Oracle® Documaker

Introduction to Enterprise Web Processing Services

version 1.2

Part number: E16256-01
March 2011
THIRD PARTY SOFTWARE NOTICES

This product includes software developed by Apache Software Foundation (http://www.apache.org/).

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

Copyright © 2000-2009 The Apache Software Foundation. All rights reserved.

This product includes software distributed via the Berkeley Software Distribution (BSD) and licensed for binary distribution under the Generic BSD license.

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

Copyright © 2009, Berkeley Software Distribution (BSD)

This product includes software developed by the JDOM Project (http://www.jdom.org/).

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

Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved.

This product includes software developed by the Massachusetts Institute of Technology (MIT).

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright © 2009 MIT

This product includes software developed by Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Copyright (c) 1995-2005 Jean-loup Gailly and Mark Adler
This product includes software developed by PJ Arends and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABILITY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHenever YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by Erwin Tratar. This source code and all accompanying material is copyright (c) 1998-1999 Erwin Tratar. All rights reserved.

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY. USE IT AT YOUR OWN RISK! THE AUTHOR ACCEPTS NO LIABILITY FOR ANY DAMAGE/LOSS OF BUSINESS THAT THIS PRODUCT MAY CAUSE.

This product includes software developed by Sam Leffler of Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

This product includes software developed by Guy Eric Schalnat, Andreas Dilger, Glenn Randers-Pehrson (current maintainer), and others. (http://www.libpng.org)

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage.

This product includes software components distributed by the Cryptix Foundation.

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

Copyright © 1995-2005 The Cryptix Foundation Limited. All rights reserved.
This product includes software components distributed by Sun Microsystems.

This software is provided "AS IS," without a warranty of any kind. ALLEXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANYIMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.

This product includes software components distributed by Dennis M. Sosnoski.

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

Copyright © 2003-2007 Dennis M. Sosnoski. All Rights Reserved

It also includes materials licensed under Apache 1.1 and the following XPP3 license

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

Copyright © 2002 Extreme! Lab, Indiana University. All Rights Reserved

This product includes software components distributed by CodeProject. This software contains material that is © 1994-2005 The Ultimate Toolbox, all rights reserved.

This product includes software components distributed by Geir Landro.

Copyright © 2001-2003 Geir Landro (drop@destroydrop.com) JavaScript Tree - www.destroydrop.com/hjavascripts/tree/version 0.96
Contents

Chapter 1, Overview

2 The Need for Web Services
3 Available Services and Methods of Use
5 Using SOAP
7 Using JSON
9 Choosing Between SOAP and JSON
11 EWPS Requirements

Chapter 2, Business Scenarios

16 Publishing a Quote Form from a Rating Application
17 Option 1: Mapping the Data Using Oracle Insurance Tools
19 Option 2: Resolving the Data Mapping Before the doPublish Request
24 Initiating an Issuance Process from a Rating Application
25 Option 1: Mapping the Data using Oracle Insurance Tools
27 Option 2: Resolving the Data Mapping Before the doPublish Request

Chapter 3, Common Schema Types

LibraryList 32
BusUnitsList 33
TemplateList 34
RecipientList 35
ComposeData 36
Props 37
CoreProperties 38
ImportFileType 39
Errors 40
Chapter 4, Available Web Services

doGetLibraries 48

doGetBusUnits 49

doGetTemplateList 50

doGetTemplateListData 52

doCreateFolder 53

doGetFolderList 55

doGetFolder 57

doModifyFolder 58

doDeleteFolder 60

doCallAPI 61

67 Configuring the Provider

doPublish 68

69 Returning a PDF File in a doPublish Response

70 Accessing a Workspace Definition File via a Web Service

Chapter 5, Additional Resources

74 SOAP

75 Web Services

75 References and Projects

76 Web Services Standards and Specifications

76 Other Resources

77 Web Services Description Language

78 Using the XML Configuration File
Appendix A, Using the Jmeter Test Script to Test EWPS

82  What is Jmeter
83  Using Jmeter
86  Running the Jmeter Test Script
Chapter 1

Overview

The need to produce customer information 24 hours a day, seven days a week has shifted a large percentage of publishing volume away from traditional batch processing to a real-time, customer-driven, business model.

