActivity exists for that Persistable and that distribution target, the task returns the ReleaseActivity status. The task output is created by the ESITransactionUtility getTargetsWithHistory method.

3.4.5.3 GetHistory
Obtains a Group of mapped ReleaseActivity objects given the UFID of Windchill Persistable object. The task calls the getHistory() method of the StandardESIService via the Windchill adapter Apply-Service webject.

3.4.6 Customization Points

3.4.6.1 Extendable classes
Extendable Windchill ESI classes, described in the Transaction Management section of this document, must be defined in Windchill ESI properties if they are extended. The following classes are designed to permit extensions by customizers:

- ESITransaction
  Property name: com.ptc.windchill.esi.txn.class.ESITransaction
  Default value: com.ptc.windchill.esi.txn.ESITransaction

- ESIRelatedTransaction
  Property name: com.ptc.windchill.esi.txn.class.ESIRelatedTransaction
  Default value: com.ptc.windchill.esi.txn.ESIRelatedTransaction

- ESITransactionObjectLink
  Property name: com.ptc.windchill.esi.txn.class.ESITransactionObjectLink
  Default value: com.ptc.windchill.esi.txn.ESITransactionObjectLink

- ReleaseActivity
  Property name: com.ptc.windchill.esi.txn.class.ReleaseActivity
  Default value: com.ptc.windchill.esi.txn.ReleaseActivity
• ESITransactionRelease
  Property name: com.ptc.windchill.esi.txn.class.ESITransactionRelease
  Default value: com.ptc.windchill.esi.txn.ESITransactionRelease
• ESITransactionRelease
  Property name: com.ptc.windchill.esi.rnd.class.objectHistoryBuilder
  Default value: com.ptc.windchill.esi.rnd.ESIObjectHistoryBuilderImpl
• ESITransactionRelease
  Property name: com.ptc.windchill.esi.rnd.class.objectHistoryRenderer
  Default value: com.ptc.windchill.esi.rnd.ESIObjectHistoryRendererImpl

3.4.6.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI Transaction Management. The options are:

• Allow deletion of ESITarget objects that have associated release history (associated ReleaseActivity objects).
  Property name: com.ptc.windchill.esi.txn.allowTargetDelete.ReleaseActivity
  Default value: false

• Allow deletion of WTPart objects that have associated release history (associated ReleaseActivity objects).
  Property name: com.ptc.windchill.esi.txn.allowPartDelete.ReleaseActivity
  Default value: false
  Note: The default is set to false so that parts which have publishing history may not be deleted. This prevents Windchill PDM from becoming out-of-synch with the distribution targets. You may desire to set this value to true if you desire to purge obsolete information from your database.

• Allow deletion of WTPart objects that have associated ESITransaction objects.
  Property name: com.ptc.windchill.esi.txn.allowPartDelete.ESITransaction
  Default value: false
  Note: The default is set to false so that parts that have publishing history are not deleted. This prevents Windchill PDM from becoming out-
of-synch with the distribution targets. You may desire to set this value to true if you desire to purge obsolete information from your database.

- If PostResult RPC request have messages with a length greater than the allowable size of 4000 bytes, this allows the message be truncated. If the property is true, the first 4000 bytes are saved in the database. If the property is false, and a message of more than 4000 bytes is received, an exception is thrown and the RPC request is rejected.

  **Property name:**  
  com.ptc.windchill.esi.txn.truncateExcessLengthMessage  
  **Default value:** true

- Allows Windchill ESI services to create ReleaseActivity objects when the status of the request is failure. This property is provided for the convenience of customizers who want to provide their own transaction GUI. Setting the property to false causes Windchill ESI to ignore PostResult RPC request for failed subtransactions.

  **Property name:** com.ptc.windchill.esi.result.logFailSubTran  
  **Default value:** true

- Allows Windchill ESI services to create ReleaseActivity objects when the status of the request is a success. This property is provided for the convenience of customizers who want to provide their own tracking of Windchill ESI publishing. Setting the property to false causes Windchill ESI to ignore PostResult RPC request for successful subtransactions. This should only be done with extreme caution; it causes the Windchill ESI RPCs GetPart, GetBOM, and GetECN to always decide that objects have not been previously published.

  **Property name:** com.ptc.windchill.esi.result.logSuccessSubTran  
  **Default value:** true

- Define the location of the map file that is used by ESIObjectBuilderImpl objects to map output.

  **Property name:** com.ptc.windchill.esi.rnd.objectHistory.mapFile  
  **Default value:** $CODEBASE$/rnd/ObjectHistory.map

  **Note:** If any map file name begins with the characters, "$CODEBASE$", Windchill ESI services replaces "$CODEBASE$" with the system file directory name where Windchill ESI is located.
3.5 Publishing ECNs

3.5.1 ECN Relationships

Change Orders contain one or more Change Activities. Each Change Activity can contain zero or more Changeable objects, each reflecting a changed business object. Changeable objects can be of the following types (see Figure 29, below):

- Part (separable, component)
- Part (BOM or assembly)
- Document (via association with parts)

![ECN Structure Diagram](image)

**Figure 29**  ECN Structure
Figure 30 Relationships of Changeables

Figure 30, above, illustrates the relationship between Changeable2 and ChangeActivity2. A Change Activity can have zero or more Changeables. Each Changeable can be associated with zero or more Change Activities.

There are two associations between Changeable2 and ChangeActivity2. The first association is represented by AffectedActivityData, and associates a Change Activity with a business object to be changed as part of the change activity. The second association is represented by ChangeRecord2, and associates a Change Activity with the business object changed as part of the change activity. The former links the business object before the change, and the latter links the business object after the change.

3.5.2 Publish ECN

Publish ECN is the process by which Windchill ESI publishes Change Orders to the appropriate distribution targets.

Windchill ESI services supports publishing parts that are listed among the Change Order’s changeable objects. Along with the part detail, Windchill ESI services
publishes the part’s dependent objects (for example, documents, alternates) if the
RPCs request those objects. Windchill ESI only publishes those objects that have
not already been published. Windchill ESI publishes these dependent objects
regardless of whether or not they appear as changeables on the Change Order. If
the part is an assembly, Windchill ESI publishes the assembly part (as a BOM)
and its component parts.

Windchill ESI does not publish documents listed as changeable objects on the
Change Order. For a document to be published, that document must be associated
with a part that is a changeable on the Change Order.

3.5.3 Added Versus Updated ECN Determination
If the Change Order has never been published to a given distribution target,
Windchill ESI creates an entry in the <AddedECN> section of the Response.
Otherwise, Windchill ESI creates an entry in the <UpdatedECN> section of the
Response.

The <ECNHeader> entry contains no information about the changeable objects on
the ECN. Information about parts is included as a separate entry in the
<AddedParts>, <ChangeParts>, <AddedBOMs>, or <ChangedBOMs> sections of
the Response.

The generation of the part’s detail, as well as that for the part’s dependent objects,
uses the same process as that followed when publishing parts outside the context
of a Change Order. For details, see the Publishing BOMs section of this
document and the Publishing Parts section of this document.
3.5.4 Classes

These are the Windchill ESI services classes that publish ECNs.

![Diagram of classes and interfaces]

**Figure 31  Windchill ESI ECN Release Support**

Figure 31, above, displays the primary classes and interfaces used for the Publish ECN process. All Windchill ESI release activities utilize the `ResponseRenderer` interface that provides an abstract mechanism for building the Response representing the set of objects included in the release.

For Publish ECN, there are two such renderers – the `ChangeOrder2Renderer` and the `ChangeHeaderRenderer`. Both interfaces extend the `ResponseRenderer` interface and add a `setData()` method with arguments specific to each rendering task. All Windchill ESI Renderers follow this pattern.
3.5.4.1 ChangeOrder2 Renderer

The ChangeOrder2Renderer interface is the primary interface for Release ECN. These are the methods of the interface:

3.5.4.1.1 setData(objectID : String, delta : boolean, componentParts : boolean, alternates : boolean, substitutes : boolean, referenceDesignators : boolean, lineNumbers : boolean, multiLevel : String, documents : boolean, configSpec : String, auth : String, wcAdapter : String, transactionID : String)

Establishes the renderer’s state so that it can produce output correctly.

3.5.4.2 ESIChangeOrder2Renderer

The ESIChangeOrder2Renderer class is the out-of-the-box implementation for this interface. The ESIChangeOrder2Renderer class utilizes the ChangeHeaderRenderer interface for constructing the ECN Header in the response. For each part listed as a changeable on the Change Order, this class determines whether the part is an assembly (BOM) or a separate part and then delegates to either a BOMRenderer or a PartRenderer respectively, as shown in Figure 32.

The class is intended to be extended. The class has several attributes that are available to subclasses. Each attribute has a protected getter method. The attributes do not have a setter method because their values are set by the setData() method. These are the available attributes of the class:

3.5.4.2.1 objectID : String

The UFID of a WTChangeOrder2. This is a reference to the ECN being processed.

3.5.4.2.2 delta : boolean

Defines whether or not the RPC response contains only information that has changed since the last time objects were released.

3.5.4.2.3 componentParts : boolean

When BOMs are published, should parts be created for BOM components?

3.5.4.2.4 alternates : boolean

When parts are published, should the alternates of the parts (if any are defined) be published?

3.5.4.2.5 substitutes : boolean

If "true", the renderer returns information about substitute parts on assemblies.

3.5.4.2.6 referenceDesignators : boolean

If "true", renderer returns information about Reference Designators on assemblies.

3.5.4.2.7 lineNumbers : boolean

If "true", returns line number information; otherwise no line number information is returned. If line numbers are not supported and the assembly has more than one component that is the same part, the result only lists the component once. In this
case, the quantity associated with the component is the sum of the quantities of all of the components on the assembly that are the same part.

3.5.4.2.8 multiLevel : String
The number of levels of child assemblies to return for each changeable on the ECN.

3.5.4.2.9 documents : boolean
If "true", returns associated documents info, default is "false".

3.5.4.2.10 configSpec : String
The class name of a subclass of wt.part.WTPartConfigSpec. Used to find the appropriate revision of a part for a given part master object.

3.5.4.2.11 auth : String
Authentication information to use when invoking Info*Engine tasks.

3.5.4.2.12 wcAdapter : String
The name of the Windchill Adapter to use when invoking Info*Engine tasks.

3.5.4.2.13 transactionID : String
The number of the Windchill ESI transaction to associate with the output from renderer. If this value is null, the VDB Builder does not flag any of the objects in the RPC response as "pending". If the value identifies an existing Windchill ESI transaction, the VDB Builder flags the objects in the RPC response as "pending". Pending objects are not published until the transaction identified by this attribute has completed.

The class has the following available methods:

3.5.4.2.14 validate()
This method makes sure that WTChangeOrder2 is in proper state for publishing to a target enterprise system. If not, an exception is thrown.

3.5.4.2.15 buildGroups()
The method:

• Calls ESIChangeHeaderRenderer to create the ECN header group
• Calls ESIChangeEvaluator to determine whether change items are parts or BOMs
• Calls ESIWTPartRenderer with the list of parts
• Calls BOMRenderer recursively with BOMs.
3.5.4.3 ChangeHeaderRenderer

This interface defines the methods that all Windchill ESI change header renderers must implement. These are the methods:

3.5.4.3.1  \textit{setData(chgOrder : WTChangeOrder2, targets : Collection, auth : String, wcAdapter : String)}

Provides the data required for the renderer to operate properly.

\begin{figure}[h]
\centering
\includegraphics{windchill-esi-change-order-renderer}
\caption{Windchill ESI Change Order Renderer}
\end{figure}
3.5.4.4 ESIChangeHeaderRenderer

This class manages the generation of an ECN header. It is the default implementation of the ChangeHeaderRenderer interface.

The class is intended to be extended. The class has several attributes that are available to subclasses. Each attribute has a protected getter method. The attributes do not have a setter method because their values are set by the setData() method. These are the available attributes of the class:

3.5.4.4.1 targets : Collection
A collection of ESITarget objects that are of interest to the renderer.

3.5.4.4.2 auth : String
Authentication information to use when invoking Info*Engine tasks.

3.5.4.4.3 wcAdapter : String
The name of the Windchill Adapter to use when invoking Info*Engine tasks.
The class has these available methods:

3.5.4.4.4  **setData(chgOrder : WTChangeOrder2, targets : Collection, auth : String, wcAdapter : String)**

Provides the data required for the renderer to operate properly.

3.5.4.4.5  **validate()**

Makes sure that targets attribute and changeOrder attribute are OK.

3.5.4.4.6  **buildGroups()**

Maps the input and save the resulting group in the VDB Builder.

3.5.4.4.7  **adjustElement(element : Element, group : String, changeOrder : WTChangeOrder2, targets : Collection) : Element**

Allows subclasses to modify the result of this renderer before returning the output to the caller.

3.5.4.5  **ESIChangeOrgFinder**

This class provides methods to get a collection of distribution targets for ECN objects. The class is not intended to be extended by customizers. It may be used by customizers. These are the available methods of the class:

3.5.4.5.1  **getTargets(persistable : Persistable) : Collection**

Returns a collection of ESITarget objects associated with the argument. Each distribution target appears only once in the collection.

3.5.4.5.2  **getTargets(parts : Collection) : Collection**

Returns a collection of ESITarget objects that are associated with the objects contained by the argument. Each distribution target appears only once in the collection.

3.5.4.5.3  **getTargets(changeOrder : WTChangeOrder2) : Collection**

Returns a collection of ESITarget objects that are associated with the changeable objects contained by the argument. Each distribution target appears only once in the collection.

3.5.4.5.4  **getTargets(changeActivity : WTChangeActivity2) : Collection**

Returns a collection of ESITarget objects that are associated with the changeable objects contained by the argument. Each distribution target appears only once in the collection.
3.5.4.6 **ChangeOrderRendererFactory**

This class returns instances of the appropriate renderer implementation classes.

3.5.4.6.1  *newChangeHeaderRenderer*(): *ChangeHeaderRenderer*

This method returns an instance of the appropriate ChangeHeaderRenderer implementation class. It obtains the class name from Windchill ESI properties.

3.5.4.6.2  *newChangeOrder2Renderer*(): *ChangeOrder2Renderer*

This method returns an instance of the appropriate ChangeOrder2Renderer implementation class. It obtains the class name from Windchill ESI properties.
Figure 35, Figure 36, and Figure 37, below, show the sequence of operations for the Release ECN process.

Release ECN

1: getECN()
2: newVdbBuilder()
3: newChangeOrder2Renderer()
4: setData()
5: render(VdbBuilder)

Sequence Diagram: ecn / Render ECN

Figure 35  Publishing an ECN
Render ECN

Figure 36  Rendering an ECN
3.5.5 Customization Points

3.5.5.1 Extendable classes

A Windchill ESI property controls the name of the classes used for rendering. By modifying these properties, a customizer can integrate specialized implementations. Extendable Windchill ESI classes described in the Publishing ECNs section of this document must be defined in Windchill ESI properties if they are extended. Customizers may also modify the logic for rendering Change Orders by providing their own specialized classes.

Figure 37  Rendering the Changeables on an ECN
These classes may extend and override the default implementation, or these classes may simply support the base interfaces defined. The following classes are designed to permit extensions by customizers:

- **ChangeHeaderRenderer**
  
  **Property name:** com.ptc.windchill.esi.ECN.changeHeaderRenderer  
  **Default value:** com.ptc.windchill.esi.ecn.ESIChangeHeaderRenderer

- **ChangeOrder2Renderer**
  
  **Property name:** com.ptc.windchill.esi.ECN.changeOrder2Renderer  
  **Default value:** com.ptc.windchill.esi.ecn.ESIChangeOrder2Renderer

### 3.5.5.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI ECN support. The options are:

- Define the name of the Info*Engine task to execute when obtaining ECN information from the Windchill Adapter.
  
  **Property name:** com.ptc.windchill.esi.ECN.queryTask  
  **Default value:** com/ptc/windchill/esi/QueryObject.xml  
  **Note:** If any map file name begins with the characters, "$CODEBASE$", Windchill ESI services replaces "$CODEBASE$" with the system file directory name where Windchill ESI is located.

- Define the name of the map file to use when mapping ECN headers.
  
  **Property name:** com.ptc.windchill.esi.ECN.changeHeader.mapFile  
  **Default value:** $CODEBASE$/ecn/ChangeHeader.map

### 3.6 Publishing BOMs

#### 3.6.1 Overview

Windchill ESI services provides a SOAP RPC, *GetBOM* to create Bill of Material (BOM) information from Windchill product structure data. Windchill ESI services provides a GetBOM Info*Engine task to support the RPC request. Windchill ESI services provides Java classes that extract the Windchill data and format the RPC response. Windchill ESI services provides access to the major features of the GetBOM RPC support classes via the Windchill Adapter. The Windchill ESI services BOM functionality is available to the engineering change notice (ECN) support in Windchill ESI services.
The Java classes are usable, and in many cases, extendable by customizers. The Windchill Information Modeler supports the publicly available methods and attributes of the classes. Javadoc descriptions of the supported classes are provided.

A BOM is a composite structure comprised of the following elements:

- One BOM Header
- One or more BOM components

  **Note:** Target enterprise systems allow BOMs that have all components removed. Most major target enterprise systems do not allow you to create a BOM with no components.

- Zero or more substitutes for each component on the BOM
- Zero or more reference designators for the components on the BOM

Windchill ESI services declares renderer interfaces for the BOM and each of the elements of the BOM structure (see Figure 38). Each interface extends the ResponseRenderer interface. Additionally, since Windchill ESI is required to navigate multilevel BOM structures, Windchill ESI services declares an interface, `BOMTreeNavigator`. The interface defines the methods necessary to obtain the BOMs that are declared in a multilevel BOM structure. These interfaces are described in detail in the BOM Classes section of this document.
BOM Interface Relationships

Figure 38  BOM Renderer Interfaces
3.6.2 BOM Classes

The following classes create the GetBOM RPC output:

![BOM Renderer Classes Diagram]

**Figure 39  BOM Renderer Classes**

3.6.2.1 BOMRenderer

This is the standard interface for all BOM renderers. These are the methods of the interface:


Initializes the BOM renderer to a ready-to-run state.

3.6.2.2 BOMRendererImpl

This class is the default implementation of BOMRenderer. It extends the `ResponseRendererImpl` and inherits the capabilities of that class. The
**BOMRendererImpl** class is intended to be extended by customizers. It has several attributes that are accessible to subclasses in read-only mode. The attributes have protected getter methods. Because the renderer state is expected to be established by the setData() method, the attributes do not have setter methods. These are the available attributes of the class:

3.6.2.2.1 **obid : String**  
The UFID of the Windchill assembly part to be rendered.

3.6.2.2.2 **delta : boolean**  
Allows the output to contain only items that have changed since the last time an object was released.

3.6.2.2.3 **bomPart : boolean**  
Allows the output to contain part creation commands, as opposed to BOM operations.

3.6.2.2.4 **componentParts : boolean**  
Allows the output to contain part creation commands, as opposed to BOM operations.

3.6.2.2.5 **alternates : boolean**  
Allows the output to also include the alternates of the parts if either bomPart = true or componentParts = true.

3.6.2.2.6 **substitutes : boolean**  
Allows substitutes to be included in the output.

3.6.2.2.7 **referenceDesignators : boolean**  
Allows reference designators to be included in the output.

3.6.2.2.8 **lineNumbers : boolean**  
Allows line numbers to be included in the output.

3.6.2.2.9 **multilevel : int**  
Determines how many levels to traverse when extracting multi-level BOMs.

3.6.2.2.10 **documents : boolean**  
Allows documents to be included in the output.

3.6.2.2.11 **configspec : String**  
The class name of the Windchill configuration specification to use when navigating product structures.

3.6.2.2.12 **authentication : String**  
Authentication information to use when using the Windchill Adapter for obtaining information.
3.6.2.13  **adapterName : String**

The name of the Windchill Adapter to use when obtaining information from Windchill.

3.6.2.14  **transactionID : Integer**

The number of a Windchill ESI transaction.

These are the available methods of the class:


Provides the data required to render a BOM.

3.6.2.16  **validate()**

Makes sure that the obid attribute is not null.

3.6.2.17  **buildGroups()**

Builds the renderer output. Does the following:

1. Obtains the list of parts to be rendered. Uses the services of a BOMTreeNavigator if multilevel > 1.

2. For each part obtained:
   - Finds the last previous released version of the part for each distribution target.
   - For each previous version : current version pair, obtains the difference from the StandardWTPartService. If there is no previous version, the entire part structure is returned.
   - Calls the BOM header renderer.
   - Calls the BOM component renderer.
   - Optionally calls the substitute renderer.
   - Optionally calls the reference designator renderer
   - Calls adjustElement().

3. If parts are requested, calls ESIPartUtility.getPart() to render the parts on the BOM.

3.6.2.3  **BOMHeaderRenderer**

This is the standard interface for all BOM header renderers. These are the methods of the interface:

3.6.2.3.1  **setData(groupName : String, part : WTPart, tgts : Collection, effHandler : ESIEffectivityHandler)**

Stores the attributes and reset any variables.
3.6.2.4 BOMHeaderRendererImpl

This class is the default implementation of the BOMHeaderRender interface. It extends the ResponseRendererImpl and inherits the capabilities of that class. The BOMHeaderRendererImpl class is intended to be extended by customizers. It has several attributes that are accessible to subclasses in read-only mode. The attributes have protected getter methods. Because the renderer state is expected to be established by the setData() method, the attributes do not have setter methods.

These are the available attributes of the class:

3.6.2.4.1 targets : Collection
A collection ESITarget objects.

3.6.2.4.2 groupName : String
The logical name of a VdbGroup.

3.6.2.4.3 part : WTPart
The assembly part for this BOM.

These are the available methods of the class:

3.6.2.4.4 setData(groupName : String, part : WTPart, tgts : Collection, effHandler : ESIEffectivityHandler)
Stores the attributes and reset any variables.

3.6.2.4.5 adjustElement(elem : Element, group : String, part : WTPart, eff : Eff, targets : Collection) : Element
Allows subclasses to fix the contents of an element before it's added to the output group(s). Default behavior is to return the argument.

3.6.2.4.6 validate()
Makes sure part, groupName, and distribution targets are correct.

3.6.2.4.7 buildGroups()
Builds the renderer output. Does the following:

1. Obtains the distribution targets for part.
2. Maps the output.
3. Calls adjustObjectID() on the base class.
4. Calls adjustTargets() on the base class.
5. Calls adjustEffectivities() on the base class.
6. Calls adjustElement().
7. Passes the element to the VdbBuilder.
3.6.2.5 BOMPartRenderer

This interface defines the common methods for all BOM renderers that deal with one or more WTPart objects. These are the methods of the interface:

3.6.2.5.1 **setData(root : WTPart, part : WTPart, deltaParts : Collection[], tgts : Collection, lineNumbers : boolean)**

Sets the renderer's state so that it is ready to process.

3.6.2.5.2 **getRenderedParts() : Collection**

Returns a collection of WTPart objects which represent the parts used for rendering.

3.6.2.6 ESIWTPartRenderer

This is an abstract class that provides services to receive, and access common data elements. The class must be extended to be used. It extends the `ResponseRendererImpl` and inherits the capabilities of that class. The `ESIWTPartRenderer` class is intended to be extended by customizers. It has several
attributes that are accessible to subclasses in read-write mode. The attributes have protected getter and setter methods.

These are the available attributes of the class:

3.6.2.6.1  \textit{enforceLineNumbers} : boolean
This static variable is set by the static initializer of the class. It defines whether an exception be thrown if a BOM has no line numbers.

3.6.2.6.2  \textit{deltaParts} : Collection[]
An array of Collections containing WTPart objects.

3.6.2.6.3  \textit{targets} : Collection
A collection of ESITarget objects.

3.6.2.6.4  \textit{lineNumberEnabled} : boolean
Are line numbers to be included on the output?

3.6.2.6.5  \textit{part} : WTPart
The part being rendered.

3.6.2.6.6  \textit{root} : WTPart
If the part being rendered is part of a tree of product structures, this attribute contains the top of the tree.

3.6.2.6.7  \textit{partMasters} : java.util.HashSet = new HashSet()
A collection of WTPartMaster objects that have been rendered. Used to maintain a list of parts to pass to the part renderer. This attribute has no setter.

These are the available methods of the class:

3.6.2.6.8  \textit{areLineNumbersEnforced}() : boolean
getter for enforceLineNumbers attribute.

3.6.2.6.9  \textit{getRenderedParts}() : Collection
Returns a collection of WTPartMaster objects which represent the parts used for rendering.

3.6.2.6.10  \textit{addRenderedPart}(pui : PartUsageInfo)
Obtains the part master object from the PartUsageInfo in the argument and add it to the partMasters collection.

3.6.2.6.11  \textit{addRenderedPart}(partMaster : WTPartMaster)
Adds the argument to the partMasters collection.

3.6.2.7  \textbf{BOMComponentRenderer}
This is the standard interface for BOM component renderers. All methods are defined in the super-classes of this interface.
3.6.2.8 BOMComponentRendererImpl

Default implementation of BOMComponentRender. The class is intended to be extended by customizers. These are the available methods of the class:

3.6.2.8.1 setData(root : WTPart, part : WTPart, deltaParts : Collection[], tgts : Collection, lineNumbers : boolean)

Sets the renderer's state so that it is ready to process.

3.6.2.8.2 validate()

Ensures that the object is ready to process. Throws an exception if:
- the part attribute is null.
- The targets attribute is null.
- The deltaParts attribute is null or has a length < 3.

3.6.2.8.3 buildGroups()

Builds the renderer output. For each Collection element in the deltaParts array, iterates the Collection and:
1. Obtains the list of ESITarget objects associated
2. If alternate item groups are required, obtains the appropriate alternate item group from the alternate item group manager.
3. Maps the PartUsageInfo object that is contained in the Collection.
4. Calls adjustObjectID().
5. Calls adjustTargets().
6. Calls adjustElement().
7. Adds the part master to the partMasters collection by calling addRenderedPart().

3.6.2.8.4 adjustElement(elem : Element, group : String, pui : PartUsageInfo, targets : Collection) : Element

Allows subclasses to fix the contents of an element before it's added to the output group(s). Default behavior is to return the argument.

3.6.2.9 SubstituteRenderer

This is the standard interface for substitute renderers. All methods are defined in the super-classes of this interface.

3.6.2.10 SubstituteRendererImpl

Default implementation of SubstituteRenderer. The class is intended to be extended by customizers. These are the available methods of the class:
3.6.2.10.1 `setData(root : WTPart, part : WTPart, deltaParts : Collection[], tgts : Collection, lineNumbers : boolean)`

Sets the renderer's state so that it is ready to process.

3.6.2.10.2 `validate()`

Calls base class validate().

3.6.2.10.3 `buildGroups()`

Produces the renderer's output. Obtains the deltaParts array of Collections from the base class. Iterates each Collection in turn. For each entry in the iterations:

1. Obtains the substitute additions and substitute deletions.
2. Maps the output.
3. If alternate item groups are required, obtains the appropriate alternate item group from the alternate item group manager.
5. Calls `adjustTargets()`
6. Calls `adjustElement()`
7. Calls `addRenderedPart()`

3.6.2.10.4 `adjustElement(elem : Element, group : String, pui : PartUsageInfo, targets : Collection) : Element`

Allows subclasses to fix the contents of an element before it's added to the output group(s). Default behavior is to return the argument.

3.6.2.11 `ReferenceDesignatorRenderer`

This is the standard interface for reference designator renderers. All methods are defined in the super-classes of this interface.

3.6.2.12 `SubstituteRendererImpl`

Default implementation of `ReferenceDesignatorRenderer`. The class is intended to be extended by customizers. These are the available methods of the class:

3.6.2.12.1 `setData(root : WTPart, part : WTPart, deltaParts : Collection[], tgts : Collection, lineNumbers : boolean)`

Sets the renderer's state so that it is ready to process.

3.6.2.12.2 `validate()`

Makes sure that the object is ready to create output. Calls base class method.

3.6.2.12.3 `buildGroups()`

Produces the renderer's output. Obtains the deltaParts array of Collections from the base class. Iterates each Collection in turn. For each entry in the iterations:

1. Obtains the reference designator additions and reference designator deletions.
2. Maps the output.
3. Calls adjustObjectID().
4. Calls adjustTargets()
5. Calls adjustElement()
6. Calls addRenderedPart()

### 3.6.2.12.4 adjustElement(elem : Element, group : String, pui : PartUsageInfo, targets : Collection) : Element

Allows subclasses to fix the contents of an element before it's added to the output group(s). Default behavior is to return the argument.

#### Figure 41  BOM Renderer Factory

### 3.6.2.13 BOMRendererFactory

This class provides methods to obtain the correct instance of the renderers that create BOM output. The class in not intended to be modified by customizers. These are the methods of the class:

#### 3.6.2.13.1 <<Static>> newBOMOrgFilter() : BOMOrgFilter

Returns an instance of BOMOrgFilter (or an appropriate subclass). Obtains the name of the class from ESIProperties.
3.6.2.13.2 \textit{<<Static>> newBOMRenderer() : BOMRenderer}
   Returns an instance of the appropriate implementation of BOMRenderer. Obtains the name of the class from ESIProperties.

3.6.2.13.3 \textit{<<Static>> newBOMHeaderRenderer() : BOMHeaderRenderer}
   Returns an instance of the appropriate implementation of BOMHeaderRenderer. Obtains the name of the class from ESIProperties.

3.6.2.13.4 \textit{<<Static>> newBOMComponentRenderer() : BOMComponentRenderer}
   Returns an instance of the appropriate implementation of BOMComponentRenderer. Obtains the name of the class from ESIProperties.

3.6.2.13.5 \textit{<<Static>> newSubstituteRenderer() : SubstituteRenderer}
   Returns an instance of the appropriate implementation of SubstituteRenderer. Obtains the name of the class from ESIProperties.

3.6.2.13.6 \textit{<<Static>> newReferenceDesignatorRenderer() : ReferenceDesignatorRenderer}
   Returns an instance of the appropriate implementation of ReferenceDesignatorRenderer. Obtains the name of the class from ESIProperties.
BOM Tree Navigators

3.6.2.14 BOMTreeNavigator

Defines the methods that are available on classes that parse the results of a Query-Tree webject invocation. These are the available methods of the interface:

3.6.2.14.1 getBomNodeUfids() : Collection

Returns a collection part UFIDs for all of the nodes in the tree that are BOMs.

3.6.2.14.2 getLeafNodeUfids() : Collection

Returns a collection part UFIDs for all of the nodes in the tree that are not BOMs.

3.6.2.14.3 setData(root : WTPart, depth : int, configSpec : String, auth : String, adapter : String)

Sets the attributes on the object.

Figure 42   BOM Tree Navigator
3.6.2.15 BOMTreeNavigatorFactory

Creates the appropriate BOMTreeNavigator and sets data for it. These are the methods of the class:

3.6.2.15.1 <<Static>> newBOMTreeNavigator(root : WTPart, depth : Integer, configSpec : String, auth : String, adapter : String) : BOMTreeNavigator

Creates the appropriate BOMTreeNavigator. Reads ESIProperties to obtain the name of the class to instantiate. If the class is BOMTreeNavigatorImpl (or a subclass of BOMTreeNavigatorImpl), the method calls setData() on the object before returning it.

3.6.2.16 BOMTreeNavigatorImpl

This class is the default implementation of the BOMTreeNavigator interface. The BOMTreeNavigatorImpl class is intended to be extended by customizers. It has several attributes that are accessible to subclasses in read-only mode. The attributes have protected getter methods. Because the renderer state is expected to be established by the setData() method, the attributes do not have setter methods unless otherwise noted. These are the available attributes of the class:

3.6.2.16.1 bomNodeUfids : Collection
A collection WTPart UFIDs that point to assembly parts. Initialized to null. Set by processTree(). Has a setter method.

3.6.2.16.2 bomLeafUfids : Collection
A collection WTPart UFIDs that point to component parts. Initialized to null. Set by processTree(). Has a setter method.

3.6.2.16.3 root : WTPart
The top-level assembly part.

3.6.2.16.4 configspec : String
The name of a configspec to use for navigating the tree. If not set, the query tree webject uses the default.

3.6.2.16.5 depth : int
The number of levels to navigate.

3.6.2.16.6 adapter : String
The name of the Windchill adapter to pass to the Query-Tree webject.

3.6.2.16.7 auth : String
The authentication information to pass to the Query-Tree webject.

These are the methods of the class:

3.6.2.16.8 processTree()
If the bomNodeUfids attribute is null:
• calls validate() to ensure that the attribute state contains all required information
• call the Windchill Adapter Query-Tree webject
• Parse the webject result and obtain all parts that have a child. Add those parts to the bomNodeUfids collection
• for each element that has no children, determine whether the part is a BOM. If so, add the element to the bomNodeUfids attribute. If not, add the element to the bomLeafUfids attribute.

3.6.2.16.9 setData(root : WTPart, depth : int, configSpec : String, auth : String, adapter : String)

Sets the attributes on the object. Throws a WTException if:
• the root argument is null
• the depth argument is < 1
• the auth argument is null or blank
• the adapter argument is null or blank

3.6.2.16.10 validate()

Makes sure all attributes are in the correct state. Throws a WTException if:
• the root attribute is null
• the depth attribute is < 1
• the auth attribute is null or blank or spaces
• the adapter attribute is null or blank or spaces

BOM Utility

![Figure 43 BOM Utility](image)

3.6.2.17 BOMUtility

This class in the interface to the GetBOM RPC processor classes. It exposes the publicly available methods of the package. These are those methods:
3.6.2.17.1 getBOMUFids(baseUFID : String, depth : int = 1) : Collection
   Returns a list of multilevel BOM UFIDs. If depth is 1, returns the root part UFID.

3.6.2.17.2 getBOM(objectID : String, delta : boolean, bomPart : boolean, componentParts : boolean, 
   alternates : boolean, substitutes : boolean, referenceDesignators : boolean, lineNumbers : boolean, 
   multilLevel : String, documents : boolean, configSpec : String, auth : String, wcAdapter : String, 
   transactionID : String) : IeCollection
   This method is the GetBOM RPC processor. It validates arguments, creates a VDBBuilder and then renders the BOM.

3.6.2.17.3 getBOM(part : WTPart, delta : boolean, bomPart : boolean, componentParts : boolean, 
   alternates : boolean, substitutes : boolean, referenceDesignators : boolean, lineNumbers : boolean, 
   multilLevel : String, documents : boolean, configSpec : String, auth : String, wcAdapter : String, 
   builder : VdbBuilder)
   This is the RPC processor when called from getECN RPC. Validates arguments then creates a VDBBuilder if one hasn't already been provided. Creates a BOM Renderer. Passes control to the renderer. Does not call asVDB() on the renderer because the caller of this method is responsible to do that.

3.6.2.17.4 isBom(part : WTPart) : boolean
   Returns true if the given part has any children (connected to a WTPartMaster via a WTPartUsageLink) or was ever released by Windchill ESI to any distribution target as a BOM. Otherwise returns false.
1. The process begins when Info*Engine receives a request and invokes the Windchill ESI GetBOM task. The task invokes the StandardESIService via the Windchill Adapter Apply-Service webject.

1.1 The StandardESIService creates an instance of BOMUtility and passes the arguments to the BOMUtility instance.

1.1.1 The BOM utility obtains a list of BOMs to process, bases on the arguments of the getBOM method invocation.

1.1.2 The BOM utility creates a VDB builder and a BOM renderer. It passes the VDB builder as an argument to the render() message.

1.1.3 If the method arguments indicate that parts are required, the BOM utility creates an instance of PartRenderer and passes the VDB builder to the PartRenderer instance.