Moreover, companies increasingly want to leverage the web to reach their customers and prospects, resulting in new requirements for scalability and reliability.

At Oracle Insurance, we recognize that organizations are changing how they do business, and we have come to the marketplace with technology and architecture in keeping with this significant market shift.

This chapter discusses the following topics:

- The Need for Web Services on page 2
- Available Services and Methods of Use on page 3
- EWPS Requirements on page 11
The Enterprise Web Processing Services (EWPS) framework offers functionality via a set of established and interoperable standards such as XML and web services. This allows a multitude of enterprise applications — including policy production and claims correspondence — to be designed and developed around a core functional infrastructure.

Oracle Insurance’s contract-first approach to design

So, what exactly is WSDL? WSDL stands for Web Services Description Language. WSDL is kind of an XML grammar for describing web services interfaces (available functions).

WSDL leverages XML schema to describe the basic types used by a web service and provides all sorts of additional information that frame the contract of the interface, including things like ports, bindings, and so on.

**NOTE**: For more information about WSDL, see Web Services Description Language on page 77.
EWPS provides access to the Oracle Insurance suite of publishing, composition, workflow, and content management engines. It enables third-party applications to build custom applications, tools, and services that leverage the full breadth of Oracle Insurance functionality.

EWPS emphasizes business value throughout the whole web services technology stack. This self-service model means you can use a multitude of essential mechanisms — WS-I SOAP interfaces for application integration, JSON for UI integration, or pre-packaged business parts for design-time integration — in any sort solution.

EWPS is available in an Apache Axis2 package for J2EE and designed to be hosted under most popular application servers such as Apache Tomcat (Java), IBM WebSphere (Java), and BEA WebLogic (Java).
Typical EWPS-enabled applications include:

- Self-service publishing solutions
- Document search utilities
- Composition and workflow systems
- Systems that embed publishing artifacts in their web pages
- Applications that assist users in creating various types of documents and forms

An EWPS-enabled application can present data in ways that best meet the needs of a particular business scenario.

EWPS supports these protocols:


- JSON (JavaScript Object Notation) See Using JSON on page 7 for more information.
**Using SOAP**

With the SOAP API, the request interface (called a proxy) contains business-object interfaces and stubs generated directly from a WSDL document that specifies the EWPS schema and service address.

The third-party application works with data in the form of object properties. It sends and receives the data by calling object methods. The auto-generated SOAP proxy handles the details of serializing/de-serializing the SOAP request from EWPS into objects that are easy to work with.

**Note:** The SOAP API is built on open standards like SOAP and WSDL. These standards are supported by a wide-range of development tools on a variety of platforms. For more information, see SOAP on page 74.

**Request:**

```xml
<?xml version="1.0" encoding="utf-8"?>
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <doGetTemplateListRequest xmlns="http://webservices.docucorp.com/ewps/schema/2005-12-01">
      <AuthUser>string</AuthUser>
      <LibraryId>string</LibraryId>
      <BusUnitsList>
        <Key1 id="string" package="string">
          <Key2 id="string" />
          <Key2 id="string" />
        </Key1>
        <Key1 id="string" package="string">
          <Key2 id="string" />
          <Key2 id="string" />
        </Key1>
      </BusUnitsList>
      <EffectiveDate>string</EffectiveDate>
      <Start>integer</Start>
      <MaxResults>integer</MaxResults>
      <NameQuery>string</NameQuery>
      <DescQuery>string</DescQuery>
      <ContentQuery>string</ContentQuery>
      <SortBy>string</SortBy>
      <ResponseGroup>
        <Response>string</Response>
        <Response>string</Response>
      </ResponseGroup>
    </doGetTemplateListRequest>
  </soap:Body>
</soap:Envelope>
```