1.1.4 The BOM Utility obtains the VDB from the VDB builder and return the result to the caller.
3.6.3 Customization Points

3.6.3.1 Extendable classes

If classes used by the Windchill ESI renderers described in the Publishing BOMs section of this document are extended, the extensions must be defined in Windchill ESI properties. The following classes are designed to permit extensions by customizers:

- **BOMOrgFilter**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomOrgFilter
  
  **Default value:** com.ptc.windchill.esi.bom.BOMOrgFilter

- **BOMRenderer**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomRenderer
  
  **Default value:** com.ptc.windchill.esi.bom.BOMRendererImpl

- **BOMHeaderRenderer**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomHeaderRenderer
  
  **Default value:** com.ptc.windchill.esi.bom.BOMHeaderRendererImpl

- **BOMComponentRenderer**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomComponentRenderer
  
  **Default value:** com.ptc.windchill.esi.bom.BOMComponentRendererImpl

- **SubstituteRenderer**
  
  **Property name:** com.ptc.windchill.esi.BOM.substituteRenderer
  
  **Default value:** com.ptc.windchill.esi.bom.SubstituteRendererImpl

- **BOMTreeNavigatorFactory**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomTreeNavigatorFactory
  
  **Default value:** com.ptc.windchill.esi.bom.BOMTreeNavigatorFactory

- **BOMTreeNavigator**
  
  **Property name:** com.ptc.windchill.esi.BOM.bomTreeNavigator
  
  **Default value:** com.ptc.windchill.esi.bom.BOMTreeNavigatorImpl

- **ReferenceDesignatorRenderer**
  
  **Property name:** com.ptc.windchill.esi.BOM.referenceDesignatorRenderer
  
  **Default value:** com.ptc.windchill.esi.bom.ReferenceDesignatorRendererImpl
3.6.3.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI BOM renderer support. The options are:

- Define whether Windchill ESI should create alternate item groups. Alternate Item Groups are required to provide compatibility with SAP R/3 ERP. SAP R/3 manages BOM substitutes by assigning an alternate item group number to all BOM components which have substitutes. SAP R/3 determines that parts with the same alternate item group number are interchangeable.

  **Property name:** com.ptc.windchill.esi.BOM.enableAlternateItemGroups  
  **Default value:** true

- Define whether Windchill ESI should throw an exception whenever a product structure is found that has no line number information. Standard Windchill behavior is to allow both product structures with line numbers and product structures without line numbers. Any one product structure must have line numbers on all of its components or no line numbers on any of its components. The standard Windchill behavior is the equivalent of setting the property to false. In cases where the target enterprise system cannot handle BOMs without line numbers, the property should be set to true.

  **Property name:** com.ptc.windchill.esi.BOM.enforceLineNumbers  
  **Default value:** false

- Define whether Windchill ESI should throw an exception whenever a product structure is found with components having zero quantities. The default behavior is to throw an exception.

  **Property name:** com.ptc.windchill.esi.BOM.requireComponentQuantity  
  **Default value:** true

- Define the Info*Engine task that is used to obtain multi-level BOM information.

  **Property name:** com.ptc.windchill.esi.BOM.treeQueryTask  
  **Default value:** com/ptc/windchill/esi/QueryTree.xml

  **Note:** If any map file name begins with the characters, "$CODEBASE$", Windchill ESI services replaces "$CODEBASE$" with the system file directory name where Windchill ESI is located.

- Define the name of the map file to use when mapping BOM components.

  **Property name:** com.ptc.windchill.esi.BOM.component.mapFile  
  **Default value:** $CODEBASE$/bom/BOMComponent.map
• Define the name of the map file to use when mapping BOM headers.
  Property name: com.ptc.windchill.esi.BOM.header.mapFile
  Default value: $CODEBASE$/bom/BOMHeader.map

• Define the name of the map file to use when mapping BOM substitutes.
  Property name: com.ptc.windchill.esi.BOM.substitute.mapFile
  Default value: $CODEBASE$/bom/Substitute.map

• Define the name of the map file to use when mapping BOM reference designators.
  Property name: com.ptc.windchill.esi.BOM.refDes.mapFile
  Default value: $CODEBASE$/bom/RefDesignator.map

3.7 Publishing Parts

3.7.1 Overview

When given the identity of an existing Windchill part, the Windchill ESI part renderer obtains the required objects, extract information from them, and put the results in Info*Engine groups that identify whether the parts are to be added, updated or removed in the target enterprise system. If alternate parts are requested, additional groups contain alternate links.

For a part to be successfully published to a target enterprise system, the part must be associated to one or more ESITarget objects. Alternate links are only returned for those parts and alternates that share assignments to a distribution target. The UnchangedParts and DeletedParts groups are always empty.

If documents are requested, the part renderer calls the Windchill ESI document utility to release any appropriate document information. See the Publishing Documents section of this document, for more information about publishing documents.
### 3.7.2 Classes

#### Part Response Renderers

- **ResponseRendererImpl**
  - `render()` (from rnd)

- **ResponseRenderer**
  - `render()` (from md)

- **ESIPartRenderer**
  - `render()` (from part)

- **PartRendererFactory**
  - `newPartRenderer()`

- **PartRenderer**
  - `setData()`
  - `initialize()`

- **ESIPartUtility**
  - `getPart()`
  - `getPartIteration()`

- **ESIAbstractPart**
  - `getMaster()`
  - `ESIAbstractPart()`
  - `ESIAbstractPart()`

- **ESIPartMasterIteration**
  - `newESIPartMasterIteration()`

- **WTPart**
  - `current` (from part)
  - `previous` (from part)

- **WTPartMaster**
  - `master` (from part)

---

**Figure 45** Windchill ESI Part Release Response Renderers
3.7.2.1 PartRendererFactory

The PartRendererFactory creates instances of the appropriate Windchill ESI part renderer. The factory obtains the name of the class to instantiate from Windchill ESI properties.

3.7.2.1.1 newPartRenderer() : PartRenderer

Returns the appropriate PartRenderer implementation. Obtains the name of the class to instantiate from Windchill ESI properties.

3.7.2.2 PartRenderer

The PartRenderer interface defines the methods that are available on all Windchill ESI part renderer implementation classes. These are the available methods of the class:

3.7.2.2.1 setData(root : WTPart, parts : Collection, buildAlts : boolean, buildDocs : boolean, auth : String, wcAdapter : String, transactionID : String)

Establishes the renderer's state so that it can process requests.

3.7.2.2.2 initialize()

Performs any required initialization processing.
3.7.2.3 ESIPartRenderer

This class is the default implementation of PartRenderer. These are the available methods of the class:

3.7.2.3.1 validate()

Makes sure the object is ready to process. If root, auth, or wcAdapter are null, throws an exception.

3.7.2.3.2 buildGroups()

Output generator for the class. Does the following:

- Obtains the contents of the parts collection.
- Finds the distribution targets associated with the parts.
- Finds the most current released version of the part for each distribution target.
- Compares the current version to the most current version. If they are the same and the effectivity has not changed, the part is ignored.
- Calls the Windchill adapter to obtain all the attributes of the part, if the part is not ignored.
- Maps the result of the Windchill adapter call.
- Calls adjustPartElement()
- Calls the document renderer, if documents are required.

3.7.2.3.3 adjustPartElement(element : com.infoengine.object.factory.Element, group : String, part : wt.part.WTPart, eff : wt.eff.Eff, targets : Collection) : com.infoengine.object.factory.Element

Allows subclasses to fix the contents of an element.

3.7.2.3.4 adjustAlternateElement(element : com.infoengine.object.factory.Element, group : String, part : wt.part.WTPart, targets : Collection) : com.infoengine.object.factory.Element

Allows subclasses to fix the contents of an element for alternate parts.

3.7.2.4 ESIPartUtility

This class is the public interface to the com.ptc.windchill.esi.esipart package. It provides generally useful methods for clients external to the package. These are the available methods of the class:

3.7.2.4.1 getPart(objectID : String, alternates : boolean, documents : boolean, auth : String, wcAdapter : String, transactionID : String) : IeCollection

GetPart RPC processor. Validates arguments then creates a VDBBuilder and a Part Renderer. Passes control to the renderer. Returns the VDBBuilder's VDB.
3.7.2.4.2 \texttt{getPart(root : WTPart, children : Collection, alternates : boolean, documents : boolean, auth : String, wcAdapter : String, builder : VdbBuilder)}

GetPart RPC processor when called from GetBOM or getECN RPC. Validates arguments then creates a VDBBuilder if one hasn't already been provided. Creates a PartRenderer. Passes control to the renderer. Does not call asVDB() on the renderer because the caller of this method is responsible to do that.

3.7.2.4.3 \texttt{getPartIteration(partMasters : Collection, rootPart : WTPart) : Collection}

This method returns a collection of ESIMasterPartAssociation objects given a collection part masters. This method uses the configspec to get the iteration that matches the configspec.

3.7.2.5 \texttt{ESIPartMasterIteration}

This class preserves the relationship between an iteration and a part master. These objects are for the convenience of methods that need to pass collections of part masters and part iterations as arguments or as method returns. These are the available methods of the class:

3.7.2.5.1 \texttt{newESIPartMasterIteration(part : WTPart, partMaster : WTPartMaster) : ESIPartMasterIteration}

Default factory method for ESIMasterPartAssociation

3.7.2.5.2 \texttt{ESIAbstractPart}

Defines common behaviors of Windchill ESI part abstractions. These are the available methods of the class:

3.7.2.5.2 \texttt{getMaster() : WTPartMaster}

Returns the part master associated with the current part.

3.7.2.5.3 \texttt{ESIAbstractPart(previousPart : WTPart, currentPart : WTPart)}

Creates an instance that is properly initialized.

3.7.2.5.4 \texttt{ESIAbstractPart()}

Declares a default constructor so static methods newInstance() can be called.

3.7.2.6 \texttt{ESIPart}

Wraps a WTPart and maintains useful relationships for the benefit of a part renderer. These are the available methods of the class:

3.7.2.6.1 \texttt{newESIPart(ufid : String) : ESIPart}

Creates an instance of ESIPart, given the UFID of a part.

3.7.2.6.2 \texttt{getPartInfo(part : ESIAbstractPart) : QueryResult}

Finds the master and obtain all alternate parts associated with it. Store the alternates in the alternates collection. Store the link in the alternateLinks collection.
3.7.2.7 ESIAlteritatePart
Manages the relationships for part that are alternates to the set of parts to be processed. These are the available methods of the class:

3.7.2.7.1 ESIAlternatePart(previousPart : WTPart, currentPart : WTPart)
Creates an instance that is properly initialized

3.7.2.7.2 ESIAlternatePart()
Declares a default constructor so static methods newInstance() can be called.

3.7.3 GetPart RPC Processing

Figure 47  GetPart RPC Processing

Figure 47 shows the high-level object interactions that occur when Windchill ESI services receives a GetPart RPC request. These are the messages that flow between objects:

1. Info*Engine obtains the RPC request, parses it and passes the parameters to an Info*Engine task provided by Windchill ESI. The task is named, "GetPart". The GetPart task validates its parameters and passes the result to the Windchill ESI Windchill service using the Apply-Service webject. When the webject process is completed, the task formats the output and returns control to Info*Engine. Info*Engine formats the output as an XML document and delivers the result to the requestor.
2. The Windchill ESI service creates an instance of the Windchill ESI Part Utility and passes the request parameters to it. It returns the result of the request made to the newly-created utility object.

3. The utility obtains a VDB builder from the VDB Builder Factory.

4. The utility obtains a part renderer from the Part Renderer Factory.

5. The utility passes the data to the part renderer.

6. The utility tells the part renderer to process its data.

7. The utility obtains the processing result from the VDB builder. It returns the result to the Windchill ESI Windchill service.

### 3.7.4 Customization Points

#### 3.7.4.1 Extendable classes

If classes used by the Windchill ESI renderers, described in the Publishing Parts section of this document, are extended, the extensions must be defined in Windchill ESI properties. The following classes are designed to permit extensions by customizers:

- **WTPart**
  
  **Property name:** com.ptc.windchill.esi.part.class.WTPart
  
  **Default value:** wt.part.WTPart

- **WTPartMaster**
  
  **Property name:** com.ptc.windchill.esi.part.class.WTPartMaster
  
  **Default value:** wt.part.WTPartMaster

- **MasterIteration**
  
  **Property name:** com.ptc.windchill.esi.part.class.MasterIteration
  
  **Default value:** wt.part.MasterIteration

- **WTPartAlternateLink**
  
  **Property name:** com.ptc.windchill.esi.part.class.WTPartAlternateLink
  
  **Default value:** wt.part.WTPartAlternateLink

- **WTPartStandardConfigSpec**
  
  **Property name:** com.ptc.windchill.esi.part.class.configSpec
  
  **Default value:** wt.part.WTPartStandardConfigSpec

- **ESIPartRenderer**
  
  **Property name:** com.ptc.windchill.esi.part.class.PartRenderer
3.7.4.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI part renderer support. The options are:

- Defines the name of the Info*Engine task to invoke when obtaining part information from the Windchill Adapter.
  
  **Property name**: com.ptc.windchill.esi.part.queryPartTask  
  **Default value**: com/ptc/windchill/esi/QueryIterationAndMaster.xml

- Defines the name of the Info*Engine task to invoke when obtaining part alternate information from the Windchill Adapter.
  
  **Property name**: com.ptc.windchill.esi.part.queryAlternateTask  
  **Default value**: com/ptc/windchill/esi/QueryObject.xml

  **Note**: If any map file name begins with the characters, "$CODEBASE$", Windchill ESI services replaces "$CODEBASE$" with the system file directory name where Windchill ESI is located.

- Defines the name of the map file to use when mapping parts.
  
  **Property name**: com.ptc.windchill.esi.part.partMapFile  
  **Default value**: $CODEBASE$/esipart/ESIPart.map

- Defines the name of the map file to use when mapping part alternate links.
  
  **Property name**: com.ptc.windchill.esi.part.alternateLinkMapFile  
  **Default value**: $CODEBASE$/esipart/ESIAlternateLink.map

- Defines the name of the Info*Engine Group that is to contain added parts.
  
  **Property name**: com.ptc.windchill.esi.part.addPartName  
  **Default value**: AddedParts

- Defines the name of the Info*Engine Group that is to contain changed parts.
  
  **Property name**: com.ptc.windchill.esi.part.changePartName  
  **Default value**: ChangedParts

- Defines the name of the Info*Engine Group that is to contain unchanged parts.
  
  **Property name**: com.ptc.windchill.esi.part.unchangedPartName  
  **Default value**: AddedParts
• Defines the name of the Info*Engine Group that is to contain deleted parts.
  Property name: com.ptc.windchill.esi.part.deletedPartName
  Default value: DeletedParts

• Defines the name of the Info*Engine Group that is to contain added alternate links.
  Property name: com.ptc.windchill.esi.part.addAltlinkName
  Default value: AddedAlternateLinks

• Defines the name of the Info*Engine Group that is to contain deleted alternate links.
  Property name: com.ptc.windchill.esi.part.deletedAltlinkName
  Default value: DeletedAlternateLinks

• Defines the value of the <Class> element of an RPC that identifies a part (as opposed to a BOM).
  Property name: com.ptc.windchill.esi.part.partReleaseClass
  Default value: com.ptc.windchill.esi.Part

• Defines the life cycle state to pass to the configspec that is used to obtain the correct version of a part iteration for a given part master.
  Property name: com.ptc.windchill.esi.part.configSpec.state
  Default value: RELEASED

### 3.8 Publishing Documents

#### 3.8.1 Part and Document Relationships

Figure 48 shows the relationship between part iteration and a document (WTDocument). These documents are documents that this part references. Windchill ESI services navigates this relationship to return documents that are
associated to the part. The linking class is also published to show the relationship between a part and the document.

Figure 49  Part Describes Links

Figure 49, above, shows the second type of relationship between a part and a document (WTDocument). These documents are documents that describe the part. Windchill ESI services navigates this relationship to return documents that are associated to the part. The linking class (WTPartDescribeLink) is also published.

Figure 50  EPM Build History Links

Figure 50, above, shows the relationship between a part and an EPMDocument. Any CAD documents used to build this part are processed to identify the documents associated with this part. The linking class (EPMBuildHistory) is also published.
Figure 51  EPM Describes Links

Figure 51, above, shows the second type of relationship between a part and an EPMDocument that describes the part. If you have installed Windchill Engineering Factor, and CAD documents were used to build a part, the EPM describe links are navigated to obtain the documents associated with this part. The linking class (EPMDescribeLink) is also published.

3.8.2 Publish Document

Publish Document is the process by which Windchill ESI services creates and updates documents in a target enterprise system. Documents are published to the same distribution target, or targets, as the part that is associated with the documents. Standalone documents are not supported.

Windchill allows assigning EPMDocuments as well as WTDocument objects to parts. Windchill ESI services supports publishing of both types of documents. Windchill also allows documents to be assigned to other documents in a document structure. Windchill ESI services does not navigate the document structure to find related documents to publish.

3.8.3 Determining Which Documents to Publish

Information that shows the changes in Windchill between a part and document is published. Any information that has already been published to a distribution target is not republished.
The following examples show what is published to the target enterprise system:

![Diagram of Part with One Document and One Distribution Target](image)

**Figure 52  Part with One Document and One Distribution Target**

Figure 52, above, shows Part 100 iteration A, assigned to distribution target 1, and with a document attached (DOC 1). If this part iteration and document iteration have never been published to distribution target 1 before, then when the part is published the document and the document link is also published. The document appears in the sub-element `<AddedDocuments>` and the document links appear in the sub-element `<AddedDocmentLinks>`. See the *Windchill Enterprise Systems Integration Open Application Programming Interface Guide* for details on the XML structure of Windchill ESI RPC responses.
Figure 53  Part with Added Document

In Figure 53, Part 100 iteration A (with DOC1 attached) is already published to distribution target 1. A second document (DOC2) is added to Part 100 iteration B and then published. If the document (DOC2) wasn’t previously published to distribution target 1, the document (DOC2) and document link is published to distribution target 1. If the document (DOC2) already exists in the target enterprise system, the documents link for the document is published but not the document (DOC2). The document (DOC2) appears in the XML element `<AddedDocuments>` . The document link appears in the XML element `<AddedDocumentLinks>`.
In Figure 54, above, Part 100 iteration A (With DOC 1 attached) has already been published to distribution target 1. If Part 200 iteration A is being published with the same document as Part 100 and the iteration for the document is the same then only the document link is published in the XML element `<AddedDocumentLinks>` for Part 200. If the document iterations are different then the document is published as a change to the document in the XML element `<ChangedDocuments>` and the document link is published as an added document link in XML element `<AddedDocumentLinks>`.
In Figure 55, Part 100 iteration A (with DOC 1 and DOC 2 attached) already has been published to target 1. Document 2 was removed from Part 100 iteration B and published. The only XML element that is created is `<DeletedDocumentLinks>`, which describes the link between the part and the document. The Document is not removed because the document might be associated to another part.
3.8.4 Classes

Figure 56, below, shows the Windchill ESI Document Renderer classes.

**Document Response Renderers**

### 3.8.4.1 DocumentRenderer

The DocumentRenderer interface declares the public methods that must be implemented by all Windchill ESI document renderers. The interface may be extended by customizers. These are the methods of the interface:

3.8.4.1.1 `setData(currentPart : WTPart, previousPart : WTPart, targetOrgs : Collection, wcAdapter : String, authorization : String)`

Provides all necessary data to a DocumentRenderer.
3.8.4.2 DocumentRendererFactory

The DocumentRendererFactory is a stateless class whose responsibility is to obtain an object of the appropriate class. The public methods are available for use by customizers, but the class may not be extended nor modified by customizers.

3.8.4.2.1 newDocumentRenderer() : DocumentRenderer

Obtains an instance of the appropriate DocumentRenderer implementation class. Obtains the class from ESIProperties. The class must implement DocumentRenderer and it must have a default constructor.

3.8.4.3 ESIDocumentRenderer

This class is the default implementation of DocumentRenderer. The class may be extended. The class implements these methods:

3.8.4.3.1 setData(currentPart : WTPart, previousPart : WTPart, targetOrgs : Collection, wcAdapter : String, authorization : String)

Establishes the renderer’s state so that it can process documents.

3.8.4.3.2 validate()

Ensures that everything provided to setData() was complete and correct.

3.8.4.3.3 buildGroups()

Creates the Document output. Does the following:

1. If there is a previous iteration of the part, obtains the differences in document associations between the current part and the previous iteration of the part. Invokes ESIDocUtility.retrievePartDocDifferences() to obtain the differences.

If there is no previous part iteration, obtains the documents from the current part iteration by requesting them from the ESIDocUtility.retrievePartDocInformation().

The result of the ESIDocUtility methods is an array of collections of PartDocInfo objects.

2. Obtains all of the documents associated with the PartDocInfo objects.
3. Obtains all of the document links associated with the PartDocInfo objects.
4. Separates the documents and document links on the basis of whether they were added, removed, changed, or unchanged.
5. Invokes the Windchill adapter query-objects to obtain the data for the document and document link objects. calls adjustObjectID(), adjustTargets(), adjustDocumentElement() and adjustDocLinkElement() 
6. Passes the resultant Info*Engine element to the VdbBuilder that is maintained by the base class.
3.8.4.3.4 \textit{adjustDocumentElement(element : Element, group : String, document : Document, targets : Collection) : Element}

Allows subclasses to adjust the contents of an element before rendering it. This method is called by the renderer for each document object that is processed. Default behavior is to return the element argument.

3.8.4.3.5 \textit{adjustDocLinkElement(element : Element, group : String, documentLink : ObjectToObjectLink, targets : Collection) : Element}

Allows subclasses to adjust the contents of an element before rendering it. This method is called by the renderer for each document link object that is processed. Default behavior is to return the element argument.

The class provides to subclasses access to several attributes. The attributes are "read-only" in that there is no setter method. These are the available attributes:

3.8.4.3.6 \textit{currentPartIteration : WTPart}

The current iteration of the part associated with this document. No setter for this attribute - it must be set using setData().

3.8.4.3.7 \textit{previousPartIteration : WTPart}

The last release iteration of the part associated with this document. No setter for this attribute - it must be set using setData().

3.8.4.3.8 \textit{targets : Collection}

A collection ESITarget objects. No setter for this attribute - it must be set using setData().

3.8.4.3.9 \textit{wcAdapter : String}

The name of the Windchill Adapter where query-object tasks are to be executed. No setter for this attribute - it must be set using setData().

3.8.4.3.10 \textit{auth : String}

The authentication information to pass to the Windchill Adapter when executing a query-objects task. No setter for this attribute - it must be set using setData().

3.8.4.3.11 \textit{taskHelper : TaskHelper}

Reference to a TaskHelper. Do not access the attribute directly. Use the lazy-initialing getter.

3.8.4.3.12 \textit{docUtility : ESIDocUtility}

Reference to a ESIDocUtility. Do not access the attribute directly. Use the lazy-initialing getter.

3.8.4.3.13 \textit{transactionUtility : ESITransactionUtility}

Reference to a ESITransactionUtility. Do not access the attribute directly. Use the lazy-initialing getter.
3.8.4.4 ESIDocUtility

ESIDocUtility is the public interface to the com.ptc.windchill.esi.esidoc package. It provides methods that are used by the Windchill ESI document processing logic. The public methods are available for use by customizers, but the class may not be extended nor modified by customizers. These are the available methods of the class:

3.8.4.4.1 getESIDoc

getESIDoc(previousPart : WTPart, currentPart : WTPart, targetOrgs : Collection, wcAdapter : String, authorization : String) : IeCollection

Given the current and previous iteration of a part this method obtains the correct DocumentRenderer, sets its state and asks it to process the data. This method is provided for the benefit of renderers in other packages that may wish to render documents. For Windchill ESI services default implementations, only part renderers call this method.

3.8.4.4.2 getDocLinks

getDocLinks(documents : Collection, partDocInfo : Collection) : Collection

This method takes a collection of ESITargetAssociation objects as the documents argument. It takes a collection of PartDocInfo objects in the partDocInfo argument. It returns a collection of ESITargetAssociation objects.

If the document in the persistable object of the documents collection is also found by asking the PartDocInfo object from its document, the entry is selected to be returned. Each entry in the returned collection has a DocumentLink obtained from the PartDocInfo object as the persistable object. Each returned entry has the ESITarget from the matching ESITargetAssociation in the documents collection.

3.8.4.4.3 retrievePartDocInformation

retrievePartDocInformation(part : WTPart) : java.util.Collection

See the Document Attachment Differences section of this document.

3.8.4.4.4 retrievePartDocDifferences


See the Document Attachment Differences section of this document.

3.8.4.4.5 getAllDocuments

getAllDocuments(partDocInfo : Collection[]) : Collection

Returns all of the documents in an array of collections of PartDocInfo objects.

3.8.4.5 PartDocInfo

See the Document Attachment Differences section of this document.
Figure 57, shows the classes that get called to create the sub elements in the Response schema for the documents. The following is the sequence of events:

1. ESIPartRenderer calls the setData() method in ESIDocumentRenderer passing in a Collection array for the current parts, previous parts and ESITarget objects

2. ESIPartRenderer calls the ESIDocumentRenderer to validate the parameters

3. ESIPartRenderer calls the asVDB() method of ESIDocumentRenderer. The asVDB() method is an implementation of the Template Method Pattern. It always calls the buildGroups() method.

4. The ESIDocumentRenderer buildGroups() method calls the getESIDoc() method in ESIDocUtility for each part in the collection

5. If the current and previous iteration of the part is passed in as parameters getESIDoc() calls getDocLinkDifference()

   The method, getDocLinkDifference(), calls the WTPartService method, retrievePartDocDifferenceMethod(). (See the Document Attachment Differences section of this document for details.)

   a. If the first element in the getDocLinkDifference() response is not null, call getDocumentLink() in PartDocInfo to get the linking class. Add the value to the XML sub-element <AddedDocumentLinks>.

   b. If the second element in getDocLinkDifference() response is not null, call getDocumentLink() in PartDocInfo to get the linking class. Add the value to the XML sub-element

6. After the document links have been added to the correct XML sub-element in the documents are added. Iterate through the elements returned from the retrievePartDocDifferences() method calling the getDocument() method in PartDocInfo. For each document that is returned, check the publishing history to see if the document was previously published

7. If the document has never been published then add the document to the sub element <AddedDocuments>. If the document has changed since it was last published then add the document to the sub element <ChangedDocuments>

   Documents and document links that have been previously released or that have a pending release are not released. The release status is determined by finding whether or not an appropriate ReleaseActivity is associated to the document or document link.

   In the default implementation of Windchill ESI, the XML sub-elements <UnchangedDocuments> and <DeletedDocuments> in the response schema are not used.
3.8.5 Output Format

Windchill ESI document release output is controlled by the Windchill ESI VDB Builder. (See the VDB Builder section of this document for details.) Windchill ESI supports instance-based attributes and Windchill soft types by utilizing the services of the Windchill Adapter. Windchill ESI logic determines the appropriate objects to process. Once the objects are identified, a Windchill Adapter Query-Objects method is invoked to obtain the relevant data from the Windchill Adapter.

3.8.6 Customization Points

3.8.6.1 Extendable classes

If classes used by the Windchill ESI renderers described in the Publishing Documents section of this document are extended, the extensions must be defined in Windchill ESI properties. The following classes are designed to permit extensions by customizers:

- **DocumentRenderer**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.DocumentRenderer
  - **Default value**: com.ptc.windchill.esi.esidoc.ESIDocumentRenderer

- **WTDocument**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.WTDocument
  - **Default value**: wt.doc.WTDocument

- **WTPartDescribeLink**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.WTPartDescribeLink
  - **Default value**: wt.part.WTPartDescribeLink

- **WTPartReferenceLink**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.WTPartReferenceLink
  - **Default value**: wt.part.WTPartReferenceLink

- **EPMDocument**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.EPMDocument
  - **Default value**: wt.epm.EPMDocument

- **EPMDescribeLink**
  - **Property name**: com.ptc.windchill.esi.esidoc.class.EPMDescribeLink
  - **Default value**: wt.epm.structure.EPMDescribeLink
3.8.6.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI document renderer support. The options are:

- Define the name of the Info*Engine task to execute when obtaining document information from the Windchill Adapter.
  
  **Property name:** com.ptc.windchill.esi.esidoc.queryTask
  
  **Default value:** com/ptc/windchill/esi/QueryObject.xml
  
  **Note:** If any map file name begins with the characters, "$CODEBASE$", Windchill ESI services replaces "$CODEBASE$" with the system file directory name where Windchill ESI is located.

- Define the name of the map file to use when mapping documents.
  
  **Property name:** com.ptc.windchill.esi.esidoc.document.mapFile
  
  **Default value:** $CODEBASE$/esidoc/Document.map

- Define the name of the map file to use when mapping document links.
  
  **Property name:** com.ptc.windchill.esi.esidoc.documentLink.mapFile
  
  **Default value:** $CODEBASE$/esidoc/DocumentLink.map

3.9 Windchill ESI Windchill Service

Windchill ESI provides two major functions that support communication with the EAI software components. The first function is the Event Emitter. The function allows Windchill java classes and Windchill workflows to create notifications that objects are ready to publish. The second function is the Windchill ESI Windchill services. There are two standard Windchill services that implement methods that are useful to customizers and/or provide access to Windchill ESI via the Windchill Adapter Apply-Service webject. One service provides methods that can be invoked from remote clients such as Windchill applet GUIs. The other service provides methods that can only be invoked from within the Windchill Method Server. These methods support the RPCs and provide generally useful services to support customization.
3.9.1 Classes

These are the classes that support the event emission functionality:

![Diagram of Windchill ESI Event Emitter Objects]

**Figure 58  Windchill ESI Event Emitter Objects**

3.9.1.1 EventEmitter

Creates an Windchill ESI event and sends it to JMS via Info*engine. The class may be extended by customizers. These are the available methods of the class:

3.9.1.1.1 `releasePart(part : WTPart, action : String, waitingPart : WTPart)`

This method will create an event for Windchill assembly parts or for component parts. In the case of assembly parts, the value of the action argument defines whether the assembly is to be released as a part or as a BOM.

The method validates arguments and then:

1. creates a new transaction
2. creates a new event
3. passes the event to a new EventRenderer
4. creates the task defined in the property "com.ptc.windchill.esi.eventTask"
5. invokes the task

3.9.1.1.2 releaseWTChangeOrder2(ecn : WTChangeOrder2, action : String, waitingObject : WTOBJECT)
Validates arguments and then:
1. creates a new transaction
2. creates a new event
3. passes the event to a new EventRenderer
4. creates the task defined in the property "com.ptc.windchill.esi.eventTask"
5. invokes the task

3.9.1.1.3 releaseWTObject(object : WTOBJECT, action : String, waitingObject : WTOBJECT)
This method is provided for the convenience of customizers. It will emit a Windchill ESI event for objects other than WTPart and WTChangeORder2. The method validates arguments and then:
1. creates a new transaction
2. creates a new event
3. passes the event to a new EventRenderer
4. creates the task defined in the property "com.ptc.windchill.esi.eventTask"
5. invokes the task

3.9.1.1.4 addTaskParams()
Allows subclasses to add additional parameters to the task before it's executed.

3.9.1.1.5 addParam(name : String, value : Object)
Adds a task parameter to the event creation task before it's executed.

3.9.1.1.6 validatePart(part : WTPart)
Makes sure that the argument is in the correct state to be published by Windchill ESI. Obtains the name of the validator to use from Windchill ESI properties.

3.9.1.1.7 validateWTChangeOrder2(ecn : WTChangeOrder2)
Makes sure that the argument is in the correct state to be published by Windchill ESI. Obtains the name of the validator to use from Windchill ESI properties.

3.9.1.1.8 validateWTObject(object : WTOBJECT)
Makes sure that the argument is in the correct state to be published by Windchill ESI. Obtains the name of the validator to use from Windchill ESI properties.

3.9.1.1.9 validate(className : String, object : WTOBJECT)
Obtains an instance of ESIValidator and invokes the validate() method on that instance. The name of the class to instantiate is the first argument.
3.9.1.10 updateTask()

Provided as a convenience to extenders of the class. Called by releasePart(), releaseWTChangeOrder2() and releaseWTObject() immediately prior to invoking Info*Engine. Default implementation is to do nothing.

3.9.1.11 invokeTask(object : WTObject, action : String, waitingObject : WTObject)

Invokes an Info*Engine task to write the Windchill ESI Event into the appropriate JMS queue.

1. Obtains from Windchill ESI properties:
   • the name of the task
   • the name of the JMS queue
   • JMS queue authentication information
2. Obtains the current user from the Session Manager.
3. Obtains the SOAP message from the SoapRequestRenderer in the renderer attribute.
4. Passes the obtained information to the task as parameters
5. Invokes the task

3.9.1.2 ESIEvent

This class is a container for information necessary for Windchill ESI event notifications. It is intentionally not thread-safe. Customizers may extend the class if they so desire. However, the Windchill ESI Event Emitter is not designed to instantiate, or to work with, subclasses of ESIEvent. These are the public methods of ESIEvent:

3.9.1.2.1 getTargets() : Collection

Returns a collection of ESITargetAssociation objects related to the WTOObject in the object attribute. If the attribute is a WTChangeOrder2, or a WTChangeActivity2, get the distribution targets from the changeable objects associated to the change order or change activity.

3.9.1.2.2 validate()

Ensures that the ESIEvent object has the correct state and is ready to be published. Obtains an ESIEventValidator implementation from the ESIEventValidatorFactory. Calls the validate() method on the ESIEventValidator. Then calls the private methods objectValidate() and actionValidate().

3.9.1.2.3 getObjectID() : String

Returns the obid from the contained object.

3.9.1.2.4 getWaitingObjectID() : String

If the waiter attribute is not null, returns the id of that object. Otherwise returns a null string.

3.9.1.2.5 getTransactionNumber() : String

Returns the number from the ESITransaction in the txn attribute. If the attribute is null, creates a new ESITransaction and saves it in the database before returning the result.
3.9.1.3 ESIValidator

Defines the methods that Windchill ESI Event object validators must implement. These are the methods of the interface

3.9.1.3.1 validate(object : WTObject)

Makes sure the object is internally consistent. Implementers of this interface are responsible to implement the validation business rules.

3.9.1.4 ESIValidatorImpl

This is a convenience class that contains methods common to all validators. The class is abstract and must be extended. These are the methods of the class:
3.9.1.4.1 validateInstance(object : Object, name : String, cls : Class)
Makes sure that the argument is not null and that is an instance of the class specified.

3.9.1.5 ESIWTOBJECTValidator
Default implementation for validating WTObject objects prior to publishing them. The class may be extended. These are the methods of the class:

3.9.1.5.1 validate(object : WTObject)
Default behavior is to make sure that the object is not null.

3.9.1.6 ESICHANGEORDER2Validator
Validates a WTChangeOrder2 for Windchill ESI publishing. The class may be extended. These are the methods of the class:

3.9.1.6.1 validate(object : WTObject)
Makes sure that the WTChangeOrder2 object has changeable items and that at least one of the items is a part. Optionally checks to see that at least one part has organizations associated with it.

3.9.1.7 ESIPARTValidator
Default implementation for validating WTPart objects prior to publishing them. The class may be extended. These are the methods of the class:

3.9.1.7.1 validate(object : WTObject)
Makes sure that the object is a WTPart and not null. Optionally ensure that the part has distribution targets associated to it.
3.9.1.8 SoapRequestRenderer

This interface declares the methods that Windchill ESI SOAP request renderers must implement.

3.9.1.8.1 asSoapRequest() : String

Every implementation of this interface must be able to render a SOAP formatted XML request string.

3.9.1.8.2 setEvent(event : Object)

Demand that all renderers are stateful and provide a setter for the event object.

3.9.1.9 ESISoapRequestRenderer

Abstract class containing the implementation of the SoapRequestRenderer interface. This class must be extended. These are the methods of the class:

3.9.1.9.1 asSoapRequest() : String

This is a template method that forces subclasses to validate() and then executes their buildArguments() method. It stores the results in the arguments attribute.
3.9.1.9.2 validate()
Abstract method forcing subclasses to ensure that their state is correct before attempting to build the request.

3.9.1.9.3 buildArguments() : String
Abstract method forcing subclasses to create the XML fragment that is enclosed in a SOAP envelope.

3.9.1.10 ESIEventRenderer
Default implementation for the SoapRequestRenderer interface. Converts an ESIEvent object to a String representation of an XML document fragment. This class may not be extended. These are the available methods of the class:

3.9.1.10.1 validate()
Makes sure that the event is not null and that its validate() method is invoked.