**Response:**

```xml
<?xml version="1.0" encoding="utf-8"?>
```
  <soap:Body>
    <doGetTemplateListResponse xmlns="http://webservises.docucorp.com/ewps/schema/2005-12-01">
      <Result>Success</Result>
      <TemplateList>
        <Story StoryName="string" id="string">
          <Key1 id="string" package="string">
            <Key2 id="string"/>
          </Key1>
          <Description>string</Description>
          <Props>
            <Prop name="string">string</Prop>
          </Props>
        </Story>
        <Story StoryName="string" id="string">
          <Key1 id="string" package="string">
            <Key2 id="string"/>
          </Key1>
          <Required>boolean</Required>
          <Description>string</Description>
          <Props>
            <Prop name="string">string</Prop>
          </Props>
        </Story>
      </TemplateList>
      <RecipientList>
        <Recipient name="string">
          <Copies>integer</Copies>
          <Story StoryName="string" id="string"/>
        </Recipient>
        <Recipient name="string">
          <Copies>integer</Copies>
          <Story StoryName="string" id="string"/>
        </Recipient>
      </RecipientList>
      <StartIndex>integer</StartIndex>
      <EndIndex>integer</EndIndex>
      <TotalResults>integer</TotalResults>
      <SearchTime>string</SearchTime>
    </doGetTemplateListResponse>
  </soap:Body>
</soap:Envelope>

Sample SOAP request and response template
USING JSON

The JSON API works just like the SOAP API, except requests and responses are handled in JSON rather than XML. JSON is a lightweight data-interchange format based upon a subset of the JavaScript language.

NOTE: For an overview of JSON, including the various tools and techniques for working with JSON, go to this web site: http://www.json.org.

Request: (using JavaScript – actual request is an HTTP POST)
```
var request = {
    "LibraryId":"string",
    "Start":integer,
    "MaxResults":integer,
    "BusUnitsList":
    [{
        "Key2":
        [{
            "id":"string"
        }],
        "id":"string",
        "package":"string"
    }]
};
http://server/EWPS/DocumentService/<request.toJSONString()>  
```

Response:
```
{
    "TemplateList":
    [{
        "Key1":
        {
            "Key2":
            [{
                "id":"string"
            }],
            "id":"string",
            "package":"string"
        },
        "Required":boolean,
        "Description":"string",
        "Props":
        [{
            "name":"string",
            "Value":"string"
        }],
        "StoryName":"string",
        "id":"string",
        "alias":"string"
    },
    {
        "Key1":
    }
}
```
Sample JSON request and response:

```json
{
   "Key2": [
      {
         "id": "string"
      },
      "id": "string",
      "package": "string"
   },
   "Required": boolean,
   "Description": "string",
   "Props": [
      {
         "name": "string",
         "Value": "string"
      },
      "StoryName": "string",
      "id": "string",
      "alias": "string"
   },
   "RecipientList": [
      {
         "Copies": "string",
         "Story": [
            {
               "extracopies": integer,
               "StoryName": "string",
               "id": "string",
               "alias": "string"
            },
            {
               "extracopies": integer,
               "StoryName": "string",
               "id": "string",
               "alias": "string"
            }
         ],
         "name": "string"
      }
   ],
   "StartIndex": integer,
   "EndIndex": integer,
   "TotalResults": integer,
   "SearchTime": "0.031",
   "Result": integer
}
```
### Choosing Between SOAP and JSON

Both the SOAP API and JSON API share the same underlying schema, so the basic format of the input and output data is the same regardless of the API being used — both access the same EWPS functionality and data. As a result, you can use one or the other or both. It just depends on whichever approach works best for your situation.