3.9.1.10.2 buildArguments() : String
Builds an XML fragment with the following:
- ObjectID
- TransactionID
- Class
- WaitingEventID
- One or more ESITarget XML elements.

3.9.1.11 ESIEventException
Carries information about exceptions that occur with Windchill ESI event-related classes. This class may be extended. These are the methods of the class:

3.9.1.11.1 ESIEventException()
Default constructor.

3.9.1.11.2 ESIEventException(msg : String)
Creates a new object with the argument as a message.

3.9.1.11.3 ESIEventException(err : Exception)
Creates a new object based on the argument.

3.9.1.12 ESIEventValidatorFactory
Provides instances of the appropriate implementation of ESIEventValidator. The factory obtains the name of the implementation from Windchill ESI properties.

3.9.1.12.1 newESIEventValidator() : ESIEventValidator
This static method creates a new instance of an implementation of ESIEventValidator. It obtains the name of the implementation class from Windchill ESI properties.
3.9.1.13 **ESIEventValidator**

This interface defines the methods that must be available on Windchill ESI event validator implementations. These are the methods:

3.9.1.13.1 `validate(event : ESIEvent)`

Makes sure that the ESIEvent has the necessary state. Throw some form of WTException if the event is not ready to process.

3.9.1.14 **ESIStandardEventValidator**

This is the default implementation of ESIEventValidator. The class may not be extended.

3.9.1.14.1 `validate(event : ESIEvent)`

Makes sure that the object is a ESIEvent and not null. Optionally, ensure that no transaction is f against the primary business object.

These are the classes that support the ECN auto-generation feature:

![ECN Auto-generation Support Classes](image)

**Figure 61** ECN Auto-generation Support Classes

3.9.1.15 **ESIECNFactory**

This interface defines the methods that ECN factory classes must implement. ECN factories auto-generate a Windchill change order when change processing is not being used in the Windchill environment, but change processing is required by the target enterprise system.
3.9.1.15.1 createChangeOrder(item : Changeable2) : WTChangeOrder2
Wraps the given changeable in a change order.

3.9.1.16 ESIStandardECNFactory
This class is the default implementation of ESIECNFactory. It auto-generates a Windchill change order when change processing is not being used in the Windchill environment, but change processing is required by the target enterprise system. The class is intended to be extended by customizers. These are the available methods of the class:

3.9.1.16.1 createChangeOrder(item : Changeable2) : WTChangeOrder2
Creates change request, change order, change activity, and associates the argument to the change activity.

1. Obtains an instance of WTChangeRequest2 from an instance of ESIEChangeRequestFactory.
2. Obtains an instance of WTChangeOrder2 from an instance of ESIEChangeOrderFactory.
3. Obtains an instance of WTChangeActivity2 from an instance of ESIEChangeActivityFactory.
4. Invokes associateChangeableToChangeActivity()

3.9.1.16.2 associateChangeableToChangeActivity(changeActivity : ChangeActivity2, item : Changeable2)
Persist the association between given change activity and given changeable.

3.9.1.16.3 getChangeRequestFactory() : ESIEChangeRequestFactory
Returns an object capable of creating change requests.

3.9.1.16.4 getChangeOrderFactory() : ESIEChangeOrderFactory
Returns an object capable of creating change orders.

3.9.1.16.5 getChangeActivityFactory() : ESIEChangeActivityFactory
Returns an object capable of creating change activities.

3.9.1.17 ESIEChangeRequestFactory
This class is responsible for creating and saving a change request. This class is may be extended by customizers. These are the available methods of the class:

3.9.1.17.1 create() : WTChangeRequest2
Creates and saves a new change request.

3.9.1.17.2 getName() : String
Returns the name to be given to the new change request. The value is localized from EsiResource.ESI_CHANGE_REQUEST_NAME.

3.9.1.17.3 getDescription() : String
Returns the description to be given to the new change request. The value is localized from EsiResource.ESI_CHANGE_REQUEST_DESC.
3.9.17.4 getPriority() : RequestPriority
Returns the priority to be assigned to the new change request. Value is always RequestPriority.LOW.

3.9.17.5 getCategory() : Category
Returns the category to be assigned to the new change request. Value is always Category.OTHER.

3.9.17.6 getComplexity() : Complexity
Returns the complexity to be assigned to the new change request. Value is always Complexity.SIMPLE.

3.9.18 ESIChangeOrderFactory
This class is responsible for creating and saving a change order. This class may be extended by customizers. These are the available methods of the class:

3.9.18.1 create(changeRequest : WTChangeRequest2) : WTChangeOrder2
Creates and save a new change order that is associated with the argument.

3.9.18.2 getName() : String
Returns the name to be given to the new change order. The value is localized from EsiResource.ESI_CHANGE_ORDER_NAME.

3.9.18.3 getDescription() : String

3.9.19 ESIChangeActivityFactory
This class is responsible for creating and saving a change activity. This class may be extended by customizers. These are the available methods of the class:

3.9.19.1 create(changeOrder : WTChangeOrder2) : WTChangeActivity2
Creates and save a new change activity associated with the argument.

3.9.19.2 getName() : String
Returns the name to be given to the new change activity. The value is localized from EsiResource.ESI_CHANGE_ACTIVITY_NAME.

3.9.19.3 getDescription() : String
Returns the description to be given to the new change activity. The value is localized from EsiResource.ESI_CHANGE_ACTIVITY_DESC.
These are the classes that support the Windchill ESI Windchill workflows:

**ESI Windchill Event**

**Figure 62  Windchill ESI Workflow Support**

### 3.9.1.20 ESIResultEvent

Windchill ESI workflows may suspend until external processes are complete. When they do, the ESIResultEvent is the trigger that tells the workflow to resume. Objects of this class convey to a workflow whether or not a Windchill ESI external request succeeded. The class may be extended by customizers; however, Windchill ESI workflows are not designed to look for a subclass of ESIResultEvent. Customizers are required to modify the Windchill ESI workflow templates if they want to extend the class ESIResultEvent. The following attributes have public getters and setters:

#### 3.9.1.20.1 successful : boolean

Defines whether or not the external process succeeded.

#### 3.9.1.20.2 errorMessage : String = ""

An optional message to provide to the workflow.

#### 3.9.1.20.3 targetUfid : String

The UFID to the ESITarget associated with this result.

#### 3.9.1.20.4 txUfid : String

The UFID to the ESITransaction associated with this result.

These are the public methods of the class:

#### 3.9.1.20.5 ESIResultEvent(eventTarget : Persistable)

Only constructor for this class. Ensures that the object is always in the correct state.
3.9.1.20.6 `getTarget()` : `Persistable`

Returns the event target associated with this event.

These are the classes that provide the publicly available capabilities of Windchill ESI services:

```
<<Final>>
ESIService

<<RemoteInterface>>
ESIService

<<Final>>
ESIHelper

<<Final>>
ESISvrHelper

ESISvrService

PersistenceHelper

StandardManager

ESIService

<<Final>>
StandardESIService
```

Figure 63  Windchill ESI Windchill Services

### 3.9.1.21 `ESIService`

This interface defines the Windchill ESI methods that are available to remote clients. This interface has a UML stereotype of RemoteInterface. Therefore, the Windchill code generator creates a class, ESIServiceFwd, when this interface is generated. ESIServiceFwd provides remote access to the methods defined in the ESIService interface. This interface is not intended to be extended. These are the methods of the interface:

#### 3.9.1.21.1 `assignTarget(object : Persistable, target : ESITarget)`

Assigns an object to an ESITarget for distribution target system destination determination. Delegates all work to an instance of ESITargetUtility.

#### 3.9.1.21.2 `assignTarget(object : Persistable, targets : Group)`

Assigns an object to an Info*Engine group of ESITarget objects for distribution target system destination determination. Delegates all work to an instance of ESITargetUtility.
3.9.1.21.3 createTarget(name : String, description : String, system : String) : ESITarget
Creates an ESITarget from the arguments and saves the result in the database. Delegates all work to an instance of ESITargetUtility.

3.9.1.21.4 deleteTarget(tgt : ESITarget)
Deletes an ESITarget from the database. Delegates all work to an instance of ESITargetUtility.

3.9.1.21.5 removeTargetAssignment(object : Persistable, target : ESITarget)
Removes the relationship between an ESITarget and a persistable object. Delegates the deletion to an instance of ESITargetUtility.

3.9.1.21.6 removeTargetAssignment(ufid : String)
Deletes from the database the ESITargetAssignmentLink that is identified by the argument. Delegates all work to an instance of ESITransactionUtility.

3.9.1.21.7 getTransaction(id : String) : ESITransaction
Returns the ESITransaction whose idNumber attribute matches the argument. Throw an ESIException if no matching transaction exists. Delegates all work to an instance of ESITransactionUtility.

3.9.1.21.8 getTarget(system : String) : ESITarget
Returns the distribution target whose system attribute matches the argument. Throw an ESIException if no matching ESITarget exists. Delegates all work to an instance of ESITargetUtility.

3.9.1.21.9 createTransaction(object : WTObject)
Creates and saves in the database a new ESITransaction that is linked to a WTOrganization. Delegates all work to an instance of ESITransactionUtility.

3.9.1.21.10 connectTargetToOrganization(tgt : ESITarget, org : WTOrganization)
Connects an ESITarget to a WTOrganization. Delegates all work to an instance of ESITargetUtility. The ESITargetUtility method is not implemented.

3.9.1.21.11 disconnectTargetFromOrganization(tgt : ESITarget, org : WTOrganization)
Deletes the ESITargetOrganizationLink that connects an ESITarget object to a WTOrganization object. If either the ESITarget object or the WTOrganization are not persistent, the method throws a WTException. Delegates all work to an instance of ESITargetUtility. The ESITargetUtility method is not implemented.

3.9.1.21.12 findESITargets(anObject : Persistable) : Collection
Returns a collection of ESITarget objects for the given object. If the object is a WTChangeOrder2, return the distribution targets associated with the changeables on the change order.

3.9.1.21.13 isWFLaunched(anObject : LifeCycleManaged) : boolean
Checks whether there is a workflow process with the status of OPEN_RUNNING for the given object.
3.9.1.22 ESIHelper

Provides access to the ESIService implementation and provides several generally useful utility methods. The class in not intended to be extended.
These are the public methods of the class:

3.9.1.22.1 **<<Static>>** getTxStatusURL(txUfid : String) : String

Returns a URL HREF for the Windchill ESI transaction log GUI associated with the given transaction UFID.

3.9.1.22.2 **<<Static>>** hasESITargets(anObject : Persistable) : boolean

Returns true if the given object has distribution target assignments.

3.9.1.22.3 **<<Static>>** isOnWfProcess(anObject : LifeCycleManaged) : boolean

Returns true if Windchill ESI workflow is OPEN_RUNNING and, therefore, processing the given object.

3.9.1.22.4 **<<Static>>** isReleaseNeeded(persistable : Persistable) : boolean

Returns true if the given object has an ESITarget assigned to which the object has not yet been released.

3.9.1.22.5 **<<Static>>** isTxActive(obj : Persistable) : boolean

Returns true if the transaction status is PENDING or PROCESSING.

The class has a public static attribute.

3.9.1.22.6 Service: ESIService

This final static attribute is initialized to a new instance of ESIServiceFwd. The class, ESIServiceFwd, is generated by the Windchill code generator and provides remote access to the methods defined in the ESIService interface.

3.9.1.23 ESISvrHelper

Provides access to the ESISvrService implementation. The class in not intended to be extended. The class has a public static attribute.

3.9.1.23.1 Service: ESIService

This final static attribute is initialized to a new instance of ESISvrService implementation.

3.9.1.24 ESISvrService

Defines the non-client methods that are available on the Windchill ESI service. This interface is not intended to be extended. These are the methods of the interface:


PostResult RPC processor. Delegates all work to an instance of ESITransactionUtility.

3.9.1.24.2 latestExport(object : Persistable, target : ESITarget) : ESITargetAssociation

Finds the highest revision of an object that has been published to a distribution target.

3.9.1.24.3 postEvent(objectID : String, transaction : String, successful : boolean, message : String)

PostEvent RPC processor. Delegates all work to an instance of ESITransactionUtility.
3.9.1.24.4 getPart(objectID : String, alternates : boolean, documents : boolean, auth : String, wcAdapter : String, transactionID : String) : IeCollection

GetPart RPC processor. Defers all work to an instance of ESIPartUtility.


This method is the getBOM RPC processor. Delegates all work to an instance of BOMUtility.


This method is the getECN RPC processor. Obtains a VdbBuilder from the VdbBuilder factory.

- Obtains a ChangeOrder2Renderer from the ChangeOrderRenderer factory.
- Provides the necessary data to the renderer from the arguments to this method.
- Invokes the render() method on the renderer.
- Returns the result of VdbBuilder asVDB().

3.9.1.24.7 adjustTargetAssignments(ufids : String, targetInfo : Group)

Given a concatenated set of UFIDs (for WTPart objects) and a Collection of ESITarget UFIDs, makes (or removes) any required assignments.

- The ufids argument contains a set of UFIDs which define the parts to operate on.
- The targetInfo argument is an Group. Each element in the group has Atts that:
  - define the ufid of an ESITarget
  - define the operation to perform on the parts. If the operation is "add", then all parts that do not have the distribution target assigned are given the assignment. If the operation is "remove", then all existing assignments for that distribution target to any of the existing parts are removed.

Delegates all action to an instance of ESITargetUtility.

3.9.1.24.8 getTargetAssociations(ufids : String, groupName : String) : Group

Given a string of concatenated UFIDs, return the ESITarget objects that are associated with those objects. If the ESITarget is assigned to all of the objects identified by a UFID, indicate that it is a full assignment; otherwise, indicate that the value is partial. Delegates all work to an instance of ESITargetUtility.
3.9.1.24.9 `getTargetsWithHistory(objects : Collection) : Group`

Returns an Info*Engine Group of mapped ESITargetHistory objects. The ESITargetHistory objects contain the following:

- The target attribute is one of the available distribution targets.
- The persistableObject is one of the objects in the argument to this method.
- If the persistable object has been published to the distribution target, the releaseAttribute contains the appropriate ReleaseActivity.

The result is serialized by the Windchill ESI mapping logic to a group of Info*Engine Element objects. All of the work is done by the ESITransactionUtility which is called by this method.

3.9.1.24.10 `getTargetsWithHistory(ufid : String) : Group`

Returns an Info*Engine Group of mapped ESITargetHistory objects. The ESITargetHistory objects contain the following:

- The target attribute is one of the available distribution targets.
- The persistableObject is one of the objects in the argument to this method.
- If the persistable object has been published to the distribution target, the releaseAttribute contains the appropriate ReleaseActivity.

The result is serialized by the Windchill ESI mapping logic to a group of Info*Engine Element objects. All of the work is done by the ESITransactionUtility which is called by this method.

3.9.1.24.11 `getTargetsWithHistory(object : Persistable) : Collection`

Returns Collection of mapped ESITargetHistory objects. The ESITargetHistory objects contain the following:

- The target attribute of the is one of the available distribution targets.
- The persistableObject is one of the objects in the argument to this method.
- If the persistable object has been published to the distribution target, the releaseAttribute contains the appropriate ReleaseActivity.

The result is serialized by the Windchill ESI mapping logic to a group of Info*Engine Element objects. All of the work is done by the ESITransactionUtility which is called by this method.

3.9.1.24.12 `getHistory(ufid : String, groupOut : String) : Group`

Returns a Group of mapped ReleaseActivity objects associated with the object identified by the argument. This is the functional equivalent of `getTargetsWithHistory()` for objects that do not have ESITargetAssignmentLinks, but that do have ReleaseActivity associations. The method does the following:

- Obtains an instance of ESIObjectHistoryRenderer and an instance of ESIObjectHistoryBuilder from the ESIObjectHistoryRendererFactory
• Invokes the render() method on the renderer
• Returns the result of builder.getGroup()

3.9.1.24.13 getActivities(ufid : String) : ReleaseActivity[]
Given the UFID of an ESITransaction, returns an array of all of the ReleaseActivity objects associated with that ESITransaction. Delegates implementation to ESITransactionUtility.

3.9.1.24.14 hideTransaction(ufid : String)
Hides an ESITransaction from the user interface.

3.9.1.24.15 recordExport(objectID : String, targetSystem : String, releaseClass : String)
RecordExport RPC processor. This method identifies an object as published to an ESITarget. It does the following:
1. Obtains the persistable object from an instance of TaskHelper using the objectID argument.
2. Obtains the distribution target from an instance of ESITargetUtility using the targetSystem argument.
3. Passes the distribution target, the persistable object, and the releaseClass argument to an instance of ESITransactionUtility.

3.9.1.24.16 getReleaseActivities(ufid : String) : Group
Returns an Info*Engine Group of mapped ESI ReleaseActivity objects given the UFID of an ESITransaction object.

3.9.1.24.17 getTransactions(elements : Group) : Group
Returns an Info*Engine Group of ESITransaction objects that are associated with the objects in the argument. The argument is an Info*Engine Group containing Elements that have been created by (or resemble the output of) a Query-Objects webject. Defers all work to an instance of ESITransactionUtility.

3.9.1.24.18 getReleaseActivitiesMessage(ufid : String) : Group
Return an Info*Engine Group of mapped ReleaseActivityMessage object given the UFID of a ReleaseActivity object.

3.9.1.24.19 getTransactionLogMessage(ufid : String) : Group
Returns an Info*Engine Group of mapped ESITransactionMessage object given the UFID of an ESITransaction object.

3.9.1.25 StandardESIService
StandardESIService is a concrete implementation of ESIService and ESISvrService. The class is not intended to be extended. These are the additional modeled methods of the class:

3.9.1.25.1 registerEvents(ms : ManagerService)
This method has no implementation.
3.9.1.25.2 dispatchVetoableEvent(eventType : String, obj : Persistable)

This method has no implementation.

3.9.1.25.3 performStartupProcess()

Prepares the Windchill Service to begin operations. This is a protected method that is called by the base class, StandardManager, start up processing. The method registers event listeners that:

- Validate deletion of ESITransaction objects
- Validate deletion of ESITarget objects
- Validate deletion of WTPart objects
- Copy distribution target assignments when new iterations, versions, and view-versions of parts are created
- Optionally, depending on Windchill ESI property settings, launch an Windchill ESI workflow when a lifecycle state change occurs.

3.9.2 Info*Engine Tasks

Windchill ESI services provides several Info*Engine tasks that provide input and output. These are the tasks that support Windchill ESI services:

3.9.2.1 EmitSoapEvent

This task writes a block of text to a JMS queue, setting the JMS message properties required for an Windchill ESI Event. See the *Windchill Enterprise Systems Integration Open Application Programming Interface Guide* for details. These are the task parameters:

3.9.2.1.1 User

This mandatory parameter is used to set the value of the JMS message property, com_infoengine_user.

3.9.2.1.2 Class

This mandatory parameter is used to set the value of the JMS message property, com_infoengine_SOAP_class.

3.9.2.1.3 message

This mandatory parameter is used to set the value of the JMS text message.

3.9.2.1.4 priority

This optional parameter is used to set the value of the JMS message priority.

3.9.2.1.5 dbuser

This optional parameter is used to identify the user who is attempting to access a secured JMS queue.

3.9.2.1.6 passwd

This optional parameter is used to supply the password for access a secured JMS queue.
3.9.2.1.7 lifetime
This optional parameter is used to declare how long the message should remain on the queue before being auto-deleted if no queue listener reads it.

3.9.2.1.8 queue
This optional parameter is used to specify the name of the queue where the message is to be written. If no value is specified, the value is obtained from Windchill ESI properties.

3.9.2.1.9 locale
This optional parameter is used to specify a Windchill supported locale. This locale should be specified by EAIcomponents for all Windchill ESI RPC invocations related to this event.

3.9.2.2 SubscribeQueues
This task is used to register Windchill ESI services' interest in messages that appear on JMS queues. The task has no parameters. The task does the following:

- Calls TaskHelper.waitForMethodServer() to make sure that Windchill is properly initialized before accepting input from the JMS queues.
- Calls GetSubscribeQueueNames to obtain the list of queues to subscribe.
- For each entry in the result, the task calls SubscribeQueue, passing the queue name, the password, and the queue user name.
- The task returns a Group, each element of which contains the result from a single call to SubscribeQueue.

3.9.2.3 GetSubscribeQueueNames
The task returns a group of elements, each with the name, connection user, and password of a JMS queue. The task has one optional parameter:

3.9.2.3.1 groupOut
This optional parameter defines the name of the Info*Engine group to create. If not specified, the default value is, "SubscribeQueueNames".

3.9.2.4 SubscribeQueue
This task invokes the Info*Engine Subscribe-Queue webject. The webject creates a Group, "subscription-results", whose group message is the result of the request. The task returns this group to the caller. The task has the following parameters:

3.9.2.4.1 jmsQueue
This mandatory parameter identifies the JMS queue name.

3.9.2.4.2 jmsQueueUser
This optional parameter identifies the name to use when connecting to a secure queue.

3.9.2.4.3 jmsQueuePwd
This optional parameter identifies the password to use when connecting to a secure queue.
3.9.2.4  **task**
This optional parameter identifies the name of an Info*Engine task to invoke when a qualifying message is received on the JMS queue.

3.9.2.4.5  **where**
This optional parameter identifies filtering to apply when determining whether to process messages that arrive on the queue. See the Info*Engine Users Guide for details on the parameters to the Subscribe-Queue webject.

3.9.2.5  **UnsubscribeQueue**
This task invokes the Unsubscribe-Queue webject to stop listening for messages on a JMS queue. The webject creates an empty Group, "unsubscription-results", that has a message. The message is written to the Windchill log if Windchill ESI verbose property is set to "true". These are the task parameters

3.9.2.5.1  **jmsQueue**
This parameter identifies the name of the queue to stop listening.

3.9.2.5.2  **where**
This optional parameter identifies filtering to apply when determining whether to process messages that arrive on the queue. See the *Info*Engine Users Guide for details on the parameters to the Unsubscribe-Queue webject.

3.9.3  **Customization Points**

3.9.3.1  **Extendable classes**
Extendable Windchill ESI classes described in the Windchill ESI Windchill Service section of this document must be defined in Windchill ESI properties if they are extended. The following classes are designed to permit extensions by customizers:

8.  ESIEventRenderer
    
    **Property name**: com.ptc.windchill.esi.event.renderer
    **Default value**: com.ptc.windchill.esi.evt.ESIEventRenderer

8.  ESIPartValidator
    
    **Property name**: com.ptc.windchill.esi.event.partValidator
    **Default value**: com.ptc.windchill.esi.evt.ESIPartValidator

8.  ESIStandardEventValidator
    
    **Property name**: com.ptc.windchill.esi.event.eventValidator
    **Default value**: com.ptc.windchill.esi.evt.ESIStandardEventValidator

8.  ESIChangeOrder2Validator
    
    **Property name**: com.ptc.windchill.esi.event.ecnValidator
Default value: com.ptc.windchill.esi.evt.ESIChangeOrder2Validator

- ESIWTObjectValidator
  Property name: com.ptc.windchill.esi.event.wtobjectValidator
  Default value: com.ptc.windchill.esi.evt.ESIWTObjectValidator

- ESIECNFactory
  Property name: com.ptc.windchill.esi.ECN.ESIECNFactory
  Default value: com.ptc.windchill.esi.ecn.ESIStandardECNFactory

- ESIChangeRequestFactory
  Property name: com.ptc.windchill.esi.ECN.ESIChangeRequestFactory
  Default value: com.ptc.windchill.esi.ecn.ESIChangeRequestFactory

- ESIChangeOrderFactory
  Property name: com.ptc.windchill.esi.ECN.ESIChangeOrderFactory
  Default value: com.ptc.windchill.esi.ecn.ESIChangeOrderFactory

- ESIChangeActivityFactory
  Property name: com.ptc.windchill.esi.ECN.ESIChangeActivityFactory
  Default value: com.ptc.windchill.esi.ecn.ESIChangeActivityFactory

### 3.9.3.2 Configurable Options

Windchill ESI properties control several configurable options for Windchill ESI services. The options are:

- Define whether, or not, Windchill ESI services modules should write debug information to the Windchill log. This property only has effect if Windchill debug logging is set to true.

  Property name: com.ptc.windchill.esi.verbose
  Default value: true

- Define the name of the JMS queue where events are to be written.

  Property name: com.ptc.windchill.esi.event.queueName
  Default value: com.ptc.windchill.esi.Event

- Define the user name to supply when connecting to secure JMS queues to write Windchill ESI events.

  Property name: com.ptc.windchill.esi.event.queueUser
  Default value: WCESI

- Define the password to supply when connecting to secure JMS queues to write Windchill ESI events.
**Property name:** com.ptc.windchill.esi.event.queuePassword  
**Default value:** WCESI  
- Define the name of the task to execute when writing events to the JMS queue.

**Property name:** com.ptc.windchill.esi.event.queueTask  
**Default value:** com/ptc/windchill/esi/EmitSoapEvent.xml
- Define the amount of time (in milliseconds) the task helper should wait after a failed test to see if the Windchill Method Server is ready to process Windchill ESI RPCs.

**Property name:** com.ptc.windchill.esi.subscribe.wait  
**Default value:** 10000
- Define the number of times the task helper should wait for the interval (specified by com.ptc.windchill.esi.subscribe.wait) after a failed test to see if the Windchill Method Server is ready to process Windchill ESI RPCs.

**Property name:** com.ptc.windchill.esi.subscribe.tries  
**Default value:** 12
- Define whether the Event Emitter should throw an exception if the object being published has no associated distribution targets.

**Property name:** com.ptc.windchill.esi.event.enforceOrgs  
**Default value:** true
- Define the Event Emitter should throw an exception if the object being published already has a pending transaction (an event has already been emitted for object, but event has not been processed).

**Property name:** com.ptc.windchill.esi.event.enforceNoPendingTxn  
**Default value:** true
- Define the name of the JMS queue where RPC requests are to be accepted.

**Property name:** com.ptc.windchill.esi.request.queueName  
**Default value:** com.ptc.windchill.esi.DataRequest
- Define the user name to supply when connecting to secure JMS queues to accept RPC requests.

**Property name:** com.ptc.windchill.esi.request.queueUser  
**Default value:** WCESI
- Define the password to supply when connecting to secure JMS queues to accept RPC requests.

**Property name:** com.ptc.windchill.esi.request.queuePassword
Default value: WCESI

- Define the name of the JMS queue where Result RPC request are to be accepted.
  
  **Property name:** com.ptc.windchill.esi.result.queueName  
  **Default value:** com.ptc.windchill.esi.Result

- Define the user name to supply when connecting to secure JMS queues to accept Result RPC requests.
  
  **Property name:** com.ptc.windchill.esi.result.queueUser  
  **Default value:** WCESI

- Define the password to supply when connecting to secure JMS queues to accept Result RPC requests.
  
  **Property name:** com.ptc.windchill.esi.result.queuePassword  
  **Default value:** WCESI

- Define whether the Event Emitter should create an ECN and emit an event identifying the newly-created ECN whenever a component part or an assembly part (BOM) is published.
  
  **Property name:** com.ptc.windchill.esi.ECN.autoGenerate  
  **Default value:** false

- Some Windchill ESI java classes create collections that are intended to hold the results calculations of other collections. When the calculation is expected to create additional entries, the new collection is given an initial size using the following formula: new collection size = the original collection size * the value of this property. This property is provided as performance tuning option.
  
  **Property name:** com.ptc.windchill.esi.collectionSizeMultiplier  
  **Default value:** 4

- Define whether Windchill ESI renderers should throw an exception if they receive an RPC request for an object that has not changed since it was last published.
  
  **Property name:** com.ptc.windchill.esi.enforceChanges  
  **Default value:** true

- Define whether Windchill ESI renderers should determine that an object that has changed since it was last published if there is a new iteration, but not a new version.
  
  **Property name:** com.ptc.windchill.esi.enableCheckIterations  
  **Default value:** false
- Define URI used to invoke the DCA GUI. Normally this property is not changed.
  
  **Property name:** com.ptc.windchill.esi.gw  
  **Default value:** wtcore/jsp/com/ptc/core/ca/web/gw/gw.jsp

- Define how to build the transaction ID argument when invoking DCA URI. Used in conjunction with the com.ptc.windchill.esi.gw to display a transaction in the DCA GUI. Normally this property is not changed.
  
  **Property name:** com.ptc.windchill.esi.tx.alias  
  **Default value:** com.ptc.windchill.esi:single.transaction

- Define whether a release workflow is to be auto-launched when a specified object lifecycle state is reached. The state at which the auto-launch occurs is controlled by the property, com.ptc.windchill.esi.ECNCentric.state.
  
  **Property name:** com.ptc.windchill.esi.wf.autoLaunch  
  **Default value:** true

- Define the Lifecycle State at which the release workflow is auto launched. This property is used when the com.ptc.windchill.esi.wf.autoLaunch is set to TRUE.
  
  **Property name:** com.ptc.windchill.esi.wf.ECNCentric.state  
  **Default value:** COMPLETED

- Define a value used to populate the Description attribute on the initiate workflow window.
  
  **Property name:** com.ptc.windchill.esi.wf.Description  
  **Default value:** *no default value specified for this property*

- Define what release model Windchill ESI services should use. If the value is true, use ECN-centric if the value of this property is false, use the part-centric release model. The ECN-centric model is used when the target enterprise systems demand engineering change notices in order to maintain product structure information. If the target enterprise systems permit product structure updates without ECNs, the part-centric model may be used.
  
  **Property name:** com.ptc.windchill.esi.wf.ECNCentric  
  **Default value:** true

- Define the value used to populate the team name on ECN-centric release workflow processes. This value should match the team name that owns the ECN centric workflow template.
  
  **Property name:** com.ptc.windchill.esi.wf.ECNCentric.team  
  **Default value:** Change Team
• Define the value used to populate the team name on part-centric release workflow processes. This value should match the team name that owns the part centric workflow template.

**Property name:** com.ptc.windchill.esi.wf.PartCentric.team  
**Default value:** ESI Team

• Define a value that is combined with an object ID and used to populate the workflow process name attribute.

**Property name:** com.ptc.windchill.esi.wf.ProcessPrefix  
**Default value:** ESIRelease

• Define the workflow template used for the ECN centric release process.

**Property name:** com.ptc.windchill.esi.wf.ReleaseECNTemplate  
**Default value:** Release ECN to Manufacturing

• Define the workflow template used for the part centric release process.

**Property name:** com.ptc.windchill.esi.wf.ReleasePartTemplate  
**Default value:** Release Part to Manufacturing

### 3.10 Effectivity Handling

#### 3.10.1 Windchill Effectivity Service

Windchill effectivity support has been extended to provide these additional API methods:

• The Windchill effectivity service provides a method, `getEffectivities(version : EffManagedVersion, context : EffContext, effType : Class, includeHistory : boolean) : Eff[]`. The method returns a sorted array of Eff objects of the concrete type represented by the effType parameter, and related to the version and context parameters. The results are sorted in the order of creation of the Eff objects. Object reference attributes of the results are not expanded. This is an internal API to support Windchill ESI product functionality.

• The Windchill effectivity helper provides an `isCurrent(eff : Eff) : boolean` method. The method returns true if and only if the argument is current (that is, it has not been factually deleted and thus relegated to effectivity history).

• The Windchill effectivity helper provides a method, `equals(e1 : Eff, e2 : Eff) : boolean`. The method returns true if and only if the objects are equal (that is, their respective range values are equal).
3.10.2 Windchill ESI Services

Windchill ESI services creates a persisted record of the effectivity that was in force when a Windchill effectivity-managed object is released. When Windchill ESI is determining whether an effectivity-managed object has already been published, it uses the following algorithm:

1. Is this a version-managed object and has the revision changed? If so, the object is to be published, skip the rest of the tests.
2. Is there an effectivity associated with the object being evaluated? If not, the object hasn’t changed, skip the rest of the tests and don’t publish it.
3. Has the latest effectivity been released to the distribution targets of interest? If not, the object has changed and should be published.

Windchill ESI uses ReleaseActivity objects to determine whether or not the effectivity information has been published. See the Transaction Management section of this document for further details.

3.10.3 Classes

These are the Windchill ESI classes that support effectivity publication:

**ESI Effectivity Handlers**

![ESI Effectivity Release Support Objects Diagram]

**3.10.3.1 ESIEffectivityHandlerFactory**

Creates an instance of ESIEffectivityHandler. The class provides these methods:
3.10.3.1 newESIEffectivityHandler() : ESIEffectivityHandler
   Reads ESIProperties to obtain the name of the ESIEffectivityHandler implementation class to instantiate. Creates a new instance and returns it to the caller.

3.10.3.2 ESIEffectivityHandler
   An interface that defines the methods that must be available on an Windchill ESI effectivity handler. The interface has these methods:

3.10.3.2.1 getLatestEffectivity(part : EffManagedVersion) : Eff
   Returns the latest non-product-specific Date Effectivity that is associated with an object.

3.10.3.2.2 isReleased(assn : ESITargetAssociation) : boolean
   Given an ESITargetAssociation containing an Eff and an ESITarget, returns true if the Eff has been published to the ESITarget.

3.10.3.3 ESIEffectivityHandlerImpl
   The default implementation of the Windchill ESI effectivity handler interface. The class has these available methods:

3.10.3.3.1 getLatestEffectivity(part : EffManagedVersion) : Eff
   Returns the latest non-product-specific Date Effectivity that is associated with an object.

3.10.3.3.2 isReleased(assn : ESITargetAssociation) : boolean
   Given an ESITargetAssociation containing an Eff and an ESITarget, returns true if the Eff has been published to the ESITarget.

3.10.3.3.3 getLastExportedEffectivity(part : WTPart, target : ESITarget) : Eff
   This is a protected method that returns the last published Eff object that is associated with a given WTPart and a given ESITarget. The method delegates the work to ESITransactionUtility.latestEffExport().

3.10.4 Customization Points

3.10.4.1 Extendable classes
   Extendable Windchill ESI classes described in the Effectivity Handling section of this document must be defined in Windchill ESI properties if they are extended. The following class is designed to permit extensions by customizers:

   - ESIEffectivityHandler
     Property name: com.ptc.windchill.esi.rnd.class.ESIEffectivityHandler
     Default value: com.ptc.windchill.esi.rnd.ESIEffectivityHandlerImpl

3.10.4.2 Configurable Options
   Windchill ESI properties control several configurable options for Windchill ESI Transaction Management. The options are:
• Define the format of effectivity dates. (The value must be a format that can be used by java.text.DateFormat.)
  Property name: com.com.ptc.windchill.esi.rnd.effectivityFormat
  Default value: yyyy-MM-dd HH:mm:ss

• Define the name of the XML element to create when processing effectivity start date.
  Property name: com.ptc.windchill.esi.rnd.StartEffectivity
  Default value: StartEffectivity

• Define the name of the XML element to create when processing effectivity end date.
  Property name: com.ptc.windchill.esi.rnd.EndEffectivity
  Default value: EndEffectivity

• Define the name of the effectivity class (subclass of wt.effectivity.Eff) that is to be processed.
  Property name: com.ptc.windchill.esi.rnd.class.Effectivity
  Default value: wt.effectivity.WTDatedEffectivity

3.11 Alternate Item Group Handling

3.11.1 Background

SAP manages BOM component substitutes by assigning the component and all of its substitutes to an alternate item group. Each BOM can have many alternate item groups. The alternate item groups have a two-character alphanumeric identifier, which must be unique on the BOM.

When maintaining Windchill substitutes on a SAP R/3 system, the correct alternate item group must be provided. It would be possible to relate alternate item group identifier to BOM line numbers. However, in the case where BOM line numbers are not used, the alternate item group numbers a BOM must be saved when the BOM is created or updated so that future maintenance activities have access to the correct alternate item group identifier.
Suppose that an assembly, 100-A, has two components, 200, and 300. Assembly component 200 has a substitute 400. Component 300 has no substitutes (see Figure 65, below).

![Simple Assembly with Alternate Item Group](attachment:image1)

**Figure 65**  
Simple Assembly with Alternate Item Group

Any one assembly can contain multiple components, as shown in Figure 65. The same assembly can also contain multiple instances of the same component. The components are attached to the assembly by WTPartUsageLink objects. The WTPartUsageLink objects may have line numbers, as shown in Figure 66. If line numbers are used, there is one alternate item group for every line number that has a substitute.

![Alternate Item Groups on an Assembly with Line Numbers](attachment:image2)

**Figure 66**  
Alternate Item Groups on an Assembly with Line Numbers
The WTPartUsageLink objects may not have line numbers, as shown in Figure 67. If line numbers are not used, there is one alternate item group if one of the links between the assembly and the component has a substitute.

![Figure 67  Alternate Item Groups on an Assembly without Line Numbers]
3.11.2 Classes

Windchill ESI services provides two classes that support Alternate Item Groups. The classes are not intended to be extended by customizers, but they may be used by customizers. These are the classes:

![Diagram of Alternate Item Group and related classes]

Figure 68 Alternate Item Group Support in Windchill ESI

3.11.2.1 AlternateItemGroup

AlternateItemGroup is a subclass of ObjectToObjectLink and is therefore a persistable object. An AlternateItemGroup object links a WTPart (the BOM assembly part) to a WTPartMaster (the BOM component). The AlternateItemGroup object is created when a BOM is published. The object is not available to Windchill GUI clients. It has these attributes with public getters and setters:

3.11.2.1.1 Number

The unique alternate item group number. This is stored as an integer, but rendered as a two-byte string. The string is the base-36 value of the number.

3.11.2.1.2 lineNumber

The line number of the usage link that this AlternateItemGroup object describes. If line numbers are not supported, the value of this attribute is −1.
3.11.2.2 AlternateItemGroupMgr

The AlternateItemGroupMgr class provides several useful utility methods for handling AlternateItemGroup objects. These are the available methods of the class:

3.11.2.2.1 createAlternateItemGroup(assembly : WTPart, partUsageInfo : PartUsageInfo) : AlternateItemGroup

Creates an AlternateItemGroup object in database based upon the given PartUsageInfo object.

3.11.2.2.2 getAlternateItemGroups(assembly : WTPart) : Collection

Returns a collection of the AlternateItemGroup objects that are associated to an assembly part.

3.11.2.2.3 getAlternateItemGroup(assembly : WTPart, partUsageInfo : PartUsageInfo) : AlternateItemGroup

Returns an AlternateItemGroup, if one exists in the database. Uses a WTPart to identify the assembly. Uses a PartUsageInfo to identify the line number and component on the assembly.

3.11.2.2.4 hasSubstitutes(assembly : WTPart, partUsageInfo : PartUsageInfo) : boolean

Checks whether substitutes exist on a part usage link associated to a PartUsageInfo object.

3.11.2.2.5 getAlternateItemGroup(assembly : WTPart, component : WTPartMaster, lineNumber : LineNumber)

Returns an alternate item group, if there is a line number, based upon an assembly part and line number; otherwise, considering as consolidated usage links, this means that only one association exists between an assembly and more than one of the same components.

3.11.2.2.6 getAIGNumber(aig : AlternateItemGroup)

Returns an AlternateItemGroup id.

3.11.3 Creating and Deleting AlternateItemGroup Objects

The Windchill ESI BOM renderer is responsible for determining when AlternateItemGroup objects should be created or deleted. The BOMRenderer calls the getBomDifferences() API, which returns an array of enumerations. If:

- The enumeration corresponding to added substitutes has any entries, the Windchill ESI BOM renderer asks AlternateItemGroupMgr for the AlternateItemGroup object for that link. If none exists, the Windchill ESI BOM renderer asks AlternateItemGroupMgr to create one.
- The enumeration corresponding to deleted substitutes has any entries, the Windchill ESI BOM renderer asks AlternateItemGroupMgr for the AlternateItemGroup object for that link. If one exists, the Windchill ESI BOM renderer asks AlternateItemGroupMgr whether there are any substitutes remaining on the link. If not, the Windchill ESI BOM renderer asks AlternateItemGroupMgr to delete the AlternateItemGroup object.
• The enumeration corresponding to deleted components has any entries, the Windchill ESI BOM renderer asks AlternateItemGroupMgr for the AlternateItemGroup object for that link. If one exists, the Windchill ESI BOM renderer asks AlternateItemGroupMgr to delete the AlternateItemGroup object.

Existing AlternateItemGroup objects for a product structure is cloned when the PartUsageLink objects are copied at checkin or checkout. The StandardESIService registers its interest in the COPY LINK event. When a WTPart is created, the StandardESIService asks the AlternateItemGroupMgr to create new AlternateItemGroup objects on the new part object.

Alternate item group support is optional. If the Windchill ESI property, com.ptc.windchill.esi.BOM.enabletAlternateItemGroups is not set to true, the Windchill ESI Alternate Item Group features are not available. The default value for the property is true.

4 EAI Software ComponentsNaming Standards

This section describes the naming standards that have been followed in the development of the EAI software components. It is strongly suggested to follow these standards when customizing the code to provide consistency to the solution.

4.1 Overall Naming Governance

Overall Rules

The following general rules should be followed consistently throughout all object naming in the EAI software components. Exceptions are noted explicitly.

• Fields should be separated by underscores (_)
  - If the contents of a field contain underscores, the underscores must be removed, except in the case of a BAPI. (That is, underscores within a particular field are acceptable.)
• No word should be abbreviated.
• All acronyms should be entirely capitalized.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windchill ESI Camelback</td>
<td>Multiple words in a phrase are to be concatenated together, with the first</td>
</tr>
<tr>
<td>Camelback</td>
<td>letter of each word capitalized, except the first letter of the expression.</td>
</tr>
<tr>
<td>Notation</td>
<td>For example: printDebugMessageAndExit()</td>
</tr>
<tr>
<td>Windchill ESI Capitaliz</td>
<td>Multiple words in each field should be concatenated together, with</td>
</tr>
<tr>
<td>Capitalization</td>
<td>capitalized first letters. For example: LocalMessages</td>
</tr>
<tr>
<td>Camelization</td>
<td></td>
</tr>
<tr>
<td>Notation</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Naming Standard - Phrase Definitions
4.2 XML Naming

4.2.1 Schema Naming

4.2.1.1 Primary Schema Definitions

ESI<Description>

Schemas that constitute a subset of a Primary Schema Definition should contain a
descriptive functional name that relates to its contents and follow Windchill ESI
Capitalization Notation.

For example, the schema that defines the input interface to the Logging Service should
be called ESILog.

4.2.1.2 Secondary Schema Definitions

<Description>

Schemas that constitute a subset of a Primary Schema Definition should contain a
descriptive functional name that relates to its contents and follow Windchill ESI
Capitalization Notation.

For example, the component of the Common Business Object that contains information
for publishing Documents should be called:

Document

4.2.2 Tag Naming

Business Object Tag Names

<Object Name> </Object Name>

The following are acceptable values for Business Objects:

<table>
<thead>
<tr>
<th>Object Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
<td>Part</td>
</tr>
<tr>
<td>BOM</td>
<td>BOM</td>
</tr>
<tr>
<td>ECN</td>
<td>ECN</td>
</tr>
<tr>
<td>Document</td>
<td>Document</td>
</tr>
<tr>
<td>DocumentLink</td>
<td>Document Link</td>
</tr>
<tr>
<td>UserArea</td>
<td>User Area</td>
</tr>
<tr>
<td>Substitute</td>
<td>Substitute</td>
</tr>
<tr>
<td>Object Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>ReferenceDesignators</td>
<td>Reference Designators</td>
</tr>
<tr>
<td>AlternateLinks</td>
<td>Alternate Links</td>
</tr>
<tr>
<td>Components</td>
<td>Components</td>
</tr>
</tbody>
</table>

Table 5  Business Object values

Element Tag Names

```xml
<AttributeName></AttributeName>
```

XML tags are given logical names that correspond to the content of the field and follow Windchill ESI Capitalization Notation.

For example, a CBO tag containing a document number reference for the ERP system would be called:

```xml
<ERPDocumentNumber></ERPDocumentNumber>
```

4.3 TIBCO BusinessWorks Naming

4.3.1 Shared Resources

4.3.1.1 TIBCO Rendezvous

**Rendezvous (RV) Configurations**

The Rendezvous Configuration in the EAI software components has a constant name: *ESIMaster_RVTransport*

**SAP R/3Adapter Subjects**

```xml
<Domain>.<Deployment>.adr3.<Configuration>.<API>.<ESIR3SystemID>.<ESIR3Client>.<ESIR3Locale>
```

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>The BusinessWorks domain in use.</td>
</tr>
<tr>
<td>Deployment</td>
<td>The deployment name of the repository in use.</td>
</tr>
<tr>
<td>Configuration</td>
<td>The object configuration name</td>
</tr>
<tr>
<td>API</td>
<td>The API being used</td>
</tr>
<tr>
<td>ESIR3SystemID</td>
<td>The destination SAP R/3 system that the data is being published to.</td>
</tr>
<tr>
<td>ESIR3Client</td>
<td>The destination SAP R/3 Client that the data is being published to.</td>
</tr>
<tr>
<td>ESIR3Locale</td>
<td>The destination SAP R/3 locale that the data is</td>
</tr>
</tbody>
</table>
For example, a message being published to SAP R/3 to create a part with Deployment Production_Deployment, Domain masterDomain, SystemID DV1, Client 850, and US locale would be called:

\texttt{Production\_Deployment.masterDomain.adr3.PartConfiguration.BAPI\_MATERIAL\_SA VEDATAServer.DV1.850.en\_US}

### 4.3.1.2 JMS Application Properties & Configurations

The JMS Configuration for the EAI software components has a constant name: \texttt{ESIMaster\_JMSTransport}

**JMS Queue Naming**

### 4.3.1.3 JMS Queues Between Windchill and TIBCO

\texttt{com.ptc.windchill.esi.<Identifier>}

JMS queue names should be organized as a sequenced hierarchy, separated by periods (.), in increasing order of detail. All queue names should begin with the phrase "com" and end with some piece of information that identifies the specific data being transported. Words in each field should follow \textit{Windchill ESI Capitalization Notation}.

The following are acceptable contents for the Identifier field:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>The initial Event generated by Windchill.</td>
</tr>
<tr>
<td>DataRequest</td>
<td>The request for information sent to Windchill from TIBCO.</td>
</tr>
<tr>
<td>DataResponse</td>
<td>The response with business data to TIBCO from Windchill.</td>
</tr>
<tr>
<td>Result</td>
<td>The results of the transaction.</td>
</tr>
<tr>
<td>ResultResponse</td>
<td>The result of Windchill’s commitment of transaction information.</td>
</tr>
</tbody>
</table>

### Table 7 JMS Queues Identifiers

Refer to the Interface Control Document for complete specifications of all JMS queues.

For example, a queue that carries the Windchill Event information should be called: \texttt{com.ptc.windchill.esi.Event}
4.3.1.4 JDBC Configurations

The JDBC Configuration for the EAI software components has a constant name: 
<code>ESIMaster_JDBCTransport</code>

4.3.1.5 Schema Definitions

Schemas should be given the same name within BusinessWorks as their filename. 

For example, the ESILog.xsd schema should be called: 
<code>ESILog</code>

4.3.2 Adapter Configurations

4.3.2.1 SAP R/3 Adapter Configurations

<code>&lt;Type&gt;Configuration</code>

Because of the TIBCO adapter architecture, it is necessary to have a different adapter configuration based on the business object it processes.

The following describes possible Types:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
<td>If the R/3 Adapter is for manipulating ERP Parts.</td>
</tr>
<tr>
<td>BOM</td>
<td>If the R/3 Adapter is for manipulating ERP BOMs.</td>
</tr>
<tr>
<td>Document</td>
<td>If the R/3 Adapter is for manipulating ERP Documents.</td>
</tr>
<tr>
<td>ECN</td>
<td>If the R/3 Adapter is for manipulating ERP ECNs.</td>
</tr>
</tbody>
</table>

<code>Table 8 Configuration Types</code>

For example, the material master R/3 Adapter should be called: 
<code>PartConfiguration</code>

4.3.2.2 Folders

<code>&lt;FolderName&gt;</code>

Folders should contain a descriptive functional name that relates to its contents and follow <a>Windchill ESI Capitalization Notation</a>.
For example, a folder containing SAP R/3 connections should be called:
R3Connections

4.3.3 Process Definitions

4.3.3.1 Process Definitions for Primary ERP Processing

<Starter>_<Object>_<Action>_PD

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starter</td>
<td>The action that invoked the Process Definition. See below for</td>
</tr>
<tr>
<td></td>
<td>possibilities.</td>
</tr>
<tr>
<td>Object</td>
<td>The name or type of the data that the data processes.</td>
</tr>
<tr>
<td>Action</td>
<td>The action that occurs when the Process Definition ends. See</td>
</tr>
<tr>
<td></td>
<td>below for possibilities.</td>
</tr>
</tbody>
</table>

Table 9 Primary Process Definition Fields

Starter Field
The following describes possible Starters:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS</td>
<td>If the process is triggered by receipt of a message on a JMS Queue.</td>
</tr>
<tr>
<td>RV</td>
<td>If the process is triggered by receipt of a RV message.</td>
</tr>
<tr>
<td>Start</td>
<td>If the process is started by an ordinary Start node.</td>
</tr>
<tr>
<td>Timer</td>
<td>If the process is started by a timer.</td>
</tr>
<tr>
<td>File</td>
<td>If the process reads or polls a file.</td>
</tr>
<tr>
<td>JDBC</td>
<td>If the process reads or polls a database.</td>
</tr>
</tbody>
</table>

Table 10 Primary Process Definition Starter Fields

Object Field
The following describes possible Objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>If the process primarily handles an Error.</td>
</tr>
<tr>
<td>CBOPart</td>
<td>If the process primarily handles a Part.</td>
</tr>
<tr>
<td>CBOBOM</td>
<td>If the process primarily handles a BOM.</td>
</tr>
<tr>
<td>Response</td>
<td>If the process primarily handles an Windchill ESI Response.</td>
</tr>
<tr>
<td>Request</td>
<td>If the process primarily handles an Windchill ESI Request.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>CBODocument</td>
<td>If the process primarily handles a Document.</td>
</tr>
<tr>
<td>CBOECN</td>
<td>If the process primarily handles an ECN.</td>
</tr>
<tr>
<td>CBODocumentLink</td>
<td>If the process primarily handles a Document Link.</td>
</tr>
<tr>
<td>ControlData</td>
<td>If the process primarily handles internal EAI software components processing, rather than business data.</td>
</tr>
<tr>
<td>ObjectHeader</td>
<td>If the process primarily handles CBO header information.</td>
</tr>
<tr>
<td>Log</td>
<td>If the process primarily handles logging.</td>
</tr>
</tbody>
</table>

**Table 11  Primary Process Definition Object Fields**

**Action Field**
The following describes possible Actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PublishJMS</td>
<td>If the process publishes to a JMS Queue.</td>
</tr>
<tr>
<td>PublishRV</td>
<td>If the process publishes to an RV Subject.</td>
</tr>
<tr>
<td>SAPCreate</td>
<td>If the process creates an object in SAP R/3.</td>
</tr>
<tr>
<td>SAPChange</td>
<td>If the process changes an object in SAP R/3.</td>
</tr>
<tr>
<td>SAPDelete</td>
<td>If the process deletes an object in SAP R/3.</td>
</tr>
<tr>
<td>ESIRelease</td>
<td>If the process releases data from Windchill ESI.</td>
</tr>
<tr>
<td>JDBCInsert</td>
<td>If the process inserts an object into a database.</td>
</tr>
<tr>
<td>JDBCSelect</td>
<td>If the process selects an object from a database.</td>
</tr>
<tr>
<td>JDBCDelete</td>
<td>If the process deletes an object from a database.</td>
</tr>
<tr>
<td>JDBCUpdate</td>
<td>If the process updates an object in the database.</td>
</tr>
</tbody>
</table>

**Table 12  Primary Process Definition Action Fields**

If a process definition performs more than one operation, concatenate the Actions together, for example, SAPCreateChangeDelete.

Words in each field should follow [Windchill ESI Capitalization Notation](#).

For example, a process definition that subscribes to the Response queue and creates a Part object in SAP R/3 should be called:

`JMS_Part_SAPCreate_PD`

**4.3.3.2 Process Definitions for Secondary Process Diagrams**

`<Starter>_<Object>_<Action>_<Purpose>_PD`

Use this standard for Process Diagrams that do not perform primary object processing.
Starter Field
The following describes possible Starters:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS</td>
<td>If the process is triggered by receipt of a message on a JMS Queue.</td>
</tr>
<tr>
<td>RV</td>
<td>If the process is triggered by receipt of a RV message.</td>
</tr>
<tr>
<td>Start</td>
<td>If the process is started by an ordinary Start node.</td>
</tr>
<tr>
<td>Timer</td>
<td>If the process is started by a timer.</td>
</tr>
<tr>
<td>File</td>
<td>If the process reads or polls a file.</td>
</tr>
<tr>
<td>JDBC</td>
<td>If the process reads or polls a database.</td>
</tr>
</tbody>
</table>

Table 13  Secondary Process Definition Starter Fields

Object Field
The following describes possible Objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>If the process primarily handles an Error.</td>
</tr>
<tr>
<td>CBOPart</td>
<td>If the process primarily handles a Part.</td>
</tr>
<tr>
<td>CBOBOM</td>
<td>If the process primarily handles a BOM.</td>
</tr>
<tr>
<td>Response</td>
<td>If the process primarily handles an Windchill ESI Response.</td>
</tr>
<tr>
<td>Request</td>
<td>If the process primarily handles an Windchill ESI Request.</td>
</tr>
<tr>
<td>CBODocument</td>
<td>If the process primarily handles a Document.</td>
</tr>
<tr>
<td>CBOECN</td>
<td>If the process primarily handles an ECN.</td>
</tr>
<tr>
<td>CBODocumentLink</td>
<td>If the process primarily handles a Document Link.</td>
</tr>
<tr>
<td>ControlData</td>
<td>If the process primarily handles internal EAI software components processing, rather than business data.</td>
</tr>
<tr>
<td>ObjectHeader</td>
<td>If the process primarily handles CBO header information.</td>
</tr>
<tr>
<td>Log</td>
<td>If the process primarily handles logging.</td>
</tr>
</tbody>
</table>

Table 14  Secondary Process Definition Object Fields

Action Field
The following describes possible Actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>If the process performs an update of some kind, either to an existing object in ERP or to a database,</td>
</tr>
<tr>
<td>Publish</td>
<td>If the process publishes information to RV or JMS.</td>
</tr>
<tr>
<td>Receive</td>
<td>If the process receives information from RV or JMS.</td>
</tr>
<tr>
<td>Process</td>
<td>If the process performs a subset of the processing required by</td>
</tr>
</tbody>
</table>
the larger object process.

| Stage | If the process prepares data for use elsewhere. |

**Table 15  Secondary Process Definition Action Fields**

**Purpose Field**
The following describes possible Purposes:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostResult</td>
<td>If the process performs work related to PostResult messaging.</td>
</tr>
<tr>
<td>Effectivity</td>
<td>If the process performs work related to Effectivity.</td>
</tr>
<tr>
<td>RevisionLevel</td>
<td>If the process performs work related to Revision Levels.</td>
</tr>
<tr>
<td>DocumentVersion</td>
<td>If the process performs work related to Document Versions.</td>
</tr>
<tr>
<td>ByOrganization</td>
<td>If the process exists solely to perform an action by organization.</td>
</tr>
<tr>
<td>Iterate</td>
<td>If the process exists solely to iterate over an object.</td>
</tr>
</tbody>
</table>

**Table 16   Secondary Process Definition Purpose Fields**

For example, a process that processes PostResult messages for a Document Link process should be called:

*Start_CBODocumentLink_Publish_PostResult*

### 4.3.3.3 Common Process Definitions

*<Purpose>_Service*

The following describes possible Purposes:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>The common logging service.</td>
</tr>
<tr>
<td>ErrorHandling</td>
<td>The common error handling service.</td>
</tr>
<tr>
<td>WCResult</td>
<td>The service to publish a result to Windchill.</td>
</tr>
<tr>
<td>SubTransactionPostResult</td>
<td>The service to publish a SubTransaction Post Result to Windchill.</td>
</tr>
</tbody>
</table>

**Table 17   Common Process Definitions Purposes**

For example, the common logging process should be called:

*Logging_Service*
4.3.4 Groups

4.3.4.1 Groups that Retry APIs

APIRetry_<APIName>_<_Context>

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIName</td>
<td>The name of the API that is being retried.</td>
</tr>
<tr>
<td>Context</td>
<td><em>(Optional)</em> If the API is being called more than once in the process, some kind of context may be applied using Windchill ESI Capitalization Notation.</td>
</tr>
</tbody>
</table>

Table 18  Groups that Retry APIs Fields

For example, an API call to BAPI_REV_LEVEL_MAINTAIN that processes all Create Parts (and another loop exists in the same process diagram to process Change and Delete Parts), should be called:

APIRetry_BAPI_REV_LEVEL_MAINTAIN_CreatePart

4.3.4.2 Groups that Error Handle

ErrorGroup_<LogicBlock>

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogicBlock</td>
<td>A logical name for the block of logic, using Windchill ESI Capitalization Notation.</td>
</tr>
</tbody>
</table>

Table 19  Groups that Error Handle Fields

For example, a group that handles all errors from a block of code that commits Changed Parts should be called:

ErrorGroup_CommitPartChange

4.3.4.3 Groups that Iterate

Iterate_<Object>_Until_<End>

Object Field
The following are acceptable values for the Object field:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>If the process primarily handles an Error.</td>
</tr>
<tr>
<td>Part</td>
<td>If the process primarily handles a Part.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>Error</td>
<td>The following are acceptable values for the Error field:</td>
</tr>
<tr>
<td>All</td>
<td>If this iterator is intended to process every object.</td>
</tr>
<tr>
<td>&lt;Condition&gt;</td>
<td>If this iterator has some other condition that makes it end, enter a logical description into the condition field.</td>
</tr>
</tbody>
</table>

**Table 20 Object Fields for Groups that Iterate**

**End Field**

For example, an iterator that loops over all Parts in a given CBO Object should be called:

`Iterate_CBOPart_Until_All`

### 4.3.5 Branches

**Branch_<Description>**

The following describes the fields in this standard, to appear in the label field on the branch:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A logical name for condition that triggers the branch being labeled, using Windchill ESI Capitalization Notation.</td>
</tr>
</tbody>
</table>

**Table 22 Branch Description**

For example, a branch that branches when an API invoke functionally succeeds:

`Branch_APIFunctionalSuccess`
Conversely, a branch that branches when an API invoke functionally fails:
*Branch_APIFunctionalFailure*

### 4.3.6 Activities (Tasks)

#### 4.3.6.1 Constant Names

**Generate Error**
The master Generate Error activity has a constant name:
*GenerateError_ESIMaster*

**Write Log**
The master Write Log activity has a constant name:
*Write Log_ESIMaster*

#### 4.3.6.2 Variable Names

\(<Node Name>_<Field 1>_<Field 2>\)

Activity naming depends on the type of activity.

The following describes each anticipated node type and how to name it.

<table>
<thead>
<tr>
<th>Node</th>
<th>Node Name</th>
<th>Field 1</th>
<th>Field 2</th>
<th>Example</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read File</td>
<td>ReadFile</td>
<td>File Name without path or targetID information, if applicable.</td>
<td>None, as the node contains the information provided in the file and does no other manipulation of it</td>
<td>ReadFile_ESIDefault_BAPI_MASTER_SAVEDATA</td>
<td></td>
</tr>
<tr>
<td>Write File</td>
<td>WriteFile</td>
<td>File Name without path or targetID information, if applicable.</td>
<td>None, as the node contains the information provided in the file and does no other manipulation of it</td>
<td>WriteFile_ESIDebugLog</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Node Name</td>
<td>Field 1</td>
<td>Field 2</td>
<td>Example</td>
<td>Comment</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JDBC Update</td>
<td>JDBCUpdate</td>
<td>None, as this is a one-way communication</td>
<td>The Service it references. Log or CrossReference</td>
<td>JDBCUpdate_ESILog_CommitBOM</td>
<td>A few words strung together to give a brief description of what was logged. No underscores.</td>
</tr>
<tr>
<td>JDBC Query</td>
<td>JDBCQuery</td>
<td>None, as this is a one-way communication</td>
<td>The Service it references. Log or CrossReference</td>
<td>JDBCQuery_ErrorHandlingCode_HandlingCode</td>
<td>A few words strung together to give a brief description of what is being queried. No underscores.</td>
</tr>
<tr>
<td>JMS Queue Sender</td>
<td>JMSSender</td>
<td>The schema (if no schema is characterized, then use the business object name) from Process Data column of the Input tab that is being mapped to the JMS Body (no underscores). For example, CBOPart or ESIPostResult</td>
<td>Everything after the last dot in the queue name (after com.ptc.windchill.esi.) For example Response JMSSender_ESIPostResult_Result</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node Name</td>
<td>Field 1</td>
<td>Field 2</td>
<td>Example</td>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>JMS Queue Receiver</td>
<td>JMSReceiver</td>
<td>Everything after the last dot in the queue name (after com.ptc.windchill.esi.) For example Response</td>
<td>The schema (if no schema is characterized, then use the business object name) from Process Data column of the Input tab that is being mapped to the JMS Body (no underscores). For example, CBOPart or ESIPostResult</td>
<td>Assumes that there is only one JMS receiver on each queue</td>
<td></td>
</tr>
<tr>
<td>RV Subscriber</td>
<td>RVSubscriber</td>
<td>The Subject Name after com.ptc.windchill.esi: For example Error</td>
<td>The schema (if no schema is characterized, then use the business object name) from Process Data column of the Input tab that is being mapped to the RV Message (no underscores). For example, CBOPart or ESIPostResult</td>
<td>RVSubscriber_Error_ESIErrror</td>
<td></td>
</tr>
<tr>
<td>Publish RV Message</td>
<td>RVPublisher</td>
<td>The schema (if no schema is characterized, then use the business object name) from Process Data column of the Input tab that is being mapped to the RV Message (no underscores). For example, CBOPart or ESIPostResult</td>
<td>The Subject Name after com.ptc.windchill.esi: For example Error</td>
<td>RVPublisher_ESIErrror_Error</td>
<td></td>
</tr>
<tr>
<td>Parse XML</td>
<td>XMLParser</td>
<td>Input Schema into Parser only if the input schema is different than the schema that you are parsing against</td>
<td>Schema Parsing against: ESIEvent</td>
<td>XMLParser_ESIEvent</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Node Name</td>
<td>Field 1</td>
<td>Field 2</td>
<td>Example</td>
<td>Comment</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Render XML</td>
<td>XMLRenderer</td>
<td>Input Schema into Renderer only if the input schema is different than the schema that you are rendering against. Typically, a map and render are combined, and thus this is the schema of the Process Data column of the Input tab.</td>
<td>Schema Rendering into: ESIREquest</td>
<td>XMLRenderer_ESIEvent_ESIREquest _GetPart</td>
<td></td>
</tr>
<tr>
<td>Checkpoint</td>
<td>CheckPoint</td>
<td>WC or SAP R/3 interaction schema that was last executed. For example, ESIREsponse.</td>
<td>None.</td>
<td>CheckPoint_ESIREsponse</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Node Name</td>
<td>Field 1</td>
<td>Field 2</td>
<td>Example</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mapper</td>
<td>Mapper</td>
<td>Input Schema Name - on the input tab, the Schema of the Process Data being used. See XML schemas for a list of the schemas used in Windchill ESI and use the schema name - without the underscores - except when using API's.</td>
<td>Output Schema Name - the schema that is selected on the Output Schema tab, and used to map to the Activity Input column of the Input tab.</td>
<td>Mapper_ESIResponse_CBOMaster</td>
<td>In the case where the same schemas are being mapped from input to output, a description field should be included: The description should not have spaces or underscores - only two or three words strung together to describe the activity: Mapper_CBOMaster_CBOMaster_FilterNoOrgs</td>
</tr>
<tr>
<td>Java Code</td>
<td>Java</td>
<td>Description of what the class does using Windchill ESI Capitalization Notation.</td>
<td></td>
<td>Java_GarbageCollection</td>
<td></td>
</tr>
<tr>
<td>Generate Error</td>
<td>Error</td>
<td>Process Definition Name without the &quot;PD&quot;.</td>
<td></td>
<td>Error_LoggingService</td>
<td>Unsure as to the functionality of this activity.</td>
</tr>
<tr>
<td>Node</td>
<td>Node Name</td>
<td>Field 1</td>
<td>Field 2</td>
<td>Example</td>
<td>Comment</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Call Process</td>
<td>CallProcess</td>
<td>Process Definition Name without the &quot;PD&quot;.</td>
<td>(optional) The inputs to the process.</td>
<td>CallProcess_Logging_Service</td>
<td>Label name is what displays as the label on the process definition workspace.</td>
</tr>
<tr>
<td>Label</td>
<td>A sentence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td>Start</td>
<td>Process Definition Name, without &quot;PD&quot;</td>
<td>None</td>
<td>Start_Logging_Service</td>
<td></td>
</tr>
<tr>
<td>End</td>
<td>End</td>
<td>Process Definition Name, without &quot;PD&quot;</td>
<td>None</td>
<td>End_Logging_Service</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Null</td>
<td>None</td>
<td>(Optional) Description</td>
<td>Null_SimplifyLogBranching</td>
<td></td>
</tr>
<tr>
<td>Invoke an Adapter Request-Response Sequence</td>
<td>Invoke</td>
<td>The name of the BAPI being invoked</td>
<td>(Optional) The BAPI previously invoked.</td>
<td>Invoke_BAPI_TRANSACTION_COMMIT_BAPI_DOCUMENT_CREATE2</td>
<td></td>
</tr>
<tr>
<td>Write to Log</td>
<td>WriteLog</td>
<td>Description of what information is being written to the log using Windchill ESI Capitalization Notation.</td>
<td>None</td>
<td>WriteLog_Invoked_BAPI_MATERIAL_SAVEDATA</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4 Code-Level Naming

#### 4.4.1 Global Variables

_ESI<VariableName>

In this standard, the Variable Name field must be populated with a brief, descriptive and unique name, using [Windchill ESI Capitalization Notation](#).
For example, the global variable to enable debug messaging should be called:

`ESIEnableDebugging`

### 4.4.2 Constants

**ESICONSTANT_<VARIABLENAME>**

In this standard, the Variable Name field must be populated with a brief, descriptive and unique name, and entirely capitalized.

For example, a constant variable to contain the Boolean value "true" should be called:

`ESICONSTANT_TRUE`

### 4.4.3 Methods (Functions)

```
<Action><Data>()
```

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>A verb describing the action of the method.</td>
</tr>
<tr>
<td>Data</td>
<td>The data being manipulated by the method.</td>
</tr>
</tbody>
</table>

Table 24 Method Fields

Function names must adhere to [Windchill ESI Camelback Notation](#).

For example, a method to write a debug message to the screen should be called:

`printDebugMessage()`

Furthermore a generic method to load a file should be called:

`readFile()`

### 4.4.4 Classes

```
<ClassName>
```

Classes must be given a descriptive name using [Windchill ESI Capitalization Notation](#).

For example, the class that contains Debug messages should be called:

`Debug`
4.5 Repository Naming

4.5.1 Primary Builds

ESI<Purpose>Repository_v<BuildNumber>.dat

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>A short description of the functionality of the repository, using Windchill ESI Capitalization Notation.</td>
</tr>
<tr>
<td>BuildNumber</td>
<td>The build number of the repository; always three digits.</td>
</tr>
</tbody>
</table>

Table 25  Primary Build Fields

For example, version 25 of the repository containing primary business logic should be called:  
ESIMasterRepository_v025.dat

Version 25 of a repository containing simulated Windchill functionality should be called:  
ESIWindchillSimulationRepository_v025.dat

4.5.2 Working Repositories

ESI<Purpose>Repository_v<BuildNumber>_<Description>.dat

The following describes the fields in this standard:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>A short description of the functionality of the repository, using Windchill ESI Capitalization Notation.</td>
</tr>
<tr>
<td>BuildNumber</td>
<td>The build number of the repository; always three digits.</td>
</tr>
<tr>
<td>Description</td>
<td>A short description of the changes in functionality contained within the working repository.</td>
</tr>
</tbody>
</table>

Table 26  Working Repository Fields

For example, version 25 of the repository containing primary business logic, containing a modified Create BOM mapping should be called:  
ESIMasterRepository_v025_UpdatedCreateBOMMapping.dat

Version 25 of a repository containing simulated Windchill functionality, with modifications to handle a new error handling code, should be called:  
ESIWindchillSimulationRepository_v025_NewErrorHandlingCode.dat
5 EAI Windchill Simulation Module

5.1 Overview
The Windchill ESI EAI Windchill Simulation module provides a method to test the EAI software components without the need for a Windchill system. The ESI EAI Windchill Simulation module is a BusinessWorks engine that connects to the JMS queues in the same way that Windchill ESI Services would; it publishes Windchill ESI Event, ESIResponse, and Windchill ESI Result Response messages and receives Windchill ESI Requests, ESIPostResult and PostEvent messages. The Windchill ESI EAI Windchill Simulation module does not perform any business logic – the information it sends to the EAI software components is simply read from files.

5.1.1 Purpose
The purpose of the Windchill ESI EAI Windchill Simulation module is to give an end-user a mechanism for simulating the Windchill Enterprise Systems Integration services within TIBCO BusinessWorks. This simulation module enables users to verify the EAI software components without the need for the Windchill Open API. This module also provides a tool for debugging issues that arise during development, QA testing, and technical support by minimizing the number of possible contributing factors to the issue. It is expected that implementation teams will leverage the module to verify customizations that have been done to the EAI software components and/or Windchill independent of a Windchill environment. The subsequent pages of this document provide an overview of the Windchill ESI EAI Windchill Simulation module as well as instructions and examples for using the module.

5.1.2 What is Simulated
The Windchill ESI EAI Windchill Simulation module is designed to simulate the Windchill Open API. It includes all JMS communication between Windchill and the EAI software components solution. It does not include validation of data or messages coming from Windchill or dynamic generation of Windchill responses to EAI software components requests. The Windchill Open API and the communication with the EAI software components solution is comprised of JMS message communication over a SOAP protocol. The Windchill Open API utilizes services within Windchill to extract object information from the Windchill database. Once this information is extracted, the Windchill Open API formats the data into XML and wraps it in SOAP so that it may be transported over JMS. The data is dropped onto a JMS queue by the Windchill Open API and is picked up and read by the EAI software components solution enabled in TIBCO BusinessWorks. The opposite also occurs when the EAI Software Components solution places requests or updates to data on JMS queues for the Windchill Open API. The Windchill ESI EAI Windchill Simulation module provides a mechanism to facilitate the two-way communication between the Windchill Open API and the EAI software components. The module emulates the sending and receipt of JMS messages by Windchill via the Windchill
Open API. The module invokes the EAI software components to perform the standard operations as defined by the *Windchill Enterprise Systems Integration User’s Guide*.