Here is a complete listing of available services and a general overview of what they are used for:

<table>
<thead>
<tr>
<th>Service</th>
<th>An operation used to...</th>
<th>Expanded terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>doGetLibraries</td>
<td>Get a list of the possible form libraries available for collaborative authoring, composition, or publishing services.</td>
<td>Form Library – Config, MRL</td>
</tr>
<tr>
<td>doGetBusUnits</td>
<td>Get a list of candidate business-unit selection criteria for a particular library. This helps refine the document-selection process.</td>
<td></td>
</tr>
<tr>
<td>doGetTemplateList</td>
<td>Get a list of candidate forms available for collaborative authoring, composition, or publishing services.</td>
<td></td>
</tr>
<tr>
<td>doGetTemplateListData</td>
<td>Get the schema for a given template list, including details pertaining to the Story, StoryFragments and Field information for the template list selection.</td>
<td>For Documaker publishing, think of a story as a form and a story fragment as a section</td>
</tr>
<tr>
<td>doCreateFolder (ComposeData)</td>
<td>Create a remote folder for composition, collaborative authoring, or publishing from a list of story templates and field data (ComposeData).</td>
<td>For Documaker publishing, think of a folder as a WIP item, Future use for Documanage Archive Folders.</td>
</tr>
<tr>
<td>doCreateFolder (Import)</td>
<td>Create a remote folder for composition, collaborative authoring, or publishing from an extract file (Import) used with a set of pre-defined rules.</td>
<td></td>
</tr>
<tr>
<td>doGetFolderList</td>
<td>Get a list of Folders for a specific owner, search criteria, or both.</td>
<td>Think of owners as a specific user</td>
</tr>
<tr>
<td>doGetFolder</td>
<td>Retrieve the contents of a folder.</td>
<td></td>
</tr>
<tr>
<td>doModifyFolder</td>
<td>Add, remove, re-arrange, and generally modify the templates/data (ComposeData) and/or CoreProperties of a folder.</td>
<td></td>
</tr>
<tr>
<td>doDeleteFolder</td>
<td>Delete a folder from the application.</td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 1

#### Overview

<table>
<thead>
<tr>
<th>Service</th>
<th>An operation used to...</th>
<th>Expanded terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>doPublish (FolderId)</td>
<td>Publish a document from a pre-existing folder (FolderId). See DistributionOptions on page 42 for information pertaining to the various publishing and distribution options.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>doPublish (ComposeData)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Publish a document from a list of story templates and field data (ComposeData). See DistributionOptions on page 42 for information pertaining to the various publishing and distribution options.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>doPublish (Import)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Publish a document from an extract file (Import) with a set of pre-defined rules. See DistributionOptions on page 42 for information pertaining to the various publishing and distribution options.</td>
<td></td>
</tr>
</tbody>
</table>
The EWPS Java web application deployment supports Java Runtime Environment (JRE) version 1.5 or higher. For best results, use JRE version 1.6. This table shows you the version of Java you need for each supported platform:

<table>
<thead>
<tr>
<th>For this platform</th>
<th>You need this version of Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows (x86-32 and x86-64)</td>
<td></td>
</tr>
<tr>
<td>XP (SP 2)</td>
<td>Sun Java 5 or higher</td>
</tr>
<tr>
<td>Server 2003 (SP2)</td>
<td></td>
</tr>
<tr>
<td>Linux (x86-32 and x86-64) 32- and 64-bit kernels</td>
<td></td>
</tr>
<tr>
<td>SuSE Linux Enterprise Server (SLES) version 9.4 or higher</td>
<td>Sun Java 5 or higher or IBM Java 5 or higher</td>
</tr>
<tr>
<td>RedHat Enterprise Linux (RHEL) version 5.1 or higher</td>
<td></td>
</tr>
<tr>
<td>Sun Solaris (SPARC) 32-and 64-bit kernels</td>
<td></td>
</tr>
<tr>
<td>Solaris 9/SunOS 5.9 or higher</td>
<td>Sun Java 5 or higher</td>
</tr>
<tr>
<td>IBM AIX 5L pSeries (RISC) 32- and 64-bit kernels</td>
<td></td>
</tr>
<tr>
<td>version 5.2 TL 5200509</td>
<td>IBM Java 5 or higher</td>
</tr>
<tr>
<td>version 5.3 TL 5300007</td>
<td>IBM Java 6 or higher</td>
</tr>
</tbody>
</table>

Web application server

Whether running on Windows, Linux, Solaris, or AIX, you can use either of the following Java Web Application Servers:

- IBM WebSphere AS, version 6.1 or higher
- Tomcat version 5.5 or higher

**NOTE:** We test with both WebSphere and Tomcat. Other Java application servers should also work. For instance, the JBoss application server works with EWPS although JBoss may fall under Tomcat.