### 5.1.3 Logical Flow

There are a total of six messages involved in the communication between the Windchill Open API and the EAI software components:

1. ESIEvent
2. ESIRequest
3. ESIResponse
4. ESIPostResult
5. ESIPostEvent
6. ESIResultResponse

Note: Because the PostResult and PostEvent are both published to the `com.ptc.windchill.esi.Result` JMS queue, five queues are used for six message types. Each of these messages is of an XML format, wrapped in SOAP, and transported over JMS. The messaging between Windchill and the EAI software components can be seen in Figure 69.
Each of the numbered circles is described in the following message flow.
1. Windchill notifies Info*Engine of a user-triggered change.
2. Info*Engine generates SOAP encapsulated message.
3. Info*Engine creates JMS message.
5. TIBCO Enterprise for JMS receives the ESIEvent messages and places it on the `com.ptc.windchill.esi.Event` queue for TIBCO BusinessWorks to retrieve.
7. A SOAP Parser extracts the message text of the JMS Text Message to determine the nature of the change within Windchill.
8. TIBCO BusinessWorks formulates a query to Windchill for more information (i.e., the actual data that has changed), sending a new SOAP encapsulated message to the `com.ptc.windchill.esi.DataRequest` JMS queue.
9. TIBCO Enterprise for JMS receives the ESIREquest message from TIBCO BusinessWorks and places it on the queue.
10. Info*Engine reads the ESIREquest message from the JMS queue.
11. JMS is stripped by Info*Engine.
12. SOAP RPC is interpreted by Info*Engine.
13. Info*Engine invokes the RPC specified in the SOAP protocol to acquire the relevant information, that is, the actual new or changed Windchill data.
14. Relevant Information is sent from Windchill to Info*Engine.
15. Relevant information is encoded in SOAP.
16. Relevant information is encoded in JMS.
17. An ESIREsponse message with all relevant content is sent to the `com.ptc.windchill.esi.DataResponse` JMS queue.
18. TIBCO Enterprise for JMS holds the message for delivery to the receiver.
19. New SOAP message is sent to TIBCO BusinessWorks.
20. TIBCO BusinessWorks strips the SOAP encoding, and processes the Windchill data contained in the text from the JMS TextMessage for publication to the target ERP System(s).
21. TIBCO BusinessWorks publishes new message to TIB via Rendezvous Daemon (RVD).
22. ERP Adapter receives relevant information and publishes to ERP System using ERP specific APIs (via TCP/IP).
23. ERP System publishes acknowledgement to ERP Adapter (via TCP/IP).
24. ERP Adapter publishes acknowledgement to TIB.
25. TIBCO BusinessWorks receives acknowledgement and wraps it in SOAP.
26. The ESIPostResult and/or ESIPostEvent message is a SOAP encapsulated acknowledgement that is sent to the `com.ptc.windchill.esi.Result` JMS queue.
27. TIBCO Enterprise for JMS holds the message for delivery to the receiver.
28. ESIPostResult and/or ESIPostEvent Message is sent from TIBCO Enterprise for JMS to Info*Engine.
29. Info*Engine receives acknowledgement message from TIBCO Enterprise for JMS.
30. Info*Engine unwraps SOAP.
31. Acknowledgement is published to Windchill.
32. Windchill sends ESIResultResponse message to Info*Engine.
33. Relevant information is encoded in SOAP.
34. Relevant information is encoded in JMS.
35. ESIResultResponse message with all relevant content is sent to the
   \textit{com.ptc.windchill.esi.ResultResponse} JMS queue.
36. TIBCO Enterprise for JMS holds the message for delivery to the receiver.
   TIBCO BusinessWorks receives and processes the ESIResultResponse.

The Windchill ESI EAI Windchill Simulation module is supplemented with static messages that
can be used to simulate the messages coming from Windchill and Info*Engine. The actions in the
messaging diagram that Windchill and Info*Engine are responsible for are emulated by the
Windchill ESI EAI Windchill Simulation module. Specifically, the Windchill ESI EAI Windchill
Simulation module performs the following functions:
1. Generate ESIEvent messages
   a. ESIEvent messages are identical to those sent by Windchill
2. Place ESIEvent messages on \textit{com.ptc.windchill.esi.Event} JMS queue
3. Listen for ESIRequest message on \textit{com.ptc.windchill.esi.DataRequest} JMS queue
4. Generate ESIResponse messages
   a. ESIResponse messages are identical to those sent by Windchill
5. Place ESIResponse messages on \textit{com.ptc.windchill.esi.DataResponse} JMS queue
7. Listen for ESIPostEvent messages on \textit{com.ptc.windchill.esi.Result} JMS queue
8. Generate ESIResultResponse messages
   a. ESIResultResponse messages are identical to those sent by Windchill
9. Place ESIResultResponse messages on \textit{com.ptc.windchill.esi.ResultResponse} JMS queue

Note: All messages the Windchill Simulation module (ESIEvent, ESIResponse, ESIResultResponse)
sends to the EAI components are identical to those sent by Windchill.

5.2 User’s Guide

5.2.1 Repository Configuration

The EAI Windchill Simulation module is included in
\textit{ESISimulationWindchillRepository_vxxx}. This repository exists entirely separately from the
ESIMasterRepository, and contains the functionality for the Windchill Simulation module, as
well as other Windchill Enterprise Systems Integration utilities.

5.2.2 JMS Configuration

Since the EAI Windchill Simulation module performs the same functions as Windchill would,
it requires the same JMS permissions as Windchill. The EAI Windchill Simulation module
JMS user is defined by the global variable \textit{ESIJMSSTUBUsername} and requires the same
permissions as the user WCESI. By default, the JMS user for the EAI Windchill Simulation
module is set to "ESISTUB". This default can be changed in the Windchill ESI Repository
with the global variables.

Starting from a baseline TIBCO Enterprise for JMS installation that has been configured
according to the \textit{Windchill Enterprise Systems Integration Installation and Configuration
Guide}, complete the following steps:
1. Start the JMS Server.
2. Start the JMS Administration Tool and log in as the administrator.
3. Create the ESISTUB user with the following commands:
   - `create user ESISTUB password=<esistub user password>.
   - `commit

4. Grant the ESISTUB user permission to send information to the Event queue with the following commands:
   ```
   >grant queue com.ptc.windchill.esi.Event ESISTUB send
   >commit
   ```

5. Repeat step 4 for each queue and permission required for ESISTUB as listed below:

<table>
<thead>
<tr>
<th>Queue</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ptc.windchill.esi.Event</td>
<td>send</td>
</tr>
<tr>
<td>com.ptc.windchill.esi.DataRequest</td>
<td>receive</td>
</tr>
<tr>
<td>com.ptc.windchill.esi.DataResponse</td>
<td>send</td>
</tr>
<tr>
<td>com.ptc.windchill.esi.Result</td>
<td>receive</td>
</tr>
<tr>
<td>com.ptc.windchill.esi.ResultResponse</td>
<td>send</td>
</tr>
</tbody>
</table>

   **Table 27  ESISTUB Queue Permissions**

5.2.3 Setting up Data and File Structure

Because the purpose of the EAI Windchill Simulation module is to simulate Windchill while minimizing complexity and contributing factors to a bug, it performs minimal processing. It simply receives information off the queues and any information it places on the queues is read from files.

The simulaton module uses three files: ESIEventUserA.xml (or ESIEventUserB.xml), ESIResponseUserA.xml (or ESIResponseUserB.xml), and ESIResultResponse.xml. These files are discussed in detail in subsequent sections of this document, and samples may be found in the Appendix.

The location of these files is determined by the Global Variable ESIDirectory, which is set to c:\tibco\esi out-of-the-box. To change the path of the ESIResponse and ESIResultResponse files, modify the global variable ESIDirectory. To change the path of the ESIEvent, navigate to the Stub_ESIEvent_Publication_PD process definition in ProcessDefinitions/WindchillCommunication/SimulatedWindchill folder and modify the configuration of the FilePoll_ESIEvent activity.

Note: To run the EAI Windchill Simulation module in a UNIX environment, the global variable ESIDirectory and the configuration in the FilePoll_ESIEvent activity **must** be modified to UNIX equivalent paths (e.g., /opt/tibco/esi).

5.2.4 XML Files

5.2.4.1 ESIEventUserA.xml or ESIEventUserB.xml

The ESIEvent XML file is a small file that is not very complex. There are two fields of note in it.
The TransactionID field is the unique identifier of the transaction that is passed to the EAI Software components from Windchill. The ESIPostResult and ESIPostEvent messages that are sent back to Windchill will be written using this field. It is recommended that the developer change this field to a descriptive name for each test run.

The Class field represents the highest-level object processing required by the subsequent ESIResponse. The hierarchy is as follows:
1. com.ptc.windchill.esi.Change – that is, ECN
2. com.ptc.windchill.esi.BOM
3. com.ptc.windchill.esi.Part

The Class must be set to the highest order of object processing performed. For example, if the release contains BOM processing but no Change Management, then the Class should be set to com.ptc.windchill.esi.BOM.

The fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectID</td>
<td>VR:wt.part.WTPart</td>
<td>Corresponds to the Unique Federation Identifier in Windchill for the object being released.</td>
</tr>
<tr>
<td>TransactionID</td>
<td>ABC-Create-BOM-Test</td>
<td>The Unique Identifier for this release; allows Windchill to associate different JMS communications to a single transaction.</td>
</tr>
<tr>
<td>Class</td>
<td>com.ptc.windchill.esi.Change</td>
<td>The highest level of processing to be performed by the release.</td>
</tr>
<tr>
<td>WaitingEventID</td>
<td>VR:wt.workflow:1232434</td>
<td>The unique identifier that represents a unique association of the release to a Windchill workflow.</td>
</tr>
<tr>
<td>ESITarget/ObjectID</td>
<td>OR:co.ptc.windchill.esi.tgt.ESITarget:032103</td>
<td>The Unique Identifier of the ESITarget object within Windchill.</td>
</tr>
<tr>
<td>ESITarget/TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The string that represents the distribution target destination for the objects in the release.</td>
</tr>
</tbody>
</table>

Table 28  ESIEventUserA.xml Field Descriptions

5.2.4.2 ESIResponseUserA.xml or ESIResponseUserB.xml

The ESIResponse file is the file that contains all of the business data that the EAI Windchill Simulation module will send over the com.ptc.windchill.esi.DataResponse JMS queue for publication to SAP R/3 Release 4.6C. The developer should populate this file with business data bound for SAP R/3 Release 4.6C, according to the ESIResponse schema.

The file contains several structures for data, organized by action (Added, Deleted, Changed, and Unchanged) and by object (ECN, Part, Document, BOM, BOMComponent, and BOMSubstitute). Each structure must be present, even if empty, and in the same order as the data appears in the schema.
The following is a list of key fields for each structure:

Table 29  ESIResponseUserA.xml Field Descriptions

<table>
<thead>
<tr>
<th>Structure</th>
<th>Field</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECN</td>
<td>Class</td>
<td>com.ptc.windchill.esi.ChangeHeader</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>12345</td>
<td>The Change Number.</td>
</tr>
<tr>
<td></td>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The Distribution Target that the ECN is bound for. This field is repeatable and can be listed multiple times.</td>
</tr>
<tr>
<td>Part</td>
<td>Class</td>
<td>com.ptc.windchill.esi.Part</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
</tr>
<tr>
<td></td>
<td>Number</td>
<td>TST_PART_01</td>
<td>The number of the Part.</td>
</tr>
<tr>
<td></td>
<td>Default Unit</td>
<td>ea</td>
<td>The Default Unit of the Part (Unit of Measure in Windchill). This field will be translated according to the translations listed in ESILookups.properties.</td>
</tr>
<tr>
<td></td>
<td>PartType</td>
<td>inseparable</td>
<td>The Type of the Part in Windchill. This field will be translated according to the translations listed in ESILookups.properties.</td>
</tr>
<tr>
<td></td>
<td>Source</td>
<td>make</td>
<td>The Source of the part in Windchill.</td>
</tr>
<tr>
<td></td>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP distribution target that the Part is bound for. This field is repeatable and can be listed multiple times.</td>
</tr>
<tr>
<td>Document</td>
<td>Class</td>
<td>com.ptc.windchill.esi.Document</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>TST_DOC_01</td>
<td>The Document unique number.</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>WCTYPE</td>
<td>wt.doc.WTDocument$$R requirements</td>
</tr>
<tr>
<td></td>
<td>PreviousVersion</td>
<td>A</td>
<td>If Changing the Document, the Version must be incremented; this value is the version in Windchill to iterate from.</td>
</tr>
<tr>
<td></td>
<td>Original</td>
<td><a href="http://wc70.mycompany.com:2715/">http://wc70.mycompany.com:2715/</a></td>
<td>This is the URL of the Original document in Windchill.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WNC70/servlet/WindchillAuthGW/</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>wt.enterprise.URLProcessor/URLtemplateAction?action=ObjProps&amp;oid=VR%3Awt.doc.WTDocument%3A81949&amp;u8=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP distribution target that the Document is bound for. This field is repeatable and can be listed multiple times.</td>
</tr>
<tr>
<td>DocumentLink</td>
<td>Class</td>
<td>com.ptc.windchill.esi.DocumentLink</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
</tr>
<tr>
<td></td>
<td>Document Number</td>
<td>TST_DOC_01</td>
<td>The Document number on which to link the Part.</td>
</tr>
<tr>
<td></td>
<td>Document Version</td>
<td>A</td>
<td>The Version in Windchill of the Document on which to add the Part.</td>
</tr>
<tr>
<td></td>
<td>Associated ObjectNumber</td>
<td>TST_PRT_01</td>
<td>The number of the Part to link the Document to.</td>
</tr>
</tbody>
</table>
### Structure Field Sample Value Description

<table>
<thead>
<tr>
<th>Structure</th>
<th>Field</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP distribution target in which the Part exists to link the Document to. This field is repeatable and can be listed multiple times.</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>com.ptc.windchill.esi.BOMHeader</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>TST_BOM_01</td>
<td>The Part number of the BOM being created or changed.</td>
<td></td>
</tr>
<tr>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP distribution target in which the BOM is bound for. This field is repeatable and can be listed multiple times.</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>com.ptc.windchill.esi.BOMComponent</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
<td></td>
</tr>
<tr>
<td>PartNumber</td>
<td>TST_COMP_01</td>
<td>The Part number of the Component being created or changed.</td>
<td></td>
</tr>
<tr>
<td>Assembly</td>
<td>TST_BOM_01</td>
<td>The Part number of the BOM underneath which this Component is being created or changed.</td>
<td></td>
</tr>
<tr>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP Organization Target in which the Part exists. This field is repeatable and can be listed multiple times.</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>com.ptc.windchill.esi.BOMComponent</td>
<td>This value allows the EAI Software Components to discern the type of object it is processing.</td>
<td></td>
</tr>
<tr>
<td>ComponentPartNumber</td>
<td>TST_COMP_01</td>
<td>The Part number of the Component underneath which this Substitute is being created or changed.</td>
<td></td>
</tr>
<tr>
<td>Assembly</td>
<td>TST_BOM_01</td>
<td>The Part number of the BOM underneath which this Substitute is being created or changed.</td>
<td></td>
</tr>
<tr>
<td>Substitute</td>
<td>TST_SUB_01</td>
<td>The Part number of the Substitute being created or changed.</td>
<td></td>
</tr>
<tr>
<td>TargetID</td>
<td>sap.46c:sys.850:1100</td>
<td>The ERP distribution target in which the Material exists. This field is repeatable and can be listed multiple times.</td>
<td></td>
</tr>
</tbody>
</table>

Examples of this file may be found in the Appendix.

**Note:** Modify this file with great care. Its unparsed XML tags can make editing tricky and any small error will cause the parser to error.

### 5.2.4.3 UserA vs. UserB

The EAI Windchill Simulation module provides the developer the ability to maintain two sets of data at the same time. The Event process listens for any change to one of two different files, ESIEventUserA.xml or ESIEventUserB.xml. The corresponding ESIResponse file (ESIResponseUserA.xml or ESIResponseUserB.xml) will be published to the queues upon receipt of the Windchill ESI Request. This is useful because it allows two developers to simultaneously run data against a single engine instance. Any activity by UserA will not conflict with activity by UserB when run concurrently.
5.2.4.4 ESIResultResponse.xml

This file is read and sent back to TIBCO BusinessWorks on the `com.ptc.windchill.esi.ResultResponse` JMS queue upon receipt of either an ESIPostResult or ESIPostEvent message.

Because the EAI Windchill Simulation module does not process information, it assumes that independent of the contents of ESIPostEvent and ESIPostResult files it receives, a successful ESIResultResponse should be sent back to the EAI software components. For this reason the ESIResultResponse.xml file remains static and a user may use the example provided in the Appendix without modification.

5.2.4.5 ESIPostResult and ESIPostEvent files

The EAI Windchill Simulation module can write any ESIPostResult and ESIPostEvent files it receives from TIBCO BusinessWorks to a file. Each ESIPostResult and ESIPostEvent is written to a file with the following name:

- `<PostResult>_<Transaction ID>_<Timestamp>.xml`
- `<PostEvent>_<Transaction ID>_<Timestamp>.xml`

For example:

- PostResult_Test-Transaction-1A_2003-05-21-170000.xml

If two messages containing the same TransactionID arrive at a close enough time where the timestamp would be the same, both messages are written to the same file.

To enable or disable writing of these files, modify the global variable `ESISTUBWriteResultsToFile` as specified in the next section of this document. The directory to which the files are written is configurable by changing the value of the global variable `ESIDirectory`.

5.2.5 Configuration and Global Variables

The following Global Variables pertain to the EAI Windchill Simulation module:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>ESI 7.0 Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESISTUBJMSPassword</td>
<td>The password for the EAI Windchill Simulation module ESI User.</td>
<td></td>
<td>esistub</td>
</tr>
<tr>
<td>ESISTUBJMSUsername</td>
<td>The username for the EAI Windchill Simulation module ESI User.</td>
<td></td>
<td>ESISTUB</td>
</tr>
<tr>
<td>ESISTUBLocale</td>
<td>This variable is used to set the locale property of JMS headers. This value tells the EAI Software Components the locale of the data in the ESIResponse file.</td>
<td>Any Java Standard locale designation. For example, en_GB for British English or fr_CA for Canadian French.</td>
<td>en_US</td>
</tr>
<tr>
<td>ESISTUBWriteResultsToFile</td>
<td>Enables the EAI Windchill Simulation module to write any ESIPostResult or ESIPostEvent messages it receives to the ESIDirectory.</td>
<td>true, false</td>
<td>true</td>
</tr>
</tbody>
</table>
### Table 30  Windchill Simulation module Global Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>ESI 7.0 Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESIDirectoryName</td>
<td>The directory in which the EAI Windchill Simulation module will look for the ESIResponse and ESIResultResponse, and the directory to which ESIPostResult and ESIPostEvent files will be written.</td>
<td>Any valid directory.</td>
<td>(c:\tibco/esi)</td>
</tr>
<tr>
<td>ESISTUBAppendPostResults</td>
<td>When set to true, this flag will write PostResult and PostEvent files associated to the same TransactionID to the same file. If it is set to false, each PostResult or PostEvent message will be written to its own file.</td>
<td>true, false</td>
<td>false</td>
</tr>
</tbody>
</table>

**Note:** Changes to Global Variables will not take effect until the Engine is restarted.

### 5.2.6 Logging and Tracing

Out-of-the-box, the EAI Windchill Simulation module does not include any logging or tracing of its activity, and does not alter the standard logging behavior of the EAI software components. In order to log or trace the EAI Windchill Simulation module’s activity, the user may leverage TIBCO out-of-the-box tools such as the Write to File activity as they desire.

The EAI Windchill Simulation module is configurable to write any ESIPostResult or ESIPostEvent messages it receives via JMS to a file. To enable or disable this functionality, set the ESISTUBWriteResultsToFile global variable in the repository. To further configure writing of PostResult and PostEvent files, use the ESISTUBAppendPostResults global variable, which controls whether the PostResult and PostEvent messages are written to separate files or appended to the same file.

### 5.2.7 Running the EAI Windchill Simulation Module

The following two sections describe different methods of running an instance of the EAI Windchill Simulation Module.

#### 5.2.7.1 Running the EAI Windchill Simulation Module in Engine Mode

It is recommended to deploy and run a TIBCO BusinessWorks engine to handle all the Windchill Simulation module functions. To deploy a EAI Windchill Simulation Module Engine, follow the same steps to deploy an engine as specified in the Enterprise Systems Integration Installation and Configuration Guide, but complete these additional steps:

1. In the Deployment Configuration, select the Process Definitions tab.
2. Uncheck the "Select All" button.
3. Click the binoculars and select
   Stub_ESIRequest_Subscribe_ESIReponse_Publish_PD in the
   ProcessDefinitions/WindchillCommunication/SimulatedWindchill folder.
4. Repeat step 3 for the other two Stub process diagrams in the same folder,
   Stub_Result_ResultResponse_PD and Stub_ESIEvent_Publication_PD.
5. Deploy the new EAI Windchill Simulation Module Engine.

TIBCO BusinessWorks generates a .cmd file in c:\tibco\bw\2.0\bin that corresponds to
the deployment name specified in the deployment configuration. To run the engine,
follow the instructions for starting a Process Engine from the TIBCO Administration
When this engine is run, it will perform all the functions of the EAI Windchill
Simulation module (listening for and publishing ESIEvents, listening for ESIRequests,
etc).

5.2.7.2 Running the EAI Windchill Simulation Module in Test Mode

   It is also possible to run the entire process in test mode. This allows a user to set
   breakpoints and observe the EAI Windchill Simulation module using the TIBCO
   BusinessWorks debugger.
   To run the process in Test Mode, complete the following steps:
   1. Navigate to the Stub_ESIEvent_Publication_PD process in the
      ProcessDefinitions/WindchillCommunication/SimulatedWindchill folder.
   2. Open the Tester window.
   3. Press the double-green arrow button
   4. Select the following Process Diagrams:
      a. Stub_ESIEvent_Publication_PD
      b. Stub_ESIRequest_Subscribe_ESIReponse_Publish_PD
      c. Stub_Result_ResultResponse_PD
   5. To start the engine, press the Start button. The process will start and begin listening
      for a change to ESIEventUserA.xml or ESIEventUserB.xml.

5.2.8 Running an Instance of the Generator Validator

   The Generator-Validator checks an ESIResponse file to see if the file would parse
   correctly when processed by the EAI software components. To run the
   Generator/Validator, follow the following steps:
   1. Save an ESIResponse file to a separate file in the directory specified by the global
      variable ESIDirectory.
   2. Navigate to the Generator_Validator process definition in the
      ProcessDefinitions/WindchillCommunication/SimulatedWindchill folder.
   3. Click the Tester button and click the single green arrow. This will bring up a test
      data window; in the Filename field enter the filename to be read.
      If the XML is properly formatted then the process definition will run to the End node
      without generating an error. If any error occurs, then the EAI software components are
      unable to parse the Response.
5.2.9 How to Simulate a Windchill Release and Trigger the Windchill ESI Process

In order to trigger an EAI Windchill Simulation Module EAI Software Components process, complete the following steps:
1. Prepare an ESIResponseUserA.xml (or ESIResponseUserB.xml) file with the data intended for publication for SAP R/3 Release 4.6C.
2. Place an ESIResultResponse.xml file in the ESIDirectory.
3. Prepare an ESIEventUserA.xml (or ESIEventUserB.xml) file using a simple text editor with a unique Transaction ID and proper Class value.
4. Ensure all relevant global variables are properly set.
5. Start a TIBCO BusinessWorks engine using one of the two methods above.

5.2.10 Troubleshooting

Q: The Windchill Simulation module engine is started but the process doesn’t start.
A: Starting the engine does not start the process. The engine, once started, listens for changes to ESIEventUserA.xml and ESIEventUserB.xml. The process will begin once one of these files is opened, changed, and saved.

Q: The ESIEventUserA.xml file was changed and saved but the process fails to start.
A: Ensure that the xml file is in the directory specified by the ESIDirectory Global Variable. Also ensure that the file has been changed; some text editors do not save the file if it has not been changed. In UNIX environments, be sure to change the path to ESIEventUserA.xml file in the FilePoll_ESIEvent activity in the Stub_ESIEvent_Publication_PD process definition.

Q: The engine gives an error on startup that it can’t contact the JMS server.
A: Ensure that the ESISTUBJMSUsername and ESISTUBJMSPassword values match the values set when configuring the Windchill ESI Stub JMS user.

Q: The process seems to be reading and publishing messages correctly to the JMS, but the EAI Software Components crashes very close to the beginning of its processing.
A: The XML parser that reads the ESIResponse is very picky. Carefully check your ESIResponse file to ensure that it is properly formatted and that no HTML tags were inadvertently modified. Also, the XML attributes must appear in the same order they appear in the XML schema.

Q: The EAI Software Components process appears to be processing a different ESIResponseUserA.xml file than the one in the directory specified by the Global Variable ESIDirectory.
A: Check for consistent use of ESIEventUserA.xml/ESIResponseUserA.xml and ESIEventUserB.xml/ESIResponseUserB.xml file pairs. If this is not the problem, open the
JMS Administration Tool and check that there are no messages waiting on the JMS queues by running the command "show queues". If messages remain, purge the queues by running run the command "purge all queues".

**Q:** The engine errors when it is restarted.
**A:** In the c:\tibco (Windows) or /opt/tibco (UNIX) folder, delete any folder named "working" and retry.

### 5.3 Technical Design

#### 5.3.1 JMS Configuration

A JMS Configuration, Stub_JMSCConnection, is in the SharedConfigurations/JMS folder. This configuration exists to handle the JMS responsibilities of the EAI Windchill Simulation module. This JMS configuration is set up with a standard set of JMS parameters and is used by all components of the stub that read and write to JMS:

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Out-of-the-box Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Stub JMSCConnection</td>
<td></td>
</tr>
<tr>
<td>User Name</td>
<td>%%ESISTUBJMSUsername%%</td>
<td>This is the Windchill ESI Global Variable ESISTUBJMSUsername, which is set out-of-the-box to ESISTUB</td>
</tr>
<tr>
<td>Password</td>
<td>%%ESISTUBJMSPassword%%</td>
<td>This is the Windchill ESI Global Variable ESISTUBJMSPassword, which is set out-of-the-box to estistub</td>
</tr>
<tr>
<td>Client ID</td>
<td>(blank)</td>
<td></td>
</tr>
<tr>
<td>JNDI Context Factory</td>
<td>com.tibco.tibjms.naming.TibjmsInitialContextFactory</td>
<td></td>
</tr>
<tr>
<td>JNDI Context URL</td>
<td>%%ESIJMSJNDIContextURL%%</td>
<td>This global variable is set out-of-the-box to tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>JNDI User Name</td>
<td>%%ESISTUBJMSUsername%%</td>
<td>This is the Windchill ESI Global Variable ESISTUBJMSUsername, which is set out-of-the-box to ESISTUB</td>
</tr>
<tr>
<td>JNDI Password</td>
<td>%%ESISTUBJMSPassword%%</td>
<td>This is the Windchill ESI Global Variable ESISTUBJMSPassword, which is set out-of-the-box to estistub</td>
</tr>
</tbody>
</table>

**Table 31  Windchill Simulation Module JMS Configuration**

#### 5.3.2 Generator_Validator

The XML validator simply reads a Response file and validates that the file will parse correctly when processed by the EAI software components.

#### 5.3.2.1 Logical Flow

This process simply reads a file, strips off the SOAP wrapper, and then parses the XML inside. It is intended for testing purposes only and mimics the process that the EAI software components use to process a Windchill ESI Response.
### 5.3.2.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>A file name to read from the ESIDirectory</td>
<td>A file name to read from the ESIDirectory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read File</td>
<td>ESIDirectory Global Variable and filename parameter from the Start node</td>
<td>The file present at ESIDirectory/filename</td>
<td>The file the node intends to read exists.</td>
<td></td>
</tr>
<tr>
<td>SOAP Parser</td>
<td>The file read by the Read File activity</td>
<td>The ESIResponse without SOAP headers</td>
<td>SOAPResponseFinder class</td>
<td>If the SOAP portion of the Response file is not proper, this node will error.</td>
</tr>
<tr>
<td>Parse XML</td>
<td>The parsed schema output from the SOAP Parser</td>
<td>The Windchill ESI Response as an AE object</td>
<td>Windchill Response schema</td>
<td>If the XML inside the SOAP envelope is erroneous, this node will error.</td>
</tr>
<tr>
<td>End</td>
<td>None</td>
<td>None</td>
<td></td>
<td>If this node is reached then the Response schema is proper.</td>
</tr>
</tbody>
</table>

**Table 32 Generator_Validator Components**

### 5.3.3 Stub_ESIEvent_Publication_PD

This is the process that listens for a change to the ESIEventUserA.xml file and kicks off a simulated Windchill Event.

#### 5.3.3.1 Logical Flow

![Flow Diagram](#)

**FilePoll_ESIEvent** → **JMSSender_ESIEvent** → **End StubESIEvent**

#### 5.3.3.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>ESIEventUserA.xml or ESIEventUserB.xml in the ESIDirectory</td>
<td>The contents of the Event file</td>
<td>The Event File</td>
<td>This file poller reads the text content of the file only.</td>
</tr>
<tr>
<td>JMSSender_ESIEvent_Event</td>
<td>The Event file from the File Poller</td>
<td>None</td>
<td></td>
<td>Publishes the content of the Event file to the com.ptc.windchill.esi.Event queue. Also sets the com_infoengine_user parameter to either “UserA” or “UserB”.</td>
</tr>
</tbody>
</table>
This activity publishes the Event to JMS but does not output any information to subsequent TIBCO BusinessWorks activities.

Table 33  Stub_ESIEvent_Publication_PD Components

5.3.4 Stub_ESIRequest_Subscribe_ESIResponse_Publish_PD

This process definition listens for the ESIRequest on the JMS queue `com.ptc.windchill.esi.DataRequest`, reads ESIResponseUserA.xml or ResponseUserB.xml, then publishes that Response on the `com.ptc.windchill.esi.DataResponse` queue. This process treats all XML as a string and performs no validation.

5.3.4.1 Logical Flow

5.3.4.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSReceiver_ESIDataRequest_DataRequest</td>
<td>The ESIRequest sent by the EAI software components via JMS</td>
<td>The contents of the ESIRequest as a text stream</td>
<td>The EAI Windchill Simulation module JMS Connection (/SharedConfigurations/JMS/Stub_JMSConnection)</td>
<td>Receipt of a message on the JMS queue will trigger this process definition</td>
</tr>
<tr>
<td>ReadFile_ESIResponse</td>
<td>The <code>com.infoengine.user</code> parameter from the JMS receiver, which is used to determine whether to read ESIResponseUserA.xml or ESIResponseUserB.xml from the ESIDirectory.</td>
<td>The contents of the ESIResponse file.</td>
<td>The file ESIResponseUserA.xml or ESIResponseUserB.xml exists.</td>
<td>This node simply reads the file.</td>
</tr>
<tr>
<td>JMSSender_ESIDataResponse_DataResponse</td>
<td>The contents of either ESIResponseUserA.xml or ESIResponseUserB.xml.</td>
<td>None.</td>
<td>None</td>
<td>This activity publishes the ESIResponse to JMS but does not output any information to subsequent BusinessWorks activities.</td>
</tr>
</tbody>
</table>

Table 34  Stub_ESIRequest_Subscribe_ESIResponse_Publish_PD Components
5.3.5 Stub_Result_ResultResponse_PD

This process receives ESIPostEvent and ESIPostResult messages from the com.ptc.windchill.esi.Result queue, and if so enabled, saves the contents of the message to a file. It then reads the ESIResultResponse file and sends it back to the EAI software components over JMS.

5.3.5.1 Logical Flow

![Logical Flow Diagram]

5.3.5.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSReceiver_ESIResultorEvent</td>
<td>PostResult and PostEvent messages sent by the EAI software components via JMS</td>
<td>The contents of the PostResult or PostEvent as a text stream</td>
<td>The Stub JMS Connection (/SharedConfigurations/JMS/Stub_JMSConnection)</td>
<td>Receipt of a message on the JMS queue will trigger this process definition.</td>
</tr>
<tr>
<td>Null_DeterminePostResultType</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>This Null allows simplification of the logical branching in this process.</td>
</tr>
<tr>
<td>Parse_ESIPostResult</td>
<td>The contents of the JMS message received by the JMS receiver process starter.</td>
<td>A PostResult file as an AE object.</td>
<td>The ESIPostResult schema.</td>
<td>The ESIPostResult file must be parsed in order to extract the TransactionID with which to generate the filename.</td>
</tr>
<tr>
<td>Parse_ESIPostEvent</td>
<td>The contents of the JMS message received by the JMS receiver process starter.</td>
<td>A PostEvent file as an AE object.</td>
<td>The ESIPostEvent schema.</td>
<td>The ESIPostEvent file must be parsed in order to extract the TransactionID with which to generate the filename.</td>
</tr>
<tr>
<td>Activity Name</td>
<td>Input</td>
<td>Output</td>
<td>Dependencies</td>
<td>Comment</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------</td>
<td>---------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WriteFile_StubPostResultAndPostEvent</td>
<td>The PostResult or PostEvent file as raw text from the JMS process starter</td>
<td>A file containing the raw PostResult or PostEvent XML received from the EAI software components</td>
<td>The com_infoengine_user property from JMS as well as the ESIDirectory global variable</td>
<td>This process dynamically generates a file name for the output file and writes the raw text of the JMS body into the file.</td>
</tr>
<tr>
<td>ReadFile_ESIResultResponse</td>
<td>The ESIResultResponse file from the ESIDirectory</td>
<td>The ESIResultResponse file as a string</td>
<td>The ESIResultResponse file must exist in the directory specified.</td>
<td>This file is not parsed.</td>
</tr>
<tr>
<td>JMSSender_ESIResultResponse_ResultResponse</td>
<td>The ESIResultResponse XML schema</td>
<td>None</td>
<td>The EAI Windchill Simulation module JMS Connection (/SharedConfigurations/JMS/Stub_JMSConnection), as well as a properly read ESIResultResponse file.</td>
<td>This activity publishes the ESIResultResponse to JMS but does not output any information to subsequent BusinessWorks activities.</td>
</tr>
<tr>
<td>End_StubResultResultResponse</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Branching details are as follows:

<table>
<thead>
<tr>
<th>Branch From Activity</th>
<th>Condition</th>
<th>If True, Branch To</th>
<th>If False, Branch To</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSReceiver_ESIResultEvent</td>
<td>If the global variable ESISTUBWriteResultsTo File is set to ESICONSTANT_TRUE</td>
<td>Null_DeterminePostResultType</td>
<td>ReadFile_ESIResultResponse</td>
<td>This allows configurability of the ESIPostResult and ESIPostEvent files. If the flag is set to false then the part of this process definition that parses and writes the Result file is skipped altogether.</td>
</tr>
<tr>
<td>Null_DeterminePostResultType</td>
<td>If the com_infoengine_SOAP_method of the JMS message received in the process starter is set to the global variable ESIPostEventMethod</td>
<td>Parse_ESIPostEvent</td>
<td>Parse_ESIPostEvent</td>
<td>This branch is used to determine which parser to use.</td>
</tr>
</tbody>
</table>

### 5.4 Windchill Simulation Module Appendix

#### 5.4.1 Sample XML Files

Copies of the sample XML files described below can be found in the Windchill ESI CDs.

#### 5.4.1.1 Windchill ESI Event

This is the file that simulates the Event generated by Windchill. It is read by the EAI Windchill Simulation module as a text file and sent over JMS without any validation.
5.4.2 Windchill ESI Response: GetPart

The Windchill Simulation Module uses the ESIResponseUserA/UserB.xml files to mimic Windchill's Response to a request from the EAI Software Components - this is the raw, unparsed XML. This file contains the content of what will be returned in the Body of the JMS message on the com.ptc.windchill.esi.DataResponse queue.

The first few elements of the file are the required SOAP elements. The data is contained within the first Collection element. The &gt; corresponds to the "greater than" symbol (>) and &lt; corresponds to the "less than" symbol (<). This notation is due to how Info*Engine processes the XML before it is sent to the EAI software components. Info*Engine does not trust that the contents of the XML will not contain any greater-than or less-than symbols. Thus, it writes XML tags without greater-than or less-than symbols, instead using their corresponding HTML escape codes. The EAI software components compensate for this upon receipt of the XML, by using a custom JAX-M parser to extract and parse the data.