An EWPS Java web application deployment requires the following version of Docupresentment to process web services requests:

- Docupresentment (IDS) version 2.2 patch 04 or higher

**NOTE:** The EWPS WAR is only available in IDS version 2.2, patch 04 or higher on Windows IDS Server installers. For other platforms you must pull it out of the Windows installer. For more information about installing IDS, see the Internet Document Server Installation Guide.
EWPS communicates to Docupresentment using a message bus provider such as JMS, WebSphere MQ, or MSMQ. This table shows the possible message bus systems. Tested systems are indicated with a single asterisk (*).

<table>
<thead>
<tr>
<th>For this platform</th>
<th>You can use one of these message busses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows (x86-32 and x86-64)</td>
<td>HTTP/SOAP(*) (**)</td>
</tr>
<tr>
<td>• XP (SP 2)</td>
<td>MSMQ (*)</td>
</tr>
<tr>
<td>• Server 2003 (SP2)</td>
<td>IBM WebSphere MQ, version 5.3 or higher (*)</td>
</tr>
<tr>
<td>• Other JMS providers</td>
<td></td>
</tr>
<tr>
<td>Linux (x86-32 and x86-64) ****</td>
<td>HTTP/SOAP(*) (**)</td>
</tr>
<tr>
<td>• SuSE Linux Enterprise Server (SLES) version 9.4 or higher</td>
<td>IBM WebSphere MQ, version 5.3 or higher (*)</td>
</tr>
<tr>
<td>• RedHat Enterprise Linux (RHEL) version 5.1 or higher</td>
<td>ActiveMQ JMS (<em>) (</em>**).</td>
</tr>
<tr>
<td>• Other JMS providers</td>
<td></td>
</tr>
<tr>
<td>Sun Solaris (SPARC)</td>
<td>HTTP/SOAP(*) (**)</td>
</tr>
<tr>
<td>• Solaris 9/SunOS 5.9 or higher</td>
<td>IBM WebSphere MQ, version 5.3 or higher (*)</td>
</tr>
<tr>
<td>• Other JMS providers</td>
<td>ActiveMQ JMS (<em>) (</em>**).</td>
</tr>
<tr>
<td>IBM AIX 5L pSeries (RISC) 32-bit and 64-bit</td>
<td>HTTP/SOAP(*) (**)</td>
</tr>
<tr>
<td>• version 5.2 TL 5200-09</td>
<td>IBM WebSphere MQ, version 5.3 or higher (*)</td>
</tr>
<tr>
<td>• version 5.3 TL 5300-07</td>
<td>ActiveMQ JMS (<em>) (</em>**).</td>
</tr>
<tr>
<td>• Other JMS providers</td>
<td></td>
</tr>
</tbody>
</table>

(*) Tested providers
(**) This provider has not been tested under heavy concurrency and load.
(***) At the time of publication, we noticed problems with the ActiveMQ JMS during heavy load testing. We expect Apache to address these issues in the future. Contact Apache for more information.
(****) Other Linux distributions should work at kernel v2.6.11.4-21 or higher but have not been tested.
**NOTE:** To find the latest version of Documaker, Docupresentment, iDocumaker, iPPS, or EWPS, log into the following web site:

http://aru.us.oracle.com/

---

### EWPS Requirements

**Docupresentment, JRE, and Documaker versions**

This table shows, for each supported and tested platform, the version of Docupresentment, the corresponding Java Runtime Environment (JRE) for Docupresentment, and the version of Documaker necessary to support EWPS.