5.4.2.1 Simple Part (no Documents, BOMs or ECNs)

The following XML file creates a part called XYZ_PRT_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200.
<SOAP-ENV:Body>
<COLLECTION>
<DeletedECN NAME="Deleted" TYPE="Object" STATUS="0">
</DeletedECN>
<AddedECN NAME="Added" TYPE="Object" STATUS="0">
</AddedECN>
<ChangedECN NAME="Changed" TYPE="Object" STATUS="0">
</ChangedECN>
<UnchangedECN NAME="Unchanged" TYPE="Object" STATUS="0">
</UnchangedECN>
<DeletedParts NAME="Deleted" TYPE="Object" STATUS="0">
</DeletedParts>
<AddedParts NAME="Added" TYPE="Object" STATUS="0">
<Part>
<ObjectID>OR:wt.part.WTPartMaster:1124792:117476152-1014739000750-6363996-137-8-253-132@ralexander.mn.ptc.com</ObjectID>
<Class>com.ptc.windchill.esi.Part</Class>
<LastChangedBy></LastChangedBy>
<Number>XYZ_PRT_V01</Number>
<StartEffectivity></StartEffectivity>
<EndEffectivity></EndEffectivity>
<DefaultUnit>ea</DefaultUnit>
>Name>test create regression</Name>
<PartType>inseparable</PartType>
<Source>make</Source>
-Version></Version>
<Iteration></Iteration>
<TargetID>sap.46c:dv1.500:1100</TargetID>
</Part>
</AddedParts>
<ChangedParts NAME="Changed" TYPE="Object" STATUS="0">
</ChangedParts>
<UnchangedParts NAME="Unchanged" TYPE="Object" STATUS="0">
</UnchangedParts>
</COLLECTION>
</ns1:GetPartResponse>
5.4.2.2 ESIResponse : GetPart containing one Part with Document and Document Link:

The Windchill Simulation Module uses the ESIResponseUserA/UserB.xml files to mimic Windchill's Response to a request from the EAI Software Components - this is the raw, unparsed XML. This file contains the content of what will be returned in the Body of the JMS message on the com.ptc.windchill.esi.DataResponse queue.

The first few elements of the file are the required SOAP elements. The data is contained within the first Collection element. The &gt; corresponds to the "greater than" symbol (>) and &lt; corresponds to the "less than" symbol (<). This notation is due to how Info*Engine processes the XML before it is sent to the EAI software components. Info*Engine does not trust that the contents of the XML will not contain any greater-than or less-than symbols. Thus, it writes XML tags without greater-than or less-than symbols, instead using their corresponding HTML escape codes. The EAI software components compensate for this upon receipt of the XML, by using a custom JAX-M parser to extract and parse the data.

The following XML file creates a part called XYZ_PRT_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200. It then creates a document called XYZ_DOC_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and
Finally, it creates two document links linking the part XYZ_PRT_V01 to the document XYZ_DOC_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200.

  &lt;DeletedECN NAME=&quot;Deleted&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/DeletedECN&gt;
  &lt;AddedECN NAME=&quot;Added&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/AddedECN&gt;
  &lt;ChangedECN NAME=&quot;Changed&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/ChangedECN&gt;
  &lt;UnchangedECN NAME=&quot;Unchanged&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/UnchangedECN&gt;
  &lt;DeletedParts NAME=&quot;Deleted&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/DeletedParts&gt;
  &lt;AddedParts NAME=&quot;Added&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/AddedParts&gt;
  &lt;ChangedParts NAME=&quot;Changed&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/ChangedParts&gt;
  &lt;UnchangedParts NAME=&quot;Unchanged&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/UnchangedParts&gt;
&lt;/COLLECTION&gt;&lt;Part&gt;&lt;ObjectID&gt;OR:wt.part.WTPartMaster:1124792:117476152-101473900750-6363996-137-8-253-132@ralexander.mn.ptc.com&lt;/ObjectID&gt;
&lt;Class&gt;com.ptc.windchill.esi.Part&lt;/Class&gt;
&lt;LastChangedBy&gt;&lt;/LastChangedBy&gt;
&lt;Number&gt;XYZ_PRTDOC_V01&lt;/Number&gt;
&lt;StartEffectivity&gt;&lt;/StartEffectivity&gt;
&lt;EndEffectivity&gt;&lt;/EndEffectivity&gt;
&lt;DefaultUnit&gt;ea&lt;/DefaultUnit&gt;
&lt;Name&gt;test reg with doc&lt;/Name&gt;
&lt;PartType&gt;inseparable&lt;/PartType&gt;
&lt;Source&gt;make&lt;/Source&gt;
&lt;Version&gt;&lt;/Version&gt;
5.4.3 ESIResponse: GetBOM

The Windchill Simulation Module uses the ESIResponseUserA/UserB.xml files to mimic Windchill's Response to a request from the EAI Software Components - this is the raw, unparsed XML. This file contains the content of what will be returned in the Body of the JMS message on the com.ptc.windchill.esi.DataResponse queue.

The first few elements of the file are the required SOAP elements. The data is contained within the first Collection element. The &gt; corresponds to the "greater than" symbol (>) and &lt; corresponds to the "less than" symbol (<). This notation is due to how Info*Engine processes the XML before it is sent to the EAI software components. Info*Engine does not trust that the contents of the XML do not contain any greater-than or less-than symbols. Thus, it writes XML tags without greater-than or less-than symbols, instead using their corresponding HTML escape codes. The EAI software components compensate for this upon receipt of the XML, by using a custom JAX-M parser to extract and parse the data.

The following XML file creates two parts, called XYZ_BOM_V01 and XYZ_CHL_V01, each in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200. It then transforms XYZ_BOM_V01 into a BOM in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200, using the Create BOM functionality, and adds XYZ_CHL_V01 as a component to that BOM in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200.

<?xml version='1.0' encoding='UTF-8'?>
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
xmlns:ns3="com.ptc.windchill.esi">
<return xsi:type="xsd:string">
&l;COLLECTION&g;
&l;DeletedParts NAME="Deleted" TYPE="Object" STATUS="0">
&l;AddedParts NAME="Added" TYPE="Object" STATUS="0">
&l;Part&g;
&l;ObjectID&g;=OR:wt.part.WTPartMaster:1124792:117476152-1014739000750-6363996-137-8-253-132@ralexander.mn.ptc.com&l;/ObjectID&g;
&l;Class&g;=com.ptc.windchill.esi.Part&l;/Class&g;
&l;LastChangedBy&g;=&l;/LastChangedBy&g;
&l;Number&g;=XYZ_BOM_V01&l;/Number&g;
&l;StartEffectivity&g;=&l;/StartEffectivity&g;
&l;EndEffectivity&g;=&l;/EndEffectivity&g;
&l;DefaultUnit&g;=ea&l;/DefaultUnit&g;
&l;Name&g;=TEST REG&l;/Name&g;
&l;PartType&g;=component&l;/PartType&g;
&l;Source&g;=make&l;/Source&g;
&l;Version&g;=&l;/Version&g;
&l;Iteration&g;=&l;/Iteration&g;
&l;TargetID&g;=sap.46c:dv1.500:1100&l;/TargetID&g;
&l;/TargetID&g;=sap.46c:dv1.500:1200&l;/TargetID&g;
&l;/Part&g;
&l;/Part&g;
</AddedParts>
</DeletedParts>
</COLLECTION>
</ns1:GetBOMResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
5.4.4 ESIResponse: GetECN

The Windchill Simulation Module uses the ESIResponseUserA/UserB.xml files to mimic Windchill's Response to a request from the EAI Software Components - this is the raw, unparsed XML. This file contains the content of what will be returned in the Body of the JMS message on the com.ptc.windchill.esi.DataResponse queue.

The first few elements of the file are the required SOAP elements. The data is contained within the first Collection element. The &gt; corresponds to the "greater than" symbol (>) and &lt; corresponds to the "less than" symbol (<). This notation is due to how Info*Engine processes the XML before it is sent to the EAI software components. Info*Engine does not trust that the contents of the XML do not contain any greater-than or less-than symbols. Thus, it writes XML tags without greater-than or less-than symbols, instead using their corresponding HTML escape codes. The EAI software components compensate for this upon receipt of the XML, by using a custom JAX-M parser to extract and parse the data.

The following XML file creates an ECN called XYZ_ECN_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200. It then creates a part called XYZ_PRTECN_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200. It then creates a document called XYZ_DOCECN_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200. Finally, it creates two document links linking the part XYZ_PRTECN_V01 to the document XYZ_DOCECN_V01 in two ERP distribution targets, sap.46c:dv1.500:1100 and sap.46c:dv1.500:1200.

```xml
<?xml version='1.0' encoding='UTF-8'?>
     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:GetECNResponse xmlns:ns1="ie-soapRPC-response"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <return xsi:type="xsd:string">
                &lt;COLLECTION&gt;
                &lt;DeletedECN NAME=&quot;Deleted&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;
                &lt;/DeletedECN&gt;
                &lt;AddedECN NAME=&quot;Added&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;
                &lt;/AddedECN&gt;
                &lt;/COLLECTION&gt;
            </return>
        </ns1:GetECNResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
&lt;TargetID&gt;sap.46c:dv1.500:1100&lt;/TargetID&gt;
&lt;TargetID&gt;sap.46c:dv1.500:1200&lt;/TargetID&gt;
&lt;/ECNHeader&gt;
&lt;/AddedECN&gt;
&lt;ChangedECN NAME="Changed" TYPE="Object" STATUS="0">
&lt;/ChangedECN&gt;
&lt;UnchangedECN NAME="Unchanged" TYPE="Object" STATUS="0">
&lt;/UnchangedECN&gt;
&lt;DeletedParts NAME="Deleted" TYPE="Object" STATUS="0">
&lt;/DeletedParts&gt;
&lt;AddedParts NAME="Added" TYPE="Object" STATUS="0">
&lt;Part&gt;
&lt;ObjectID&gt;OR:wt.part.WTPartMaster:1124792:117476152-1014739000750-6363996-137-8-253-132@ralexander.mn.ptc.com&lt;/ObjectID&gt;
&lt;Class&gt;com.ptc.windchill.esi.Part&lt;/Class&gt;
&lt;LastChangedBy&gt;&lt;/LastChangedBy&gt;
&lt;Number&gt;XYZ_PRTECN_V01&lt;/Number&gt;
&lt;StartEffectivity&gt;2004-04-04 00:00:00&lt;/StartEffectivity&gt;
&lt;EndEffectivity&gt;&lt;/EndEffectivity&gt;
&lt;DefaultUnit&gt;ea&lt;/DefaultUnit&gt;
&lt;Name&gt;LALALA&lt;/Name&gt;
&lt;PartType&gt;inseparable&lt;/PartType&gt;
&lt;Source&gt;make&lt;/Source&gt;
&lt;Version&gt;1&lt;/Version&gt;
&lt;Iteration&gt;&lt;/Iteration&gt;
&lt;/Part&gt;
&lt;/AddedParts&gt;
&lt;ChangedParts NAME="Changed" TYPE="Object" STATUS="0">
&lt;/ChangedParts&gt;
&lt;UnchangedParts NAME="Unchanged" TYPE="Object" STATUS="0">
&lt;/UnchangedParts&gt;
&lt;DeletedDocuments NAME="Deleted" TYPE="Object" STATUS="0">
&lt;/DeletedDocuments&gt;
&lt;AddedDocuments NAME="Added" TYPE="Object" STATUS="0">
&lt;/AddedDocuments&gt;
&lt;/Part&gt;
&lt;/ChangedParts&gt;
&lt;/UnchangedParts&gt;
&lt;/DeletedDocuments&gt;
&lt;/AddedDocuments&gt;
&lt;/Part&gt;
&lt;/ChangedParts&gt;
&lt;/UnchangedParts&gt;
&lt;/DeletedDocuments&gt;
&lt;/AddedDocuments&gt;
&lt;/Part&gt;
&lt;/ChangedParts&gt;
&lt;/UnchangedParts&gt;
&lt;/DeletedDocuments&gt;
&lt;/AddedDocuments&gt;
&lt;/Part&gt;
&lt;/ChangedParts&gt;
&lt;/UnchangedParts&gt;
&lt;/DeletedDocuments&gt;
&lt;/AddedDocuments&gt;
&lt;/Part&gt;

&lt;ObjectID&gt;OR:wt.part.WTPartMaster:1124792:117476152-1014739000750-6363996-137-8-253-132@ralexander.mn.ptc.com&lt;/ObjectID&gt;
&lt;Class&gt;com.ptc.windchill.esi.Document&lt;/Class&gt;
&lt;LastChangedBy&gt;&lt;/LastChangedBy&gt;
&lt;Number&gt;XYZ_DOCECN_V01&lt;/Number&gt;
&lt;Name&gt;Regression Doc&lt;/Name&gt;
&lt;DocumentType&gt;WCTYPE|wt.doc.WTDocument$$Requirements Document&lt;/DocumentType&gt;
&lt;Description&gt;TEST CHANGES&lt;/Description&gt;
&lt;File&gt;HTTP://WWW.PTC.COM&lt;/File&gt;
&lt;Title&gt;TEST&lt;/Title&gt;
&lt;Version&gt;A&lt;/Version&gt;
&lt;Iteration&gt;&lt;/Iteration&gt;
&lt;PreviousVersion&gt;A&lt;/PreviousVersion&gt;
&lt;TargetID&gt;sap.46c:dv1.500:1100&lt;/TargetID&gt;
&lt;TargetID&gt;sap.46c:dv1.500:1200&lt;/TargetID&gt;
&lt;/Document&gt;
&lt;/AddedDocuments&gt;
&lt;ChangedDocuments NAME=&quot;Changed&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/ChangedDocuments&gt;
&lt;UnchangedDocuments NAME=&quot;Unchanged&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/UnchangedDocuments&gt;
&lt;DeletedDocumentLinks NAME=&quot;Deleted&quot; TYPE=&quot;Object&quot; STATUS=&quot;0&quot;&gt;&lt;/DeletedDocumentLinks&gt;
&lt;/DeletedDocumentLinks&gt;
&lt;/AddedDocumentLinks&gt;
&lt;/DeletedBOMs&gt;
5.4.5 ESIResultResponse

This is a file that simulates the ESIResultResponse generated by Windchill. It is read by the EAI Windchill Simulation module as a text file and sent over JMS without any validation.

The first few elements of the file are the required SOAP elements. The data is contained within the first Collection element. The &gt; corresponds to the "greater than" symbol (>) and &lt; corresponds to the "less than" symbol (<). This notation is due to how Info*Engine processes the XML before it is sent to the EAI software components. Info*Engine does not trust that the contents of the XML do not contain any greater-than or less-than symbols. Thus, it writes XML tags without greater-than or less-than symbols, instead using their corresponding HTML escape codes. The EAI software components compensate for this upon receipt of the XML, by using a custom JAX-M parser to extract and parse the data.
6 EAI Automated Test Fixture

6.1 Overview

6.1.1 Purpose

The purpose of the Windchill ESI EAI Automated Test Fixture is to give an end-user a mechanism for automating testing using the EAI Windchill Simulation module to mimic the Windchill Open API (that is, the Windchill ESI to EAI Software Components Interface) within TIBCO BusinessWorks. This automated test fixture enables users to quickly run tests against the ESI 7.0 EAI solution without having to manually invoke individual tests. This module also provides a tool for quickly regression testing EAI Software Components to quickly identify issues within the weekly build prior to its release. It is expected that implementation teams will leverage the module in the same way as the EAI Software Components Development Team.

The subsequent pages of this document provide an overview of the Windchill ESI EAI Automated Test Fixture as well as instructions and examples for using the module.

6.1.2 What is Automated

The Windchill ESI EAI Automated Test Fixture is designed to automate tests being run against the Windchill ESI EAI Software Components solution. It does not include automated comparison or validation of data or messages coming from the EAI Software Components.
The Winchill ESI EAI Automated Test Fixture requires the use of the ESI EAI Windchill Simulation Module. Please see the EAI Windchill Simulation Module Section for more information about the setup and use of the module.

The Winchill ESI EAI Automated Test Fixture provides a mechanism to automatically run multiple tests against the Windchill ESI EAI Software Components solution without manual intervention. The module utilizes the EAI Windchill Simulation Module to emulate the sending and receiving of JMS messages by Windchill via the Windchill Open API. The module invokes the EAI Software Components solution to perform the standard operations as defined by the Winchill Enterprise Systems Integration User’s Guide.

6.2 User’s Guide

6.2.1 Repository Configuration

The EAI Automated Test Fixture has its own repository, called ESISimulationWindchillRepository_vxxx. This repository contains the functionality for the Windchill Simulation as well as automated testing and exists entirely separately from the ESIMasterRepository.

6.2.2 Setting up Data and File Structure

The EAI Automated Test Fixture uses two directories in the directory identified by the global variable ESIDirectory as a staging area for the test data to be passed into the test fixture. It is necessary to create these new directories when running the EAI Automated Test Fixture. The first new directory that is required in the ESIDirectory is "ESIEventHOME".

**Note:** It is critical that the spelling and case-sensitivity for the name of the directory are identical to that listed. The purpose of this directory is to store ESIFile files for each test that is to be run; therefore, there must be at least one file for each test that is to be run in the EAI Automated Test Fixture. Within the ESIEventHOME directory, each ESIFile file MUST be named with the following notation:

ESIEvent_<TestNumber>.xml

where TestNumber is the test case number (any integer, the sequence of test numbers must start with 1 and be in order)

The second new directory that is required in the ESIDirectory is "ESIResponseHOME". Note: It is critical that the spelling and case-sensitivity for the name of the directory are identical to that listed above. The purpose of this directory is to store ESIResponse files for each test that will be run. Each ESIResponse file should contain the test data required for each test. Therefore, there must be at least one file for each test that is to be run in the EAI Automated Test Fixture.

Within the ESIResponseHOME directory, each ESIResponse file must be named with the following notation:

ESIResponse_<TestNumber>.xml

where TestNumber is the test case number (any integer, the sequence of test numbers must start with 1 and be in order)
The EAI Automated Test Fixture will continue to use the files required by the EAI Windchill Simulation module. See the EAI Windchill Simulation Module Section for more information about these files.

6.2.3 XML Files

The XML files required for the EAI Automated Test Fixture are the same as those required by the EAI Windchill Simulation Module. See the EAI Windchill Simulation Module Section for a detailed description and sample of each file.

6.2.4 Configuration and Global Variables

The following Global Variables pertain to the Test Fixture process:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>ESI 7.0 Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESIAutoTest_ESIDirectory</td>
<td>The location of the ESI directory.</td>
<td></td>
<td>C:\tibco\esi</td>
</tr>
<tr>
<td>ESIAutoTest_Event_Sleep_Time</td>
<td>Amount of time to wait after the new Event file is copied.</td>
<td>Any integer.</td>
<td>60000</td>
</tr>
<tr>
<td>ESIAutoTest_LastTest_Sleep_Time</td>
<td>Amount of time to wait after the test is completed before resuming.</td>
<td>Any integer.</td>
<td>60000</td>
</tr>
<tr>
<td>ESIAutoTest_TestID</td>
<td>Test Case number of the first test in the sequence.</td>
<td>Any integer.</td>
<td>1</td>
</tr>
<tr>
<td>ESIAutoTest_TotalTests</td>
<td>Test Case number of the last test in the sequence.</td>
<td>Any integer.</td>
<td>10</td>
</tr>
</tbody>
</table>

6.2.5 Logging and Tracing

Out-of-the-box, the EAI Automated Test Fixture does not include any logging or tracing of its activity, and does not affect any tracing or logging within the EAI Software Components. In order to log or trace the EAI Automated Test Fixture’s activity, the user may leverage the Debug and Logging shared services TIBCO out-of-the-box tools such as the Write to File activity as they desire. For more information on how to use the Debug and Logging shared services, see the Windchill Enterprise Systems Integration Installation and Configuration Guide.

6.2.6 Running the Automated Test Fixture

The following two sections describe different methods of running an instance of the Automated Test Fixture. The Automated Test Fixture works in conjunction with the EAI Software Components main Repository. Before running the Test Fixture, start the Process Engine and the adapters as described in the Windchill Enterprise Systems Integration Installation and Configuration Guide.

6.2.6.1 Running the Automated Test Fixture in Engine Mode

It is recommended to deploy and run a TIBCO BusinessWorks engine to handle all the Windchill Simulation module functions. To deploy an EAI Windchill Simulation Module and Automated Test Engine, follow the same steps to deploy an engine as specified in the
Windchill Enterprise Systems Integration Installation and Configuration Guide, but complete these additional steps:

1. In the Deployment Configuration, select the Process Definitions tab.
2. Uncheck the "Select All" button.
3. Select the following Process Diagrams:
   a. Automated_Regression_Test_Process
   b. Stub_ESIEvent_Publication_PD
   c. Stub_ESIRequest_Subscribe_ESIResponse_Publish_PD
   d. Stub_Result_ResultResponse_PD
4. Deploy the new EAI Windchill Simulation Module and Automated Test Engine.

TIBCO BusinessWorks will generate a .cmd file in c:\tibco\bw\2.0\bin that corresponds to the deployment name specified in the deployment configuration. To run the engine, follow the instructions for starting a Process Engine from the TIBCO Administration GUI in the Windchill Enterprise Systems Integration Installation and Configuration Guide. When this engine is run, it will perform all the functions of the EAI Windchill Simulation module (listening for and publishing ESIEvents, listening for ESIRequests, etc).

6.2.6.2 Running the Automated Test Fixture in Test Mode

It is also possible to run the process in test mode. This allows a user to set breakpoints and observe the EAI Automated Test Fixture using the TIBCO BusinessWorks debugger.
To run the process in Test Mode, complete the following steps:
2. Open the Tester window.
3. Press the double-green arrow button
4. Select the following Process Diagrams:
   - Automated_regression_test_process
   - Stub_ESIEvent_Publication_PD
   - Stub_ESIRequest_Subscribe_ESIResponse_Publish_PD
   - Stub_Result_ResultResponse_PD
5. To start the engine, press **Start**. The process will start and copy the appropriate files to the ESI Directory.

### 6.3 Technical Design

#### 6.3.1 Automated_test_trigger_process

This process definition copies the ESIResponse and ESIEvent files of the particular test number that is going to be run to the ESI Directory.

#### 6.3.1.1 Logical Flow

![Logical Flow Diagram]

#### 6.3.1.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>None</td>
<td>Test_Number which is passed from the Automated_regression_test_processes as the Test_Number_Index. The Test_Number_Index defines the test that will be run.</td>
<td>There must be ESIResponse and ESIEvent files with the Test_Number listed for the process to work.</td>
<td>None.</td>
</tr>
<tr>
<td>Activity Name</td>
<td>Input</td>
<td>Output</td>
<td>Dependencies</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>--------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>Wait_for_Last_Test_to_Complete</td>
<td>Global variable - ESIAutoTest_LastTest_Sleep_Time</td>
<td>None</td>
<td>None</td>
<td>This activity waits to make sure that last test has completed.</td>
</tr>
<tr>
<td>Read_Relevant_ESIResponse_File</td>
<td>The name of the ESIResponse file that will be read in. The name will be defined at ESIResponse_&lt;Test_Number&gt; where Test_Number is the number of the test that is being run. File encoding : UTF-8</td>
<td>File content of ESIResponse_&lt;Test_Number&gt; is read into memory.</td>
<td>There must be ESIResponse file with the Test_Number listed for the process to work.</td>
<td>This activity reads the relevant ESIResponse file in a UTF-8 encoded format.</td>
</tr>
<tr>
<td>Rename_ESIResponse_File_To_ESIResponseUserA.xml</td>
<td>The name of the ESIResponse file that will be read in. The name will be defined at ESIResponse_&lt;Test_Number&gt; where Test_Number is the number of the test that is being run. File encoding : UTF-8 The file content from the previous activity.</td>
<td>New file written to the ESI Directory with the name ESIResponseUserA.xml</td>
<td>The previous activity must have run successfully.</td>
<td>This activity writes the relevant ESIResponse file into the ESI Directory named as ESIResponse_UserA.xml and UTF-8 encoded.</td>
</tr>
<tr>
<td>Read_ESIEventUserA_XML_File_From_ESIEventHOME_Directory</td>
<td>The name of the ESIEvent file that will be read in. The name will be defined at ESIEvent_&lt;Test_Number&gt; where Test_Number is the number of the test that is being run. File encoding : ANSI</td>
<td>File content of ESIEvent_&lt;Test_Number&gt; is read into memory.</td>
<td>There must be ESIEvent file with the Test_Number listed for the process to work.</td>
<td>This activity reads the relevant ESIEvent file in an ANSI encoded format.</td>
</tr>
<tr>
<td>Copy_ESIEventUserA_XML_File_To_ESI_HOME_Directory</td>
<td>The name of the ESIEvent file that will be read in. The name will be defined at ESIEvent_&lt;Test_Number&gt; where Test_Number is the number of the test that is being run. File encoding : ANSI The file content from the previous activity.</td>
<td>New file written to the ESI Directory with the name ESIEvent.xml</td>
<td>The previous activity must have run successfully.</td>
<td>This activity writes the relevant ESIEvent file into the ESI Directory named as ESIEvent.xml and ANSI encoded.</td>
</tr>
<tr>
<td>Wait_for_Event_to_be_Copied</td>
<td>Global variable - ESIAutoTest_Event_Sleep_Time</td>
<td>None</td>
<td>None</td>
<td>This activity waits to ensure the ESIEvent has been copied.</td>
</tr>
<tr>
<td>End</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

### 6.3.2 Automated_Regression_Test_Process

This process loops through each test number to process each test by calling the automated test trigger process.
6.3.2.1 Logical Flow

6.3.2.2 Components

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>This timer starts the automated test process.</td>
</tr>
<tr>
<td>Iterate_Total_Number_Regression_Tests</td>
<td>This iterator starts at 1 and continues to increment to the number defined by ESIAutoTest_TotalTests.</td>
<td>Test_Number_Index that defines the test that is going to be run.</td>
<td>There must be ESIResponse and ESIEvent files with the Test_Number_Index listed for the process to work.</td>
<td>None</td>
</tr>
<tr>
<td>Call_Automated_Test_Trigger_Process</td>
<td>Test_Number_Index that defines the test that is going to be run.</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

7 SAP R/3 Readiness Validation Process

7.1 Overview

7.1.1 Purpose

The purpose of the Windchill Enterprise Systems Integration SAP R/3 Readiness Validation Process is to confirm that a given SAP R/3 distribution target system is correctly configured according to documented assumptions and API characterizations. Each of the component systems of Windchill Enterprise Systems Integration, including SAP R/3, may be subject to an extensive degree of customization and patching before a Windchill Enterprise Systems Integration implementation. This poses an implementation risk when introducing Windchill Enterprise Systems Integration into an existing environment and infrastructure. The SAP R/3 Readiness Validation Process is a toolset for systems integrators to leverage in the early stages of a Windchill Enterprise Systems Integration deployment to help mitigate this risk.

7.1.2 How It Works

The SAP R/3 Readiness Validation Process involves manual validation and testing, as well as use of an API testing application provided with the Windchill Enterprise Systems Integration
EAI Windchill Simulation module. The SAP R/3 Readiness Validation Process consists of six steps. Each step is designed to build upon the previous steps. Completion of each step moves you further along the "Confidence Curve," increasing overall confidence that each of your SAP R/3 distribution target systems will work successfully with Windchill Enterprise Systems Integration. This concept is illustrated below.

Steps 4, 5, and 6 of the SAP R/3 Readiness Validation Process create actual business objects in your SAP R/3 distribution target system based on the data you specify.

Caution: Do not perform these steps of the SAP R/3 Readiness Validation Process with production or business-critical systems. Instead, use development or sandbox systems that replicate the production environment as closely as possible.

7.1.3 Benefits
The SAP R/3 Readiness Validation Process provides the following benefits:

- Confirms that your SAP R/3 Release 4.6C distribution target systems are correctly configured per documented Windchill Enterprise Systems Integration assumptions and API characterizations
• Increases your confidence that Windchill Enterprise Systems Integration will work with a given SAP R/3 distribution target system before actually using the solution
• Allows you to isolate Windchill Enterprise Systems Integration software issues from SAP R/3 issues
• Helps PTC Technical Support to more quickly isolate and resolve issues

7.1.4 Required Skills
Proper use of the SAP R/3 Readiness Validation Process requires the following skills:

TIBCO
• TIBCO BusinessWorks run-time environment
• TIBCO Adapter for R/3 configuration and use

SAP R/3
• Functional module testing within transaction SE37
• Basis/ABAP: Verification of SAP Note and Support Package application
• Functional configuration verification:
  - Materials: Materials Management (MM) Module
  - Material BOMs: Production Planning (PP) Module
  - Documents: Cross-Application (CA) Module
  - Change Masters: Logistics (LO) Module
• Familiarity with key transactions to verify functional test results:
  - MM03: View Material
  - CS03: View Material BOM
  - CV03N: View Document
  - CC03: View Change Master

7.2 The Six Steps
The six steps of the SAP R/3 Readiness Validation Process are detailed below.

7.2.1 Step 1: Complete the Configuration Checklist
The purpose of Step 1 is to confirm that each of your SAP R/3 distribution target systems meets the minimum Windchill Enterprise Systems Integration configuration specifications as documented in the *Windchill Enterprise Systems Integration Installation and Configuration Guide*. Complete the following checklist for each of your SAP R/3 distribution target systems.

Minimum required system configuration:
- SAP R/3 Release 4.6C system
- ABA + R/3 + Basis Support Packages through 39 applied

Confirm that the following individual SAP Notes are applied.

Included in Release 4.6C Support Package 33:

SAP Note #514873: "Locking not released after BAPI_MATERIAL_SAVEDATA"
SAP Note #522308: "Still lockings [sic] after error in BAPI_MATERIAL_SAVEDATA"
SAP Note #523737: "CCAP_REV_LEVEL_MAINTAIN is not remote-enabled"

Included in Release 4.6C Support Package 44:
SAP Note #613818: "Function group FL_WARNING always filled"\(^3\)

Included in Release 4.6C Support Package 45:
SAP Note #650382: "CSAP_MAT_BOM_CREATE doesn't recognize newly created material"
SAP Note #656093: "Error when unlocking: CCAP_REV_LEVEL_MAINTAIN"\(^4\)

RFC User Account:
- "ESISYS" SAP R/3 user account configured as specified

Functional Configuration:
- Documents
- Revision Levels for Materials
- Engineering Change Management
- Materials
- BOMs

Account for any gaps in completing the above checklist and determine how to close them. If you cannot configure your SAP R/3 distribution target systems as specified, you may have to customize Windchill Enterprise Systems Integration.

7.2.2 Step 2: Test TIBCO Adapter for R/3 Connectivity with SAP R/3

The purpose of Step 2 is to prove basic TIBCO Adapter for R/3 network connectivity to the SAP R/3 distribution target system. It is important to note that this step does not validate Windchill Enterprise Systems Integration configuration.

Use the following procedure to complete Step 2.

\(^3\) Some SAP R/3 Basis administrators have reported difficulty applying this OSS note using SAP R/3 utilities. You may have to apply and verify it manually. SAP recommends using the Note Assistant (transaction SNOTE). The only prerequisite for note 613818 is that notes 611271 and 615634 have previously been applied. If you run into problems, create a customer message on the SAP Service Marketplace (http://service.sap.com), specifying the BC-UPG-NA component.

\(^4\) This SAP Note is only required if SAP Note #578909 ("CCAP_REV_LEVEL_MAINTAIN locks only the change number"), included in Release 4.6C Support Package 41, has been applied. SAP Note #656093 corrects an SAP R/3 software defect introduced by SAP Note #578909. SAP Note #578909 is not required by Windchill Enterprise Systems integration, but if it has been applied, you must also apply SAP Note #656093.
1. Per the *Windchill Enterprise Systems Integration Installation and Configuration Guide*, install TIBCO BusinessWorks and the TIBCO Adapter for R/3. You do not have to configure these products yet.

2. Start TIBCO Designer.

3. Open the Enterprise Systems Integration business logic repository (.dat file).

4. Navigate to the "PartConfiguration" resource in the folder path: 
   /AdapterConfigurations/SAP, as shown in the diagram below.

![Image of TIBCO Designer with "PartConfiguration" resource highlighted.](image)

*Figure 71  Accessing the "PartConfiguration" Resource*

5. Select the Design-Time Connection tab.
6. In the Designer configuration panel, overwrite the SAP R/3 connection information with the specifics for your particular distribution target system (Do not save these modifications to the repository!):

- Application Server
- System Number
- Client
- User Name
- Password
- R/3 Logon Language

7. De-select the "Allow Automatic Discovery" checkbox.

8. Click the "Select Design Time Adapter…” pushbutton and select your DTA, as illustrated in Figure 72, below.

![Image of TIBCO Designer interface showing R/3 connection parameters]

Figure 72  Specifying Design-Time Adapter for R/3 Connection Parameters
9. Click "Test Connection".
10. Confirm that you receive a "Connection Success" pop-up window with the host name, SAP R/3 system ID, and R/3 Release, similar to the one illustrated below.

![Figure 73 Adapter for R/3 Test Connection Success Message](image)

If you do not receive this message, double-check your SAP R/3 connection parameters. If they are correct, debug the adapter connection according to the messages TIBCO Designer provides.

11. Repeat steps 4-10 for the remaining Adapter Configuration Resources:

   - BOMConfiguration
   - DocumentConfiguration
   - ECNConfiguration

12. Exit TIBCO Designer (Do not save your modifications to the repository!):

### 7.2.3 Step 3: Validate SAP R/3 API Structures

The purpose of Step 3 is to confirm that your SAP R/3 API structures match those assumed by Windchill Enterprise Systems Integration. You can do this by importing each SAP R/3 API used by the solution into TIBCO Designer and comparing the structure to the baseline, reference Windchill Enterprise Systems Integration structures.

Discrepancies are most likely to exist if you are using custom SAP R/3 APIs or if the SAP R/3 distribution target system does not satisfy the configuration assumptions described in Step 1 of the SAP R/3 Readiness Validation Process. In the actual Enterprise Systems Integration business logic, these discrepancies will likely manifest in the incorrect parsing or mapping of SAP R/3 API fields, since the API schema in TIBCO BusinessWorks will not match the actual API schema in the SAP R/3 distribution target system. In this event, you must re-import the API structures from the SAP R/3 distribution target system into the Enterprise Systems Integration business logic, using a procedure similar to the one described below.

Use the following procedure to validate the SAP R/3 API Structures.

1. Create a new, empty, local repository in TIBCO BusinessWorks Designer.

2. Drag an R3 Adapter Configuration Resource from the TIBCO Designer palette panel into the design panel.
3. In the Designer configuration panel, overwrite the SAP R/3 connection information with the specifics for your particular distribution target system:

- Application Server
- System Number
- Client
- User Name
- Password
- R/3 Logon Language

4. De-select the "Allow Automatic Discovery" checkbox.

5. Click the "Select Design Time Adapter…" pushbutton and select your DTA, as illustrated in Figure 74, below.

![Image of Designer configuration panel showing R/3 connection parameters]

Figure 74  Specifying Design-Time Adapter for R/3 Connection Parameters
6. Click "Apply".

7. Navigate to the Adapter Services folder in the TIBCO Designer project tree panel.

You are now about to import each of the SAP R/3 API structures used by Windchill Enterprise Systems Integration directly from your SAP R/3 distribution target system into TIBCO Designer. The APIs to import are as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>API Function Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BAPI_DOCUMENT_CHANGE2</td>
</tr>
<tr>
<td>2</td>
<td>BAPI_DOCUMENT_CREATE2</td>
</tr>
<tr>
<td>3</td>
<td>BAPI_DOCUMENT_CREATENEWVRS2</td>
</tr>
<tr>
<td>4</td>
<td>BAPI_MATERIAL_SAVEDATA</td>
</tr>
<tr>
<td>5</td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>6</td>
<td>BAPI_TRANSACTION_ROLLBACK</td>
</tr>
<tr>
<td>7</td>
<td>CALO_INIT_API</td>
</tr>
<tr>
<td>8</td>
<td>CCAP_ECN_CREATE</td>
</tr>
<tr>
<td>9</td>
<td>CCAP_ECN_MAINTAIN</td>
</tr>
<tr>
<td>10</td>
<td>CCAP_REV_LEVEL_MAINTAIN</td>
</tr>
<tr>
<td>11</td>
<td>CSAP_MAT_BOM_CREATE</td>
</tr>
<tr>
<td>12</td>
<td>CSAP_MAT_BOM_MAINTAIN</td>
</tr>
</tbody>
</table>

Table 36  SAP R/3 APIs Used by Windchill Enterprise Systems Integration

If you have modified any of these standard SAP R/3 APIs, you will likely want to fetch the enhanced version(s). If you are unsure whether such modifications have occurred, consult your ABAP programming team and try searching for APIs beginning with the letter "Y" or "Z." Later, you will have to customize the Enterprise Systems Integration business logic repository to utilize the enhanced API(s).

If you have any difficulty with the following steps, refer to the section entitled, "Adding Adapter Services Automatically," in the TIBCO Adapter for R/3 User’s Guide.

8. In the TIBCO Designer configuration panel, fetch each of the SAP R/3 APIs listed above, using suitable wildcards, such as BAPI_DOCUMENT*, CCAP*, etc., in the RFC Filter field and clicking the "Fetch RFC/BAPIs" button, as illustrated below.
Figure 75  Specifying an RFC Filter for Fetching

9. Go to the tab labeled "RFC/BAPI."

10. Expand each relevant API to display its structure. To do this, you must click once on the API name, wait for TIBCO to fetch it from the SAP R/3 distribution target system, and then click the "+" sign next to the API name to reveal its structure, as illustrated below.
11. Compare the actual, imported structure of each API with the API structures and field descriptions provided in the "SAP R/3 API Structures" appendix of the Windchill Enterprise Systems Integration Installation and Configuration Guide. Note that this appendix lists only those API fields that are expected and used by the out-of-the-box Windchill Enterprise Systems Integration solution. The actual, imported API structures will contain many other unused fields; this is to be expected.

You must account for any additional discrepancies, such as missing APIs or missing API fields required by Windchill Enterprise Systems Integration, to ensure the correct operation of the solution. Resolving discrepancies may require:

- Upgrading the SAP R/3 distribution target system to a Release supported by Windchill Enterprise Systems Integration
- Application of SAP Notes or Support Packages to the SAP R/3 distribution target system
- Custom API development in the SAP R/3 distribution target system
7.2.4 Step 4: Validate API Atomic Functionality Inside SAP R/3

The purpose of Step 4 is to isolate any SAP R/3-based issues with individual, or "atomic," APIs used by Windchill Enterprise Systems Integration. Step 4 is completed entirely within the SAP R/3 distribution target system. Finding problems at this step can streamline troubleshooting by ruling out the EAI software components as root causes.

This step of the SAP R/3 Readiness Validation Process will create actual business objects in your SAP R/3 distribution target system based on the data you specify.

Caution: Do not perform these steps with production or business-critical systems. Instead, use development or sandbox systems that replicate the production environment as closely as possible.

Steps 4 and 5 leverage a standard SAP transaction, SE37, used for developing, testing, and analyzing ABAP function modules. This transaction has several testing features you may wish to take advantage of that are not discussed in this document, for example, saving test data for future use and regression testing. Refer to SAP R/3 online application help for further details on using the advanced features of this transaction.

To complete Step 4, use the following procedure.

1. Log on to the SAP R/3 distribution target system.
2. Go to the Function Builder Initial Screen (transaction SE37).
3. For each Windchill Enterprise Systems Integration SAP R/3 API (per Table 36):
   a. Enter the name of the API function module in the Function module field, as illustrated below for BAPI_MATERIAL_SAVEDATA.
b. In the SAP R/3 menu bar, navigate to Function module -> Test -> Single test (F8), or click the "Single Test" icon. A screen like the one below appears.

Figure 77  SAP R/3 Function Builder Initial Screen (Transaction SE37)
c. Provide suitable test data in the API fields by clicking on the icon next to each API structure, entering the data, and using the SAP R/3 "Back (F3)" icon to return to the Function Module Tester initial screen, as illustrated.
Your test data conditions depend on your particular Windchill Enterprise Systems Integration deployment. You must hard-code field values that are normally defaulted or cross-referenced by the actual Windchill Enterprise Systems Integration business logic. For guidance, refer to the following documentation:

- The "Functional Overview" section of the *Windchill Enterprise Systems Integration User’s Guide*
- The "SAP R/3 API Description/Characterization" section of this document
- The "SAP R/3 API Structures" appendix of the *Windchill Enterprise Systems Integration Installation and Configuration Guide*

Because you are only testing individual, or "atomic," APIs, your test conditions will not cover complete Windchill Enterprise Integration business scenarios, in...
which different APIs are invoked sequentially to complete a business function. This is "molecular" testing, which you will complete in Step 5 of the SAP R/3 Readiness Validation Process. The purpose of Step 4 is to ensure that each, atomic API is behaving as expected prior to combining APIs for molecular functionality.

d. In the SAP R/3 menu bar, navigate to Function modules -> Execute (F8), or click the "Execute" icon to run the test.

e. Confirm the successful execution of the API function module according to the "SAP R/3 API Response Characterization" section of this document. Any errors are likely due to invalid test data or your SAP R/3 distribution target system configuration. These issues must be resolved prior to implementing Windchill Enterprise Systems Integration.

An example of a successful test of BAPI_MATERIAL_SAVEDATA is shown below (notice that the RETURN structure indicates a message ID of ‘MM’ and a message NUMBER of ‘356’).
f. If applicable, verify that the business object corresponding to the data you specified exists in the SAP R/3 system with the correct attributes, using the transactions indicated below.

<table>
<thead>
<tr>
<th>Transaction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM03</td>
<td>View Material</td>
</tr>
<tr>
<td>CS03</td>
<td>View Material BOM</td>
</tr>
<tr>
<td>CV03N</td>
<td>View Document</td>
</tr>
<tr>
<td>CC03</td>
<td>View Change Master</td>
</tr>
</tbody>
</table>

Table 37  SAP R/3 Transaction Codes to Verify Business Object Creation

Several API function modules, especially those originally developed after SAP R/3 Release 3.1, do not contain a built-in ABAP command to commit the results to the SAP R/3 database. Hence, you are not be able to view the associated business object until you complete the next logical unit of work or exit from transaction SE37. Other APIs, including CALO_INIT_API, BAPI_TRANSACTION_COMMIT, and BAPI_TRANSACTION_ROLLBACK, do not produce observable business objects, by design.

g. Repeat this procedure for each Windchill Enterprise Systems Integration SAP R/3 API (per Table 36).

7.2.5 Step 5: Validate API Molecular Functionality Inside SAP R/3

The purpose of Step 5 is to isolate any SAP R/3-based issues with "molecular" sequences of APIs, exactly as they are invoked by Windchill Enterprise Systems Integration to complete business functions. Step 5 is completed entirely within the SAP R/3 distribution target system. Finding problems at this step can streamline troubleshooting by ruling out the EAI software components as root causes.

This step of the SAP R/3 Readiness Validation Process will create actual business objects in your SAP R/3 distribution target system based on the data you specify.

Caution: Do not perform these steps with production or business-critical systems. Instead, use development or sandbox systems that replicate the production environment as closely as possible.
Steps 4 and 5 leverage a standard SAP transaction, SE37, used for developing, testing, and analyzing ABAP function modules. This transaction has several testing features you may wish to take advantage of that are not discussed in this document, for example, saving test data for future use and regression testing. Refer to SAP R/3 online application help for further details on using the advanced features of this transaction.

To complete Step 5, use the following procedure.

1. Log on to the SAP R/3 distribution target system.
2. Go to the Function Builder Initial Screen (transaction SE37).
3. For each Windchill Enterprise Systems Integration Business Function Test Scenario (per Table 38):

<table>
<thead>
<tr>
<th>Business Function Test Scenario</th>
<th>SAP R/3 APIs Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Part</td>
<td>BAPI_MATERIAL_SAVEDATA</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>Create Part Revision</td>
<td>CCAP_REV_LEVEL_MAINTAIN</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>Change Part</td>
<td>BAPI_MATERIAL_SAVEDATA</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>Create BOM</td>
<td>CALO_INIT_API</td>
</tr>
<tr>
<td></td>
<td>CSAP_MAT_BOM_CREATE</td>
</tr>
<tr>
<td>Change BOM</td>
<td>CALO_INIT_API</td>
</tr>
<tr>
<td></td>
<td>CSAP_MAT_BOM_MAINTAIN</td>
</tr>
<tr>
<td>Create Document</td>
<td>BAPI_DOCUMENT_CREATE2</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>Create Document Revision plus</td>
<td>BAPI_DOCUMENT_CREATENEWVRS2</td>
</tr>
<tr>
<td>Change Document plus, optionally,</td>
<td>BAPI_TRANSACTION_COMMIT</td>
</tr>
<tr>
<td>Add or Delete Document Links</td>
<td>BAPI_DOCUMENT_CHANGE2</td>
</tr>
<tr>
<td>Create ECN</td>
<td>CCAP_ECN_CREATE</td>
</tr>
<tr>
<td>Change ECN</td>
<td>CCAP_ECN_MAINTAIN</td>
</tr>
</tbody>
</table>

Table 38   SAP R/3 APIs for each Business Function Test Scenario

a. Enter the name of any valid API function module in the Function module field. For example, you may enter "BAPI_MATERIAL_SAVEDATA" in this field, as shown in Figure 77. The API function module you choose is unimportant, as long as it is valid.
b. In the SAP R/3 menu bar, navigate to Function module -> Test -> Test Sequences (Shift + F8).

c. In the dialog window that appears, enter the sequence of APIs required to execute the business function test scenario, according to Table 38. The order of the APIs is very important. For example, for the "Create Part" scenario, you should enter "BAPI_MATERIAL_SAVEDATA," followed by "BAPI_TRANSACTION_COMMIT," as illustrated.

![Figure 81 Entering API Test Sequence for "Create Part" Scenario](image)

d. Click the "Execute (Enter)" icon.

e. Provide suitable test data in the API fields by clicking on the icon next to each API structure, entering the data, and using the SAP R/3 "Back (F3)" icon to return to the Function Module Tester initial screen, just as you did in Step 4 of the SAP R/3 Readiness Validation Process (see Figure 79).
Your test data conditions will depend on your particular Windchill Enterprise Systems Integration deployment. You must hard-code field values that are normally defaulted or cross-referenced by the actual Enterprise Systems Integration business logic.
For guidance, refer to the following documentation:

- The "Functional Overview" section of the Windchill Enterprise Systems Integration User's Guide
- The "SAP R/3 API Description/Characterization" section of this document
- The "SAP R/3 API Structures" appendix of the Windchill Enterprise Systems Integration Installation and Configuration Guide

f. In the SAP R/3 menu bar, navigate to Function modules -> Execute (F8), or click the "Execute" icon to run the test.

g. Confirm the successful execution of the API function module according to the "SAP R/3 API Response Characterization" section of this document. Any errors are likely due to invalid test data or your SAP R/3 distribution target system configuration. These issues must be resolved prior to implementing Windchill Enterprise Systems Integration.

h. Use the SAP R/3 "Back (F3)" icon to proceed to the next API in the test sequence, and repeat steps e through g, above, for each API in the test sequence. Remember that you are testing a "molecular" sequence of APIs to emulate complete Windchill Enterprise Integration business functions.

i. Verify that the business object corresponding to the data you specified exists in the SAP R/3 system with the correct attributes, using SAP R/3 transactions indicated in Table 37.

For API test sequences that include CALO_INIT_API, you may obtain additional logging information in the SAP R/3 application log (transaction SLG1). Refer to SAP R/3 online application help for further details on using this transaction.

j. Repeat this procedure for each Windchill Enterprise Systems Integration Business Function Test Scenario (per Table 38).

### 7.2.6 Step 6: Validate API Molecular Functionality Outside SAP R/3

#### 7.2.6.1 SAP R/3 Readiness Validator Introduction

The Purpose of Step 6 is to isolate issues with the TIBCO BusinessWorks-to-SAP R/3 interface with realistic Windchill Enterprise Systems Integration test scenarios. Finding problems at this stage can streamline troubleshooting by ruling out the actual Enterprise Systems Integration business logic, Windchill PDM or PDMLink, and the Enterprise Systems Integration services as root causes.

Your test scenarios should match those developed for Step 5 of the SAP R/3 Readiness Validation Process. Invoke a connected, "molecular" series of SAP R/3 APIs, emulating the
manner in which they are invoked by the actual Windchill Enterprise Systems Integration solution. However, in this case, rather than invoking the APIs from within SAP R/3, you will use an API testing application developed in TIBCO BusinessWorks, and provided with the Windchill Enterprise Systems Integration EAI Windchill Simulation module. This testing utility is called the SAP R/3 Readiness Validator.

### 7.2.6.2 What It Tests

The SAP R/3 Readiness Validator allows you to execute each Windchill Enterprise Systems Integration business function and confirm that the APIs are behaving as expected. The SAP R/3 Readiness Validator supports testing of the following business functions.

<table>
<thead>
<tr>
<th>No.</th>
<th>Windchill Enterprise Systems Integration Business Function</th>
<th>Prerequisites/Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create Part</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>Create Part Revision</td>
<td>Part and change master exist</td>
</tr>
<tr>
<td>3</td>
<td>Change Part</td>
<td>Part exists</td>
</tr>
<tr>
<td>4</td>
<td>Create BOM</td>
<td>Part exists</td>
</tr>
<tr>
<td>5</td>
<td>Change BOM</td>
<td>BOM exists</td>
</tr>
<tr>
<td>6</td>
<td>Create Document</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>Create Document Revision plus Change Document plus, optionally, Add or Delete Document Links</td>
<td>Document exists; document links assume linked part exists</td>
</tr>
<tr>
<td>8</td>
<td>Create ECN</td>
<td>N/A</td>
</tr>
<tr>
<td>9</td>
<td>Change ECN³</td>
<td>ECN exists</td>
</tr>
</tbody>
</table>

Table 39 Business Functions Supported by SAP R/3 Readiness Validator

### 7.2.6.3 What It Includes

Each of the Windchill Enterprise Systems Integration Business Functions shown in Table 39 is realized as a separate process definition within the Windchill Enterprise Systems Integration EAI Windchill Simulation module, as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>Windchill Enterprise Systems Integration Business Function</th>
<th>BusinessWorks Process Definition Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create Part</td>
<td>RV_Part_SAPCreate_PD</td>
</tr>
<tr>
<td>2</td>
<td>Create Part Revision</td>
<td>RV_Revision_SAPCreate_PD</td>
</tr>
<tr>
<td>3</td>
<td>Change Part</td>
<td>RV_Part_SAPChange_PD</td>
</tr>
<tr>
<td>4</td>
<td>Create BOM</td>
<td>RV_BOM_SAPCreate_PD</td>
</tr>
<tr>
<td>5</td>
<td>Change BOM</td>
<td>RV_BOM_SAPChange_PD</td>
</tr>
<tr>
<td>6</td>
<td>Create Document</td>
<td>RV_Document_SAPCreate_PD</td>
</tr>
</tbody>
</table>

³ Change ECN is not a core Windchill Enterprise Systems Integration function, but the solution uses the Change ECN API during Create ECN operations to add alternate effectivity dates for each line item, and to update the effectivity date in the ECN header.
<table>
<thead>
<tr>
<th>No.</th>
<th>Windchill Enterprise Systems Integration Business Function</th>
<th>BusinessWorks Process Definition Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Create Document Revision plus Change Document plus, optionally, Add or Delete Document Links</td>
<td>RV_Document_SAPChange_PD</td>
</tr>
<tr>
<td>8</td>
<td>Create ECN</td>
<td>RV_ECN_SAPCreate_PD</td>
</tr>
<tr>
<td>9</td>
<td>Change ECN</td>
<td>RV_ECN_SAPChange_PD</td>
</tr>
</tbody>
</table>

**Table 40  BusinessWorks Process Definitions in the SAP R/3 Readiness Validator**

The SAP R/3 Readiness Validator also uses a shared process definition, called "Logging_Service," to provide logging functionality to each of the business functions.

The process definitions described above are located in the /ProcessDefinitions/SAPReadinessValidator folder and /ProcessDefinitions/Services folder of the Windchill Enterprise Systems Integration EAI Windchill Simulation module repository, as illustrated.
To supply test data input, the SAP R/3 Readiness Validator uses a series of external XML document files (schema instances) matching the SAP R/3 API input schemas for the fields that are expected and used by the out-of-the-box Windchill Enterprise Systems Integration solution. Samples of these files are provided on the Windchill Enterprise Systems Integration CDs. The XML document files are as follows:

<table>
<thead>
<tr>
<th>Business Function Test Scenario</th>
<th>Source Test Data XML Files Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Part</td>
<td>BAPI_MATERIAL_SAVEDATA.xml</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT.xml</td>
</tr>
<tr>
<td>Create Part Revision</td>
<td>CCAP_REV_LEVEL_MAINTAIN.xml</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT.xml</td>
</tr>
<tr>
<td>Change Part</td>
<td>BAPI_MATERIAL_SAVEDATA.xml</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT.xml</td>
</tr>
<tr>
<td>Create BOM</td>
<td>CALO_INIT_API.xml</td>
</tr>
<tr>
<td></td>
<td>CSAP_MAT_BOM_CREATE.xml</td>
</tr>
<tr>
<td>Change BOM</td>
<td>CALO_INIT_API.xml</td>
</tr>
<tr>
<td>Create Document</td>
<td>CSAP_MAT_BOM_MAINTAIN.xml</td>
</tr>
<tr>
<td>Create Document Revision plus Change Document plus, optionally, Add or Delete Document Links</td>
<td>BAPI_DOCUMENT_CREATE2.xml</td>
</tr>
<tr>
<td></td>
<td>BAPI_TRANSACTION_COMMIT.xml</td>
</tr>
<tr>
<td>Create ECN</td>
<td>CCAP_ECN_CREATE.xml</td>
</tr>
<tr>
<td>Change ECN</td>
<td>CCAP_ECN_MAINTAIN.xml</td>
</tr>
</tbody>
</table>

Table 41 Sample Source Test Data XML Files for SAP R/3 Readiness Validator

The SAP R/3 Readiness Validator closely imitates the API invocations of the actual Enterprise Systems Integration business logic, but simplifies some of the processing, for example, by not using BAPI_TRANSACTION_ROLLBACK and not re-attempting API invocations upon timeout errors. This allows you to focus your troubleshooting efforts on the expected, correct SAP R/3 API functionality rather than exception conditions or complex programming logic in TIBCO BusinessWorks.

The SAP R/3 Readiness Validator will create actual business objects in your SAP R/3 distribution target system based on the data you specify in the source test data XML files.

Caution: Do not perform these steps with production or business-critical systems. Instead, use development or sandbox systems that replicate the production environment as closely as possible.
7.2.6.4 User’s Guide

To use the SAP R/3 Readiness Validator, use the following procedure.

1. If you haven’t already, install TIBCO BusinessWorks and the TIBCO Adapter for R/3 per the Windchill Enterprise Systems Integration Installation and Configuration Guide. You do not have to configure these products yet. The SAP R/3 Readiness Validator does not require TIBCO Enterprise for JMS or Windchill PDM or PDMLink. TIBCO TurboXML is optional.

2. Start TIBCO Designer.

3. Open the EAI Windchill Simulation module repository, called ESISimulationWindchillRepository_vxxx.dat, where xxx is a version number. This repository contains the functionality for the SAP R/3 Readiness Validator and exists entirely separately from the ESIMasterRepository.

4. Configure the SAP R/3 Readiness Validator global variable values from their default settings if required; for example, you will need to change the ESIValidatorDirectory global variable value if you are using a UNIX system.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Possible Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESIValidatorDirectory</td>
<td>The directory in which the EAI Windchill Simulation module will look for the source test data XML files</td>
<td>Any valid directory.</td>
<td>c:\tibco\esi\SAPValidator\</td>
</tr>
<tr>
<td>ESIValidatorLogRole</td>
<td>Label which appears next to TIBCO BusinessWorks process engine log messages issued by the SAP R/3 Readiness Validator</td>
<td>Any text string</td>
<td>APITEST</td>
</tr>
<tr>
<td>ESIValidatorRVSubjectPrefix</td>
<td>TIBCO Rendezvous message subject sub-string used to differentiate users of the same SAP R/3 Readiness Validator domain and deployment.</td>
<td>Any valid TIBCO Rendezvous message subject sub-string</td>
<td>ESI</td>
</tr>
</tbody>
</table>

Table 42   SAP R/3 Readiness Validator Global Variables

Unlike the actual Enterprise Systems Integration business logic, the SAP R/3 Readiness Validator does not re-attempt API invocations upon timeout errors. Hence, there is a global variable for ESIR3RetryPeriod, but no global variable for ESIR3APIRetryCount. ESIR3RetryPeriod defines the time (in milliseconds) the TIBCO BusinessWorks process engine should wait for a response from an adapter before raising a technical error.
**Note:** Changes to global variable values do not take effect until the TIBCO BusinessWorks process engine is started (or restarted, if it is already running).

5. Save the EAI Windchill Simulation module repository as a server-based repository.

6. Create the subdirectory on your operating system indicated by the value of the `ESIValidatorDirectory` global variable.

7. Copy the source test data XML files, listed in Table 41, from the Windchill Enterprise Systems Integration CDs into the subdirectory on your operating system indicated by the value of the `ESIValidatorDirectory` global variable.


9. Deploy a TIBCO BusinessWorks process engine as described in the *Enterprise Systems Integration Installation and Configuration Guide*, with the following adjustments in the Deployment Configuration:
   a. Select the Process Definitions tab.
   b. Uncheck **Select All**.
   c. Click the binoculars and select all of the SAP R/3 Readiness Validator starter process definitions listed in Table 40.

10. Determine which Windchill Enterprise Systems Integration business function test scenario you are going to run, and edit the contents of the corresponding test data XML files, per Table 41.

   Your test data conditions will depend on your particular Windchill Enterprise Systems Integration deployment. You must hard-code field values that are normally defaulted or cross-referenced by the actual Enterprise Systems Integration business logic. For guidance, refer to the following documentation:

   - The "Functional Overview" section of the *Windchill Enterprise Systems Integration User’s Guide*
   - The "SAP R/3 API Description/Characterization" section of this document
   - The "SAP R/3 API Structures" appendix of the *Windchill Enterprise Systems Integration Installation and Configuration Guide*

   You may edit the XML files directly, or using any XML editor. However, be sure to save the files in UTF-8 format.
11. Start the TIBCO Adapter for R/3 instances and the TIBCO BusinessWorks process engine as described in the *Enterprise Systems Integration Installation and Configuration Guide*.

12. Execute your business function test scenario. Each SAP R/3 Readiness Validator process definition begins with a Rendezvous subscriber activity. You can trigger a desired process to execute by issuing a simple Rendezvous message at the operating system command line. On UNIX platforms, you should first navigate to the /opt/tibco/tra/1.0/bin directory. The format of the command is as follows:

```
tibrvsend [Subject] [Message]
```

where [Subject] is constructed as follows:

```
[Domain].[ESIValidatorRVSubjectPrefix].adr3.[Action].[Object]
```

and [Message] is any message text string you like, such as "start"

For example, the following command will execute the Create Part test scenario in a TIBCO BusinessWorks Administration Domain named, "ABCDomain."

```
tibrvsend ABCDomain.ESI.adr3.Create.Part "start"
```

Note the following:

- The SAP R/3 Readiness Validator will ignore any Rendezvous message that does not strictly adhere to these subject naming conventions. If your `tibrvsend` command fails to trigger execution of the corresponding BusinessWorks process, double-check the format of the command and try again.
- [Domain] is the name of your TIBCO BusinessWorks Administration Domain
- [ESIValidatorRVSubjectPrefix] is the value of the global variable of that name; the default value is "ESI"
- The valid [Action]-[Object] pairs are as follows:

<table>
<thead>
<tr>
<th>Business Function Test Scenario</th>
<th>[Action]</th>
<th>[Object]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Part</td>
<td>Create</td>
<td>Part</td>
</tr>
<tr>
<td>Create Part Revision</td>
<td>Create</td>
<td>Revision</td>
</tr>
<tr>
<td>Change Part</td>
<td>Change</td>
<td>Part</td>
</tr>
<tr>
<td>Create BOM</td>
<td>Create</td>
<td>BOM</td>
</tr>
<tr>
<td>Change BOM</td>
<td>Change</td>
<td>BOM</td>
</tr>
<tr>
<td>Create Document</td>
<td>Create</td>
<td>Document</td>
</tr>
<tr>
<td>Business Function Test Scenario</td>
<td>[Action]</td>
<td>[Object]</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Create Document Revision plus Change Document plus, optionally, Add or Delete Document Links</td>
<td>Change</td>
<td>Document</td>
</tr>
<tr>
<td>Create ECN</td>
<td>Create</td>
<td>ECN</td>
</tr>
<tr>
<td>Change ECN</td>
<td>Change</td>
<td>ECN</td>
</tr>
</tbody>
</table>

Table 43  [Action]-[Object] Pairs for tibrvsend Trigger Command

13. Verify successful execution of the test scenario. The SAP R/3 Readiness Validator logs the results of each API invocation in the TIBCO BusinessWorks process engine log. The log messages indicate whether the TIBCO Adapter for R/3 was able to invoke each API ("technical success"), and whether the test data processed by that API resulted in a "functional success" based on the guidelines specified in the "SAP R/3 API Response Characterization" section of this document. The SAP R/3 Readiness Validator also logs additional debugging data where appropriate, such as the API return message from SAP R/3, or the TIBCO BusinessWorks stack trace.

For example, the following messages will be written to the TIBCO BusinessWorks process engine log upon successful execution of the Create Part scenario.

```

2003 Sep 03 12:12:06:340 GMT -4 Engine APITEST [] PE-APITEST Job-1000 [ProcessDefinitions/Services/Logging_Service/Log]: Create Part Step 2: BAPI_MATERIAL_SAVEDATA functional success. TYPE=S, ID=MM, NUMBER=356, MESSAGE=The material STUBTEST1 has been created or extended


```

In the following example, the API invocation of BAPI_MATERIAL_SAVEDATA was successful, but the TIBCO BusinessWorks process encountered a technical error upon invoking BAPI_TRANSACTION_COMMIT.

```
```
If a scenario fails, there is a chance that the API invoke activities may freeze upon subsequent execution attempts. This may be due to the lack of BAPI_TRANSACTION_ROLLBACK invocations in the SAP R/3 Readiness Validator. If this occurs, restart the relevant adapter instance and try again.

14. Verify that the business object corresponding to the data you specified in the source test data XML files exists in SAP R/3 with the correct attributes, using the SAP R/3 transactions indicated in Table 37.

15. Repeat steps 10-14 of this procedure for each of your test scenarios and conditions.

7.3 SAP R/3 Readiness Validator Technical Design

This section describes the technical design of the SAP R/3 Readiness Validator, the API testing application developed in TIBCO BusinessWorks, and provided with the Windchill Enterprise Systems Integration EAI Windchill Simulation module (ESISimulationWindchillRepository_xx.dat). The SAP R/3 Readiness Validator is used during Step 6 of the SAP R/3 Readiness Validation Process. Refer to the section entitled, "Step 6: Validate API Molecular Functionality Outside SAP R/3," for a thorough introduction to the SAP R/3 Readiness Validator.

7.3.1 Rendezvous Configuration

A Rendezvous Configuration, ESIMaster_RVTransport, is in the /SharedConfigurations/RV folder. This configuration exists to facilitate the triggering of business function test scenarios via tibrvsend commands, as described in step 12 of the User’s Guide for the SAP R/3 Readiness Validator. ESIMaster_RVTransport is configured as follows:

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Out-of-the-box Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ESIMaster_RVTransport</td>
</tr>
<tr>
<td>Daemon</td>
<td>%%RvDaemon%%</td>
</tr>
<tr>
<td>Network</td>
<td>%%RvNetwork%%</td>
</tr>
</tbody>
</table>
Table 44  SAP R/3 Readiness Validator Rendezvous Configuration

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Out-of-the-box Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>%%RvService%%</td>
</tr>
<tr>
<td>RV Type</td>
<td>Reliable</td>
</tr>
</tbody>
</table>

### 7.3.2 Business Function Test Scenario Process Definitions

Each of the SAP R/3 Readiness Validator business function test scenarios is realized as a separate process definition within the Windchill Enterprise Systems Integration EAI Windchill Simulation module, as listed in Table 40. Each of these process definitions closely imitates the API invocations of the actual Enterprise Systems Integration business logic, but simplifies some of the processing, for example, by not using BAPI_TRANSACTION_ROLLBACK and not re-attempting API invocations upon timeout errors. The "Create Part" process definition, named RV_Part_SAPCreate_PD, is described below as an illustrative example. Refer to the Windchill Enterprise Systems Integration Installation and Configuration Guide and the "SAP R/3 API Descriptions/Characterization" section of this document for further details on the SAP R/3 API invocation logic of Windchill Enterprise Systems Integration.

#### 7.3.2.1 Logical Flow

The "Create Part" process definition, named RV_Part_SAPCreate_PD, waits for the required Rendezvous message to trigger its execution. The process then reads in the XML test data input for BAPI_MATERIAL_SAVEDATA, invokes the BAPI, and determines whether the invocation was a technical success and a functional success. If an error occurs, the process ends. If there are no errors, the process then repeats the same basic procedure for BAPI_MATERIAL_SAVEDATA. Technical and functional successes and failures are logged to the TIBCO BusinessWorks process engine log via inline calls to the shared Logging_Service process. The process definition is illustrated below.
### 7.3.2.2 Components

The following chart describes the TIBCO BusinessWorks activities and branching conditions that comprise the "Create Part" process definition, RV_Part_SAPCreate_PD.

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVSubscriber_CreatePart_Part</td>
<td>Receipt of any Rendezvous message with the subject format: [Domain].[ESIValidatorRVSubjectPrefix].adr3.Create.Part</td>
<td>None</td>
<td>A Rendezvous message with the required subject format must be received to trigger the start of the process</td>
</tr>
<tr>
<td>ReadFile_BAPI_MATERIAL_SAVEDATA</td>
<td>UTF-8-encoded data file named &quot;BAPI_MATERIAL_SAVEDATA.xml&quot; in the path specified by the global variable, ESIValidatorDirectory</td>
<td>Contents of the file as a single, text string</td>
<td>The file exists with UTF-8 encoding</td>
</tr>
<tr>
<td>XMLParser_BAPI_MATERIAL_SAVEDATA</td>
<td>The contents of BAPI_MATERIAL_SAVEDATA.xml in a single, text string</td>
<td>The contents of BAPI_MATERIAL_SAVEDATA.xml in an XML structure</td>
<td>Schema at /SharedConfigurations/ESISchemas/SAPInteractionSchemas/BAPI_MATERIAL_SAVEDATA</td>
</tr>
<tr>
<td>Invoke_BAPI_MATERIAL_SAVEDATA</td>
<td>The parsed output of BAPI_MATERIAL_SAVEDATA.xml</td>
<td>The SAP R/3 return structure from executing BAPI_MATERIAL_SAVEDATA</td>
<td>Imported (fetched) structure of BAPI_MATERIAL_SAVEDATA in the SAP Part Configuration Adapter Services</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_TechnicalSuccess</td>
<td>Required input parameters of the Logging_Service process.</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_MATERIAL_SAVEDATA technical success</td>
<td>None</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_TechnicalFailure</td>
<td>Required input parameters of the Logging_Service process, plus the StackTrace</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_MATERIAL_SAVEDATA technical failure</td>
<td>None</td>
</tr>
<tr>
<td>Mapper_BAPI_MATERIAL_SAVEDATA_BAPI_MATERIAL_SAVEDATA_DetermineSuccess</td>
<td>&quot;ID&quot; and &quot;NUMBER&quot; parameters of return structure of BAPI_MATERIAL_SAVEDATA</td>
<td>CreatePartSuccess variable = &quot;true&quot; or &quot;false&quot; depending on contents of return structure of BAPI_MATERIAL_SAVEDATA. Functional success is established if and only if ID=&quot;MM&quot; and &quot;NUMBER&quot;=&quot;356</td>
<td>BAPI_MATERIAL_SAVEDATA invocation was successful, technically</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_FunctionalSuccessFailure</td>
<td>Required input parameters of the Logging_Service process, plus the API return structure fields, TYPE, ID, MESSAGE, and NUMBER. Functional success or failure is indicated by the value of the variable CreatePartSuccess.</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_MATERIAL_SAVEDATA functional success or failure</td>
<td>CreatePartSuccess variable must have a value of &quot;true&quot; or &quot;false&quot;</td>
</tr>
<tr>
<td>ReadFile_BAPI_TRANSACTION_COMMIT</td>
<td>UTF-8-encoded data file named &quot;BAPI_TRANSACTION_COMMIT.xml&quot; in the path specified by the global variable, ESIValidatorDirectory</td>
<td>Contents of the file as a single, text string</td>
<td>The file exists with UTF-8 encoding</td>
</tr>
<tr>
<td>XMLParser_BAPI_TRANSACTION_COMMIT</td>
<td>The contents of BAPI_TRANSACTION_COMMIT.xml in a single, text string</td>
<td>The contents of BAPI_TRANSACTION_COMMIT.xml in an XML structure</td>
<td>Schema at /SharedConfigurations/ESISchemas/SAPInteractionSchemas/BAPI_TXANSCTION_COMMIT</td>
</tr>
</tbody>
</table>
### Activity Name | Input | Output | Dependencies |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke_BAPI_TRANSACTION_COMMIT</td>
<td>The parsed output of BAPI_TRANSACTION_COMMIT.xml</td>
<td>The SAP R/3 return structure from executing BAPI_TRANSACTION_COMMIT</td>
<td>Imported (fetched) structure of BAPI_TRANSACTION_COMMIT in the SAP Part Configuration Adapter Services</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_TRANSACTION_COMMIT_TechnicalSuccess</td>
<td>Required input parameters of the Logging_Service process.</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_TRANSACTION_COMMIT technical success</td>
<td>None</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_TRANSACTION_COMMIT_TechnicalFailure</td>
<td>Required input parameters of the Logging_Service process, plus the StackTrace</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_TRANSACTION_COMMIT technical failure</td>
<td>None</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_TRANSACTION_COMMIT_FunctionalSuccess</td>
<td>Required input parameters of the Logging_Service process, plus the API return structure fields, TYPE, ID, MESSAGE, and NUMBER.</td>
<td>Line appended to BusinessWorks process engine log indicating BAPI_TRANSACTION_COMMIT functional success</td>
<td>None</td>
</tr>
</tbody>
</table>

### Branching details are as follows:

<table>
<thead>
<tr>
<th>Branch From Activity</th>
<th>Condition</th>
<th>If True, Branch To</th>
<th>If False, Branch To</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke_BAPI_MATERIAL_SAVEDATA</td>
<td>If the BAPI_MATERIAL_SAVEDATA invocation does not encounter a technical error</td>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_TechnicalSuccess</td>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_TechnicalFailure</td>
<td>A standard error branch condition automatically detects any technical API invocation errors, such as timeouts</td>
</tr>
<tr>
<td>CallProcess_Logging_Service_BAPI_MATERIAL_SAVEDATA_FunctionalSuccessFailure</td>
<td>If CreatePartSuccess = &quot;false&quot;</td>
<td>End</td>
<td>ReadFile_BAPI_TRANSACTION_COMMIT</td>
<td>Determines whether to continue the process based on the functional success of BAPI_MATERIAL_SAVEDATA</td>
</tr>
<tr>
<td>Invoke_BAPI_TRANSACTION_COMMIT_BAPI_MATERIAL_SAVEDATA</td>
<td>If the BAPI_TRANSACTION_COMMIT invocation does not encounter a technical error</td>
<td>CallProcess_BAPI_TRANSACTION_COMMIT_TechnicalSuccess</td>
<td>CallProcess_BAPI_TRANSACTION_COMMIT_TechnicalFailure</td>
<td>A standard error branch condition automatically detects any technical API invocation errors, such as timeouts</td>
</tr>
</tbody>
</table>

Table 45 "Create Part" Process Definition Components

7.3.3 Logging_Service Process Definition

The SAP R/3 Readiness Validator used a shared logging service process definition, named Logging_Service, to handle the writing of functional and technical API success and failure messages to the TIBCO BusinessWorks process engine log.
7.3.3.1 Logical Flow

The Logging_Service process definition consists of Start and End nodes, and a Write To Log activity. A schema defines required and optional input parameters for the process. The process definition is illustrated below.

![Diagram](image)

Figure 84  "Create Part" Process Definition, RV_Part_SAPCreate_PD

7.3.3.2 Components

The following chart describes the TIBCO BusinessWorks activities that comprise the Logging_Service process definition.

<table>
<thead>
<tr>
<th>Activity Name</th>
<th>Input</th>
<th>Output</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>APIReturnLog schema instance, with required fields populated</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Log</td>
<td>APIReturnLog schema instance, with required fields populated</td>
<td>Line appended to BusinessWorks process engine log indicating API technical or functional success or failure</td>
<td>None</td>
</tr>
<tr>
<td>End</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 46  Logging_Service Process Definition Components

7.3.3.3 Input Schema

The input schema for the Logging_Service process is called APIReturnLog. The following chart describes the required and optional input parameters of this schema.

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Cardinality</th>
<th>Type</th>
<th>Value Restrictions</th>
<th>Comment</th>
</tr>
</thead>
</table>
| LeadingText       | Required    | String  |  | Recommended format: 
  
  "[Action] [Object] Step n: [API Name]"
  
  For example: "Create Part Step 1: BAPI_MATERIAL_SAVEDATA" |
| FunctionalorTechnical | Required    | String  | "F" or "T"         | Indicates whether logging functional ("F") or technical ("T") API invocation results. |
| Success           | Required    | String  | "true" or "false"  | "true" indicates the API was invoked successfully; "false" indicates a failure. |
| TYPE              | Optional    | String  |  | Mapped directly from the API Return structure for API functional success and failure messages. Standard SAP R/3 message types include:
  • S (Success)
  • I (Information)
  • W (Warning)
  • E (Error)
  • A (Abend) |
| ID                | Optional    | String  |  | Mapped directly from the API Return structure for API functional success and failure messages. SAP R/3 message IDs are also known as message classes. |
### 8 Windchill ESI Best Practices

The following practices help minimize problems resulting from customizing the EAI software components:

- Maintain consistency with the basic structure of the standard Windchill ESI code
- Employ modular programming techniques. Define customizations via separate, "black box" sub-processes with well-defined inputs and outputs.
- Use special care in modifying the Master Process Flow, as this module serves crucial roles in communications with Windchill and transaction management.
- Use special care in modifying the shared services. These processes are called in-line from many locations throughout the code, so even minor changes can have substantial overall impacts.
- Abide by the Windchill ESI naming standards and conventions described elsewhere in this guide.
- Use labeling and comments to clearly distinguish customizations from standard Windchill ESI code.
- Attempt to minimize potential impacts to technical support and upgrade processes.

### 9 Examples - Windchill to SAP R/3

This chapter contains examples of typical customizations of Windchill ESI. It also includes descriptions of possible modifications to Windchill and/or SAP R/3 to accommodate these customizations.

#### 9.1 Publish Costing Information From Windchill to SAP R/3

In this example, WTPart is extended by adding new attributes. Parts are then created using the new WTPart and published to SAP R/3. In SAP R/3 two additional views are created, Costing 1 and Costing 2.

**Note:** The Windchill ESI Services and Middleware processes would be the same, if a soft sub-type of WTPart were created in Windchill instead of adding soft attributes directly to WTPart. For more information about soft attributes and soft types, refer to the *Windchill PDMLink Customizer's Guide*.
Windchill Process:
A. Create Attributes
1. In the Attribute Definition Manager window, create a new Organizer by clicking the Create Organizer button and call it CustomizationOrganizer.

![Create Attribute Organizer](image)

Figure 85  Create Attribute Organizer

2. Highlight the CustomizationOrganizer and click the Create Attribute button to create the attributes.

3. Enter one of the following values in the Create Attribute screen and click OK. Repeat 2 and 3 for the three attributes:
   a. **StandardPrice** (String)
   b. **PriceUnit** (String)
   c. **PlantStatus** (String)

![Create Attribute](image)

Figure 86  Create Attribute
B. Modify the WTPart type
   1. In the **Type Manager** window, highlight the wtpart type
   2. Click the **Update** button
   3. Add the attributes created in step 3 to the wtpart type
      a. Click the **Update** button
      b. Go to the Template tab
      c. Select Root and click the **Add Attribute** button
      d. Select PlantStatus and click **Select**

      Figure 87  Add attribute to the WTPart Type

      e. Re-select Root and click on the **Add Attribute** button
      f. Select Price Unit and click **Select**
      g. Repeat step e and f and select the StandardPrice attribute
      h. Repeat step e and select the PlantStatus attribute and click on the **Show Constraint** button
      i. Click **Add**
      j. Select the Discrete Set Constraint
Figure 88  Base Selector

k. Enter the following value in the Data field:

|ProcurementWarehouse|TaskListBOM|Purchasing|Costing|PilotPhase|

l. Enter default values seen in Figure 89, below.
Windchill ESI services Process:

1. Copy the existing Part Map file ($CODEBASE$/esipart/ESIPart.map), or create a new one

   *Caution:* Do not modify the map files that were provided when you installed Windchill ESI. These files may be replaced by future software releases. If you wish to modify them, copy them to a new location.

2. Verify the names of the attributes

3. Add the following entries to the new map file, save the new map file. Make sure the order of the attributes in this file is the same as the ESIResponse schema
   - StandardPrice=StandardPrice
   - PriceUnit=PriceUnit
   - PlantStatus=PlantStatus

4. Modify the Windchill ESI properties file ($CODEBASE$/esi.properties) to point to the new map file created in step 1

5. Restart Windchill to effect the changes

EAI Software Components Process:

1. **Modify the ESIResponse schema with the changes.** This requires completion of the following steps:
   - Use an XML editor to add the Costing fields to the ESIResponse schema. Save it as a new xsd file. Each section in the Response uses the Part schema, so adding the Costing information solely to Parts persists the addition into all four Part sections (AddedParts, ChangedParts, UnchangedParts, DeletedParts). Make the three Costing fields optional, so Windchill may send costing information optionally in its release.
     b. In the Configuration tab, browse for the new file. Then click **Load Schema** in the lower left hand corner and apply the changes.
   - The changes filter through the BusinessWorks process to anywhere the ESIResponse schema is used.

   The following is a screenshot of the AddedParts section of the Response schema after the modifications were made. Remember the Costing data was added to the Part schema, so it appears AddedParts, ChangedParts, and UnchangedParts.
2. **Modify the CBO UserArea.** In each object’s part of the CBO, a UserArea exists. These UserAreas may be modified to persist more data through the EAI software components. To add Costing information to the PartUserArea structure of the CBO, follow the following steps:
a. Create a new XSD schema with the Costing structure. The root element of the XSD must be called PartUserArea. (NB: When adding to UserArea structures of other objects, the root node must match the name of the existing UserArea in that structure; for example, DocumentUserArea.) Make sure the Costing structure is optional, so it can be mapped whether or not costing information comes in the release from Windchill. You may also modify any existing xsd schema, as long as the root element is called PartUserArea. Save the new schema as a new xsd file.


c. In the Configuration tab, browse for the new file. Then click Load Schema in the lower left hand corner and apply the changes.

The following is a screenshot of the PartUserArea schema after the Costing information was added:

Figure 91 Updated PartUserArea

3. **Ensure the changes were properly applied.** Spot-check the places in the BusinessWorks process where these new sections are used, such as in the Output tab of the Start node of Start_ESIResponse_Map_CBO_PD. If the changes were not persisted, go back to the previous step and try again.

4. **Map the new Response information to the new fields in the CBO.** In this case the Costing structure was mapped to the CBO in the activities Mapper_ESIResponse_CBO_Headers and Mapper_CBO_CBO_ChangeNumber on the process definition Start_ESIResponse_Map_CBO_PD. Be sure to map the structure from each applicable structure; the Response is organized by action while the CBO changes the action to a field within it.

**Mapper_ESIResponse_CBO_Headers**

Because BusinessWorks differentiates between optional fields that do not exist and optional fields that are present and empty, we need to check that all three of the fields are present, then check that they contain business data.

a. Click on the PartUserArea of the AddedParts section.
b. Click the exclamation point icon and insert a statement of type "If". This inserts a conditional above the PartUserArea structure.

c. Click on the "If" structure that was just created and click the exclamation point icon again. Click the "Insert Statement" radio button and choose type "If". This creates a doubly nested if statement:

![Figure 92  Response to CBO mapping](image)

```plaintext
If

- PartUserArea
  - Costing
    - StandardPrice
    - PriceUnit
    - PlantStatus

Figure 92  Response to CBO mapping
```

d. In the first If statement, create an Xpath formula that uses the exists() function to check that all three of the Costing elements are present. The statement is as follows:

```plaintext
((exists(pfx:StandardPrice)) and (exists(pfx:PriceUnit)) and (exists(pfx:PlantStatus)))
```

e. In the second If statement, create an Xpath formula that uses the string-length() function to check that all three Costing elements have a length greater than zero. The statement is as follows:

```plaintext
((string-length(pfx:StandardPrice) > 0) and (string-length(pfx:PriceUnit) > 0) and (string-length(pfx:PlantStatus) > 0))
```

f. Repeat steps a-f for the ChangedParts and UnchangedParts buckets.

The final mapping should include the two populated if statements for each section:

![Figure 93  Response to CBO mapping – if Statements](image)

```plaintext
Mapper_CBO_CBO_ChangeNumber
```

Copy the contents of the PartUserArea as shown in the figure below:
5. **Add to the ESIDefaults.properties and ESILookups.properties files.** Add the defaulted and looked-up information to the files according to the data structure of those files.

The following entries were added to the ESIDefaults.properties file:

- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATAX.STD_PRICE.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATAX.VAL_AREA.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATAX.VAL_CLASS.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATAX.PRICE_UNIT.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATAX.PRICE_CTRL.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATA.PRICE_CTRL.*.*=S`
- `BAPI_MATERIAL_SAVEDATA.VALUATIONDATA.VAL_CLASS.*.*=X`
- `BAPI_MATERIAL_SAVEDATA.HEADDATA.COST_VIEW.*.*=X`

The following entries were added to the ESILookups.properties file:

- `PlantStatus.ProcurementWarehouse.*.*=01`
- `PlantStatus.TaskListBOM.*.*=02`
- `PlantStatus.Purchasing.*.*=BP`
- `PlantStatus.Costing.*.*=KA`
- `PlantStatus.PilotPhase.*.*=PI`
- `ValuationData.ROH.*.*=3000`
- `ValuationData.HALB.*.*=7900`

6. **Modify the mappings in other parts of the code.** In the process definition `Start_CBOPart_ProcessPartByOrganization_PD`, map a copy of the `PartUserArea` from CBO to CBO in the activities `CallProcess_Start_CBOPart_SAPCreate_PD` and `CallProcess_Start_CBOPart_SAPChange_PD`.

7. **Modify the API mapping.** Map the new fields from the CBO into the API.
The following shows the defaulting of the COST_VIEW variable into the HEADDATA structure of the API:
The following shows the new API mappings to the VALUATIONDATA structure of the API:
The shows the new API mappings to the VALUATIONDATAX structure of the API:

How to run this example:
1. Verify that the properties files in the ESIDirectory have the appropriate modifications
2. Verify that the appropriate distribution targets have been loaded into Windchill
3. Verify that there is at least one Lifecycle that has Released state
4. Verify that all components (Windchill, TIBCO, SAP R/3) are up and running
5. Create a new part
6. Enter the required attributes, and the three attributes for this scenario (StandardPrice, PriceUnit and PlantStatus).
7. Add the desired Distribution Targets
8. Release the part
9. At this point, the Event should be kicked off, and the process should be started.
10. Once the process is complete, log into SAP R/3 and verify the data was created correctly and the Costing Views are populated.
9.2 Publish Iteration and Lifecycle State From Windchill to SAP R/3

Windchill ESI services out-of-the-box supports the publishing of object iterations. For this customization, the EAI software components were updated to persist the iteration data to a text field in SAP R/3 along with the Lifecycle State.

**Windchill ESI Services Process:**
1. Copy the existing Part Map file ($CODEBASE$/esipart/ESIPart.map), or create a new one

   **Caution:** Do not modify the map files that were provided when you installed Windchill ESI. These files may be replaced by future software releases. If you wish to modify them, copy them to a new location.

2. Verify the names of the attributes
3. Add the following entries to the new map file, save the new map file
   state.state=State
4. Modify the Windchill ESI properties file ($CODEBASE$/esi.properties) to point to the new map file created in step 1
5. Restart Windchill to effect the changes

**EAI Software Components Process:**
1. **Modify the ESIResponse schema with the changes.** This requires completion of the following steps:
   a. Use an XML editor to add State to the ESIResponse schema. Save it as a new xsd file. Each section in the Response uses the Part schema, so adding the State information solely to Parts persists the addition into all four Part buckets (CreatedParts, ChangedParts, UnchangedParts, DeletedParts). Make State optional, so Windchill may send costing information optionally in its release.
   c. In the Configuration tab, browse for the new file. Then click **Load Schema** in the lower left hand corner and apply the changes.
   
   The changes should filter through the BusinessWorks process to anywhere the ESIResponse schema is used.

   The ESIResponse schema change in the EAI software components should mirror the changes made to the map file in Windchill.

2. **Modify the CBO UserArea.** In each object’s part of the CBO, a UserArea exists. These UserAreas may be modified to persist more data through the EAI software components.
3. To add lifecycle state information to the PartUserArea structure of the CBO, follow the following steps:
   a. Create a new XSD schema with the State structure. The root element of the XSD must be called PartUserArea. (NB: When adding to UserArea structures of other objects, the root node must match the name of the existing UserArea in that structure; for example, DocumentUserArea.) Make sure State is optional, so it can be mapped whether or not state information comes in the release from Windchill. You may also modify any existing xsd schema, as long as the root element is called PartUserArea. Save the new schema as a new xsd file.
   c. In the Configuration tab, browse for the new file. Then click "Load Schema" in the lower left hand corner and apply the changes.

The following is a screenshot of the PartUserArea schema after the State information was added:

4. **Ensure the changes were properly applied.** Spot-check the places in the BusinessWorks process where these new sections are used, such as in the Output tab of the Start node of Start_ESIResponse_Map_CBO_PD.

5. **Map the new Response information to the new fields in the CBO.** In this case State was mapped to the CBO in the activities Mapper_ESIResponse_CBO_Headers, and Mapper_CBO_CBO_ChangeNumber on the process definition Start_ESIResponse_Map_CBO_PD. Be sure to map the structure from each applicable structure; the Response is organized by action while the CBO changes the action to a field within it.

6. **Add to the ESIDefaults.properties.** Add the defaulted information to the ESIDefaults.properties file according its data structure. There are no cross referenced information for this scenario.

The following entries were added to the ESIDefaults.properties file:

```
BAPI_MATERIAL_SAVEDATA.MATERIALLONGTEXT.APPOBJECT.*.*=MATERIAL
BAPI_MATERIAL_SAVEDATA.MATERIALLONGTEXT.TEXT_ID.*.= GRUN
BAPI_MATERIAL_SAVEDATA.MATERIALLONGTEXT.LANGU_ISO.*.=EN
BAPI_MATERIAL_SAVEDATA.MATERIALLONGTEXT.TEXT_LINE.*.=Iteration
```

7. **Modify the mappings in other parts of the code.** In the process definition Start_CBOPart_ProcessPartByOrganization_PD, map a copy of the PartUserArea from CBO to CBO in the activities CallProcess_Start_CBOPart_SAPCreate_PD and CallProcess_Start_CBOPart_SAPChange_PD.
8. **Modify the API mapping.** Map the new fields from the CBO into the BAPI_MATERIAL_SAVEDATA API. Ensure that this is done for both create and change Part code. In the example case, the mappings are as follows (in the create part section):

![Diagram](image)

The mapping for the TEXT_LINE field is:

```
concat(ESIDataProcessing:GetDefaults("BAPI_MATERIAL_SAVEDATA", "MATERIALLONGTEXT", "TEXT_LINE", ObjectHeader/ESITarget[1]/System, ../ControlData/Locale), " ", Iteration, ": " , PartUserArea/State)
```

This is a very straightforward mapping, with one exception. The mapping must occur when the Iteration field is populated with information. BusinessWorks differentiates between optional fields that exist and are empty, and optional fields that do not exist at all. Therefore the MATERIALLONGTEXT mapping must be mapped only if two conditions are met: one, that the Iteration field exists; and two, that the Iteration field is not empty. It is also good practice to only check the length of a field if it exists. Thus the MATERIALLONGTEXT structure contains two "if" conditions, one ensuring that the Iteration field exists, and if that is true, whether it is populated. If both conditions are met, the mappings to the structure are completed.

**How to run this example:**
1. Verify that the properties files in the ESIDirectory have the appropriate modifications
2. Verify that the appropriate distribution targets have been loaded into Windchill
3. Verify that there is at least one Lifecycle that has Released state
4. Verify that all components (Windchill, TIBCO, SAP R/3) are up and running
5. Create a new part and release it to SAP R/3 using the sample Windchill ESI workflow while keeping the part in the IN WORK lifecycle state
6. Once the process is complete, log into SAP R/3 and verify the data was created correctly and the iteration and lifecycle state (INWORK) are populated in the long text field of the material
7. Return to Windchill, change the iteration of the part and release it
8. Verify that the long text field data has changed and now says RELEASED

10 Glossary

10.1 Application Programming Interface (API)
A set of functions that can be invoked by other programs. An API provides a standard way to access the publicly available services of a software system.

JMS (Java Message Service)
A standard set of interfaces to Message Oriented EAI Software Components, such as TIBCO. The standard was published by Sun Microsystems, Inc.

XML (eXtended Markup Language)

SOAP (Simple Object Access Protocol)
A lightweight XML-based protocol for exchange of information in a decentralized, distributed environment. The SOAP 1.1 standard was published by the W3C.

SOAP RPC (SOAP Remote Procedure Call)
A SOAP-formatted request for information coupled with a SOAP-formatted response to the request. The SOAP 1.1 standard contains more details about SOAP RPCs.

10.2 Architecture
A description of the elements (for example, modules, components, subsystems) that comprise a software system. Architecture includes the relationships between the elements and the interfaces between elements.

Adapter
The term, "adapter," means: "software that allows one software module or system to use the services of another module or system."
Components

Anything that is necessary for system operations. Elements such as hardware, operating systems, applications and databases. Includes supporting utilities, required operating system services, and configuration files/repositories.

Distribution Engine

A set of components that work together to move data between one software system and another.

Info*Engine Adapter

An adapter that uses the services of Info*Engine. Info*Engine adapters are often written using PTC’s Java Adapter Development Kit. Info*Engine adapters accept requests in the form of webjects.

Info*Engine ERP Adapter

An Info*Engine adapter that is designed to access and/or update objects in an ERP system. These adapters accept requests in the form of webjects. They also support molecular transactions, such as "Update BOM."

Logical Architecture

A definition of the software processes that comprise a software system. Logical architecture operates one or more physical architectures.

Physical Architecture

A definition of the computers, peripheral devices, and network connections that comprise a software system.

System Architecture (a.k.a. Application Architecture)

A definition of the software components that cooperate to deliver the services of a software system. Application architecture defines the way logical architecture processes are utilized in the system.

TIBCO Adapter

An adapter that uses the services of TIBCO. These adapters are invoked using TIBCO messages, rather than webjects. They do not support molecular or composite transactions. Instead, they support atomic transactions by exposing the individual API’s of the software for which the adapter is designed. While making the configuration of these adapters more complicated (low level), this architecture adds more flexibility by enabling implementers to compose molecular transactions from logically organized atomic transactions. Atomic transactions can be selected, then logic can be applied to configure the molecular transaction.

Windchill Adapter (a.k.a. WTAdapter)

An Info*Engine adapter that accesses and updates objects in Windchill.
10.3 Enterprise Systems Integration

An information bridge between two enterprise systems. In the current release, Enterprise Systems Integration connects PDM systems with other enterprise systems, such as ERP systems.

Alternate Part

In Windchill, a form, fit, and function replacement for a part in every product structure usage. The replacement is global to a Windchill implementation.

Assembly Part

In Windchill, a part that can contain other parts— that is, a product structure.

BOM (Bill of Materials)

An ERP structure that identifies the components that are used to make an assembly.

Component Part

In Windchill, a part that cannot contain other parts.

Configuration

A procedure performed by a System Integrator that enables standard Windchill functionality. The criterion is that the System Integrator must be able to perform this activity with the "Quick Start" timeframe. In general, the procedure should not require advanced programming skills.

Custom attributes

Custom attributes are attributes that are added to a Windchill object. They may be added by modifying Windchill object models, by adding IBAs to a Windchill object, or by adding a soft-type to a Windchill object.

Customization

A procedure performed by a Customizer that extends standard Windchill functionality. In general, it requires Java programming skills (or equivalent other technical skills) and requires more time and resource than the "Quick Start" timeframe will allow.

EAI (Enterprise Application Integration)

A topic or discipline that covers the tools, technology, companies and resources associated with the integration of enterprise software applications. EAI is responsible for:

- Determining destinations for messages and data
- Mapping data to the appropriate format for destinations
- Providing services (adapters) that deliver messages to destinations
- Insuring that messages are delivered to destinations
- Reporting the response of the destination to the delivery

ERP (Enterprise Resource Planning)

Software that plans and tracks manufacturing activities.
**Getter Method**

Attributes may be defined for a Windchill object. Frequently, the contents of the attribute are made available to other objects via a java method. Typically, such methods have names that begin with "get…" (e.g., "getNumber()" would return the contents of the number attribute). Windchill code generation tools provide the option to generate getter methods for modeled attributes. Windchill developers and customizers may add additional getter methods.

**Local attributes**

Attributes that are contained by an object are local attributes. Attributes of an object may be other objects.

**Multi-Level BOM**

In Windchill, an assembly that contains one or more assemblies.

**Non-local attributes**

If an object has access to another object by any other means than containment (such as a calculation), that object is a non-local attribute. Attributes of contained objects are also non-local. In other words, if you are required to navigate from one object to another to obtain a value, the attribute you obtain is non-local.

**Selection Rule**

Determines which object types are released by Windchill ESI. Also determines which attributes of the object are released.

**Source System**

Source systems identify the systems to send data when discussing enterprise systems integration. For Enterprise Systems Integration, the source system is Windchill PDM.

**Substitute Part**

In Windchill, a form, fit, and function replacement for a part in one specific product structure usage.

**System of Record**

Data elements are often shared between systems. Sometimes more than one of these systems create and modify the data. If only one system is allowed to modify the data, that system is the system of record for the data.

**Target System**

Target systems identify the systems to receive data when discussing enterprise systems integration. For example:

- SAP R/3 Instances and Plants (for receiving product data, such as parts, documents, assemblies, and engineering change orders)
**Transaction**

Typical distributed systems transaction, for example, order processing. Does not imply Database Management System commit and rollback capability.

- **Atomic Transaction** - A transaction containing one request that cannot be further subdivided into other transactions.
- **Molecular Transaction** - A group of atomic transactions that must be processed as a unit.
- **Composite Transaction** - A complex aggregate of other transactions. It could be comprised of atomic and/or molecular transactions.

**Windchill Core**

PDM functionality, including product structures, change processes, and workflows. Info*Engine is the communication vehicle for exchanging information with external systems via EAI.
11 Appendices

11.1 Windchill ESI Services Customization Scenarios

11.1.1 Functional Overview

Windchill ESI services provide SOAP RPCs that allow enterprise applications to access to ECNs, BOMs, Parts, and Documents related to Parts. The RPCs are coded to extract data describing specific Windchill objects by reading the objects’ attributes and mapping them to XML. The XML format is described in the *Windchill Enterprise Systems Integration Open Application Programming Interface Guide*.

Object attributes types supported by Windchill ESI services include modeled attributes, instance based attributes (IBAs), and soft-typed attributes. Typically a modeled attribute is local to the object and can be obtained by a "getter" method that is created by the Windchill code generation process. Sometimes, though, the RPC code must derive the attribute value by calculation or by navigating object relationships. In those cases, the getter method is not provided by code generation. Java must be coded explicitly to navigate relationships and obtain the attribute value.

Generally when custom attributes are modeled, they can be mapped in Windchill ESI by performing a configuration procedure. But for non-local custom attributes, a customization procedure is required. (See [Glossary](#) for a definition of non-local custom attributes).
The following table describes how customized Windchill Objects are supported in ESI Services:

<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects(^6)</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTPart</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>None, works as is</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Publish a WTPart with documents; documents requested</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>None, works as is</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
<tr>
<td>Publish a WTPart; documents modified, but documents not requested</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>None, works as is</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
<tr>
<td>Publish a WTPart with custom additional modeled(^7) attributes not mapped(^8); documents not modified or documents not requested</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>None, works as is</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
<tr>
<td>Publish a WTPart with custom additional IBA or soft-typed attributes(^9) not mapped; documents not modified or documents not requested</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>None, works as is</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
</tbody>
</table>

\(^6\) WTDocument, WTDocumentMaster, and EPMDocument are only affected if the relevant Windchill ESI RPC supports documents and if the Windchill ESI RPC invocation requests that documents be included. Windchill ESI RPCs GetPart, GetBOM and GetECN do support documents.

\(^7\) Attributes described in the table as “modeled” attributes are presumed to have public getter methods.

\(^8\) Non-mapped attributes are not included in the Windchill ESI Open API RPC responses.

\(^9\) Windchill ESI uses the Windchill Adapter Query-Object webject to obtain all the attributes of parts and part master objects. As long as the Query-Object webject returns the attribute, that attribute is available to Windchill ESI mapping logic.
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects⁶</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTPart with custom additional modeled attributes on a part usage link, substitute link, reference designator link, and/or alternate link that are not mapped; documents not modified or documents not requested</td>
<td>GetPart GetBOM GetECN</td>
<td>Part or change ready to publish</td>
<td>WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>None, works as is</td>
</tr>
</tbody>
</table>
| Publish a WTPart with custom additional modeled attributes that are mapped; documents not modified or documents not requested | GetPart GetBOM GetECN | Part or change ready to publish | WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument | Windchill ESI Configuration:  
  - Modify map file(s)  
  - Modify XML schema¹⁰ |
| Publish a WTPart with custom additional modeled attributes on a substitute, alternate, or reference designator links that are mapped; documents not modified or documents not requested | GetPart GetBOM GetECN | Part or change ready to publish | WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument | Windchill ESI Configuration:  
  - Modify map file(s)  
  - Modify XML schema |

¹⁰ Although it is possible to change the XML schema by customization, it is safer to create a custom RPC so that future Windchill ESI maintenance will not overlay customizations.
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects(^6)</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTPart with extensions of substitute, reference designator, or alternate links, additional modeled attributes on the extensions that are not mapped; documents not modified or documents not requested</td>
<td>GetPart GetBOM GetECN</td>
<td>Part or change ready to publish</td>
<td>WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>Windchill ESI Configuration: * Modify ESI properties</td>
</tr>
<tr>
<td>Publish a WTPart with extensions of substitute, reference designator, or alternate links, additional modeled attributes on the extensions that are mapped; documents not modified or documents not requested</td>
<td>GetPart GetBOM GetECN</td>
<td>Part or change ready to publish</td>
<td>WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>Windchill ESI Configuration: * Modify map file(s) * Modify XML schema * Modify ESI properties</td>
</tr>
<tr>
<td>Publish a WTPart with custom additional modeled attributes on a part usage link that are mapped; documents not modified or documents not requested</td>
<td>GetPart GetBOM GetECN</td>
<td>Part or change ready to publish</td>
<td>WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>Windchill ESI Customization: * Extend java code(^1) * Modify map file(s) * Modify ESI properties</td>
</tr>
</tbody>
</table>

\(^1\) The Windchill BOM difference logic does not expose the part usage links directly. Instead it returns a set of wrapper objects of class, PartUsageInfo. If attributes are to be available to Windchill ESI mapping logic, PartUsageInfo must be modified. Otherwise, Windchill ESI java code must be modified to obtain the required attribute value(s).
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects⁶</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTPart after modifying the master relationship between WTPart and WTPartMaster¹²</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>Windchill ESI Customization:</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>WTPartMaster</td>
<td>• Extend ESI java code</td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td>• Modify ESI properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
<tr>
<td>Publish a custom object, MyPart, extended from WTPart and/or a custom object, MyPartMaster, extended from WTPartMaster with additional modeled or non-modeled attributes which are not mapped; documents not modified or documents not requested</td>
<td>GetPart</td>
<td>Part or change ready to publish</td>
<td>MyPart and/or</td>
<td>Windchill ESI Configuration:</td>
</tr>
<tr>
<td></td>
<td>GetBOM</td>
<td></td>
<td>MyPartMaster</td>
<td>• Modify ESI properties¹³</td>
</tr>
<tr>
<td></td>
<td>GetECN</td>
<td></td>
<td>WTDocument</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WTDocumentMaster</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EPMDocument</td>
<td></td>
</tr>
</tbody>
</table>

¹² Windchill WTPart objects are related to WTPartMaster objects by a complex relationship that is defined by the Windchill code generation tool. The relationship is central to product structure navigation and to the release of parts by Windchill ESI. Changing the relationships will disrupt Windchill ESI logic. These are the kinds of modifications that require java coding to repair the disruption of Windchill ESI logic:

- Replacing the existing relationship implementation with a link class
- Creating a new object, extending WTPart, that disrupts the inherited relationship to WTPartMaster
- Creating a new object, extending WTPartMaster, that disrupts the inherited relationship to WTPart

¹³ The properties define the name of the part class and the part master class. If these property values are changed, then it will only be possible to release the extension objects identified by the values. The GetPart RPC will no longer support WTPart and/or WTPartMaster.
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects⁶</th>
<th>Procedure Applied</th>
</tr>
</thead>
</table>
| Publish a custom object, MyPart, extended from WTPart and/or a custom object, MyPartMaster, extended from WTPartMaster with additional modeled or non-modeled attributes which are mapped; documents not modified or documents not requested | GetPart GetBOM GetECN                               | Part or change ready to publish | MyPart and/or MyPartMaster WTDocument WTDocumentMaster EPMDocument               | Windchill ESI Configuration:  
  • Modify ESI properties¹⁴  
  • Modify map files  
  • Modify XML schema                                                                 |
| Publish a WTPart with documents requested and modeled attribute or IBA or soft-type has been added to WTDocument, WTDocumentMaster, and/or EPMDocument | GetPart GetBOM GetECN                               | Part or change ready to publish | MyPart and/or MyPartMaster WTDocument WTDocumentMaster EPMDocument               | Windchill ESI Configuration:  
  • Modify map files  
  • Modify ESI properties¹⁵  
  • Define XML schema                                                                 |
| Publish a custom object, MyPart, extended from WTPart and/or a custom object, MyPartMaster, extended from WTPartMaster with additional modeled or non-modeled attributes which are mapped | Custom¹⁶          | Part or change ready to publish | WTPart               | Windchill ESI Customization:  
  • Author a custom RPC  
  • Define XML Schema                                                                 |

¹⁴ The properties define the name of the part class and the part master class. If these property values are changed, then it will only be possible to release the extension objects identified by the values. The GetPart RPC will no longer support WTPart and/or WTPartMaster.

¹⁵ Declare the document soft type.

¹⁶ The Windchill ESI GetPart RPC will still support for WTPart and WTPartMaster in this case.
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTPart with documents requested and:</td>
<td>Custom&lt;sup&gt;18&lt;/sup&gt;</td>
<td>Part or change ready to publish</td>
<td>MyPart and/or MyPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>Windchill ESI Customization:</td>
</tr>
<tr>
<td>• Document class(es) extended&lt;sup&gt;17&lt;/sup&gt; or</td>
<td></td>
<td></td>
<td></td>
<td>• Customize Document Difference API</td>
</tr>
<tr>
<td>• Document links modified</td>
<td></td>
<td></td>
<td></td>
<td>• Extend ESI java code</td>
</tr>
<tr>
<td>Publish a WTPart with modified linkages to (or services supporting)</td>
<td>Custom</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>• Author a custom RPC</td>
</tr>
<tr>
<td>non-local mapped attributes or modified services supporting non-local</td>
<td></td>
<td></td>
<td></td>
<td>• Modify XML Schema</td>
</tr>
<tr>
<td>mapped attributes&lt;sup&gt;19&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Publish a WTPart with modified linkages (uses, substitute, or references)</td>
<td>Custom</td>
<td>Part or change ready to publish</td>
<td>WTPart</td>
<td>ESI Customization:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Author a custom RPC based on GetBOM and/or GetECN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Customize BOM Difference API</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Modify XML Schema</td>
</tr>
<tr>
<td>Publish a WTChangeOrder2</td>
<td>GetECN</td>
<td>Change ready to publish</td>
<td>WTChangeOrder2 WTPart WTPartMaster WTDocument WTDocumentMaster EPMDocument</td>
<td>None, works as is</td>
</tr>
</tbody>
</table>

<sup>17</sup> In this category extensions do not include soft types. Adding a new soft type document is a configuration option. See above

<sup>18</sup> GetPart, GetBOM and GetECN could be customized in this case, but such a customization would be vulnerable to future ESI DSU maintenance. A custom RPC is a safer choice.

<sup>19</sup> For example, modifying the part alternate relationship.
<table>
<thead>
<tr>
<th>Specific Use</th>
<th>RPC</th>
<th>Relevant Event</th>
<th>Windchill Objects</th>
<th>Procedure Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish a WTChangeOrder2 with additional attributes not mapped</td>
<td>GetECN</td>
<td>Change ready to publish</td>
<td>WTChangeOrder2 WTPart WTPartMaster WTDocumentMaster EPMDocument</td>
<td>None, works as is</td>
</tr>
<tr>
<td>Publish a WTChangeOrder2 with additional attributes mapped</td>
<td>GetECN</td>
<td>Change ready to publish</td>
<td>WTChangeOrder2 WTPart WTPartMaster WTDocumentMaster EPMDocument</td>
<td>Windchill ESI Configuration:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Modify map files</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Modify XML schema</td>
</tr>
<tr>
<td>Publish a custom object, MyObject, not extended from WTPart</td>
<td>Custom</td>
<td>MyObject ready to publish</td>
<td>MyObject</td>
<td>Windchill ESI Customization:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Author a custom RPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Modify XML Schema</td>
</tr>
<tr>
<td>Modify ESITarget</td>
<td>Custom</td>
<td>Any</td>
<td>Any</td>
<td>See Distribution Targets for details.</td>
</tr>
</tbody>
</table>

For the current release, documents are only published if they are related to a Windchill part. Documents published by Windchill ESI are obtained by Windchill ESI Services from the Windchill document-difference logic. See Document Attachment Differences for information about customizing the Windchill document-difference logic.
11.1.1.1 **Advantages**

- Provides greater solution flexibility without coding. This in turn reduces deployment cost and time.
- Provides a supported customization path for situations not covered by OOTB code. This is an enhancement over RTP.

11.1.1.2 **Constraints**

- Windchill ESI uses the Windchill Adapter Query-Object webject to obtain all the attributes of parts and part master objects, as well as the attributes of supported documents types. If the Query-Object webject does not return the attribute, that attribute is not available to ESI mapping logic.
- Windchill ESI BOM processing uses the Windchill Standard WTPart Service BOM difference calculation. The BOM difference customizations are required if:
  - New object types are included on a BOM
  - New relationships are relevant to BOM structure
  - Additional data elements are to be considered when determining whether a change has been made.
- ESI document processing uses the Windchill Standard WTPart Service part document difference calculation. The part document difference customizations are required if:
  - New links are relevant to the part/document relationship
  - Additional data elements are to be considered when determining whether a change has been made.
- Changing the existing relationships (associations) between existing Windchill objects always requires customization of Windchill ESI.
- Adding relationships between existing Windchill objects requires customization of ESI if the relationships, relationship attributes, or objects participating in the relationship are to be included in the data that is published by Windchill ESI.