<table>
<thead>
<tr>
<th>For this platform</th>
<th>Docupresentment</th>
<th>JRE</th>
<th>Documaker</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microsoft Windows</strong> (x86-32 and x86-64)</td>
<td>Version 2.2, patch 04* or higher</td>
<td>Sun JRE v1.5**</td>
<td>Version 11.3, patch 05 or higher</td>
</tr>
<tr>
<td>• XP (SP 2)</td>
<td></td>
<td>Sun JRE v1.6</td>
<td></td>
</tr>
<tr>
<td>• Vista (SP 1)</td>
<td></td>
<td>IBM JRE v1.5**</td>
<td></td>
</tr>
<tr>
<td>• Server 2003 (SP2)</td>
<td></td>
<td>IBM JRE v1.6**</td>
<td></td>
</tr>
<tr>
<td><strong>Linux (x86-32 and x86-64)</strong>**</td>
<td>Version 2.2, patch 04* or higher</td>
<td>Sun JRE v1.5**</td>
<td>Version 11.3, patch 05 or higher</td>
</tr>
<tr>
<td>• SuSE Linux Enterprise Server (SLES) version 9.4 or higher</td>
<td></td>
<td>Sun JRE v1.6</td>
<td></td>
</tr>
<tr>
<td>• RedHat Enterprise Linux (RHEL) version 5.1 or higher</td>
<td></td>
<td>IBM JRE v1.5**</td>
<td></td>
</tr>
<tr>
<td><strong>Sun Solaris (SPARC)</strong></td>
<td>Version 2.2, patch 04* or higher</td>
<td>Sun JRE v1.5**</td>
<td>Version 11.3, patch 05 or higher</td>
</tr>
<tr>
<td>• Solaris 9/SunOS 5.9 or higher</td>
<td></td>
<td>Sun JRE v1.6</td>
<td></td>
</tr>
<tr>
<td><strong>IBM AIX 5L pSeries (RISC)</strong> 32-bit and 64-bit</td>
<td>Version 2.2, patch 04* or higher</td>
<td>IBM JRE v1.5**</td>
<td>Version 11.3, patch 05 or higher</td>
</tr>
<tr>
<td>• version 5.2 TL 5200-09</td>
<td></td>
<td>IBM JRE v1.6**</td>
<td></td>
</tr>
<tr>
<td>• version 5.3 TL 5300-07</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

* A Docupresentment Windows installation includes a bundled Sun J2SE JRE version 1.6 which is used by default by the Docupresentment server.

** Running the Docupresentment web services interface under JRE version 1.5 requires JAXB 2. See Downloading JAXB on page 14 for more information.

*** IBM’s J2SE JRE version 1.6 is only supported in Docupresentment version 2.2, patch 05 or higher and Documaker Shared Objects version 11.3, patch 06 or higher.

**** Other Linux distributions should work at kernel version 2.6.11.4-21 or higher but have not been tested.
JAXB provides a way to map XML and Java code and extend applications with XML and Web Services technologies. You must install and implement JAXB 2 to run the Docupresentment web services interface under JRE version 1.5.

Follow these steps to download the latest version of JAXB:

1. Go to the JAXB web site:
   https://jaxb.dev.java.net/

2. Click Download Now for the latest version of JAXB.

3. On the new page, click the Download the Binary link and save the ZIP file to your local machine.

4. Unzip the downloaded ZIP file into directories. There will be a \jaxb-ri directory with a \lib subdirectory.

5. From the \lib subdirectory, copy the jaxb-api.jar and jaxb-impl.jar files to the \lib directory of your Docupresentment installation.
Chapter 2

Business Scenarios

As mentioned earlier, Enterprise Web Processing Services (EWPS) provides a set of well-defined services that have been designed from the standpoint of functionality and business-use.

Here are some typical business scenarios and how you can use Enterprise Web Processing Services to address them:

- Publishing a Quote Form from a Rating Application on page 16
- Initiating an Issuance Process from a Rating Application on page 24
Suppose a carrier wants to use Oracle Insurance to produce a real-time quote form from their rating application. Some carriers may host an online rating application as an added benefit to their agents. Since an agent is providing sufficient data to get a quote, this data can be used to populate and publish the quote form.

How do you map the rating application data to the actual quote form? Should the customer be responsible for resolving all mapping prior to the web service call or should you use Oracle Insurance tools for mapping?

Using EWPS, you can accomplish this task using the doPublish web service two ways:

- **Use Oracle Insurance tools to do the mapping** — The carrier produces an Oracle Insurance standard XML or data extract file to be used as an import file.
  
  See Option 1: Mapping the Data Using Oracle Insurance Tools on page 17 for more information.

- **Mapping resolved prior to the doPublish request** — The carrier can get schema for the quote form via the doGetTemplateListData web service and then provide a ComposeData structure populated with data.
  
  See Option 2: Resolving the Data Mapping Before the doPublish Request on page 19 for more information.
**OPTION 1: MAPPING THE DATA USING ORACLE INSURANCE TOOLS**

- Rating data is mapped to the quote form using Oracle Insurance tools.
- Rating engine generates a standardized extract with data.
- Rating engine calls EWPS doPublish Web service with extract.
- Carrier disburses the quote form as necessary.

**Process flow diagram**

**Step 1** MAPPING. The rating data is mapped to the quote form using Oracle Insurance tools. A layout or copy-book of the extract feed is provided as a reference point.

**Step 2** EXTRACT. The rating engine generates a standardized extract file with data for the quote form.

**Step 3** doPUBLISH. The rating engine calls the EWPS doPublish web service with the extract data. For example, here is a sample request:
Step 4  DISTRIBUTION. The carrier distributes the quote form as necessary.
OPTION 2: RESOLVING THE DATA MAPPING BEFORE THE doPUBLISH REQUEST

Option 2: Carrier maps using internal tools

- Carrier uses the doGetTemplateListData web service to obtain field-level data for mapping purposes.
- Rating engine calls EWPS doPublish Web service with ComposeData.
- Carrier disburses the quote form as necessary.

Step 1: MAPPING. The carrier uses the doGetTemplateListData web service to get field-level information for mapping purposes. For example, here is a sample request:
...and the response:
Step 2 **DO PUBLISH.** The rating engine calls the EWPS doPublish web service with ComposeData. For example, here is a sample request:

```
<doGetTemplateListDataResponse
 xmlns="http://websites.services.documorp.com/ewps/schema/2005-12-01">
 <Result>Success</Result>
 <ComposeData>
  <Field name="QUOTATION" />
  <Field name="POLICY" />
  <Field name="TO: " />
  <Field name="FROM: " />
  <Field name="BIND" />
  <Field name="INSNAM" />
  <Field name="INSAD1" />
  <Field name="INSAD2" />
  <Field name="INSCTY" />
  <Field name="INSST" />
  <Field name="INSZIP" />
  <Field name="COMPANY LINE 1" />
  <Field name="EFFDATE" />
  <Field name="EXPDTE" />
  <!-- More fields omitted for brevity -->
  <Story StoryName="Quote Form" id="35">
   <Key1 id="AMERGEN">
    <Key2 id="QUOTE" />
   </Key1>
  </Story>
  <Description>Quote form</Description>
  <Props>
   <Prop name="OPTIONS"/>N</Prop>
  </Props>
  <StoryFragments>
   <StoryFragment FragmentName="quote">
    <Field name="TODAYS DATE" />
    <Field name="VOICE #" />
    <Field name="FAX #" />
    <Field name="COVERAGE" />
    <Field name="NOTES" />
    <Field name="NOTES #002" />
   </StoryFragment>
  </StoryFragments>
 </Story>
</ComposeData>
</doGetTemplateListDataResponse>
```
...and the response:
Step 3  **DISTRIBUTION.** The carrier distributes the quote form as necessary.
INITIATING AN ISSUANCE PROCESS FROM A RATING APPLICATION

Once a quote is bound, an underwriter initiates the issuance process. Typically, the assigned underwriter produces the quote and will know when the time is right to issue the policy, which means some human intervention is required for issuance.

So how do you know what forms are needed to issue the policy and how do you know how to map the data? Can the carrier manage the job of mapping/triggering the policy forms based on the type of quote and the data (using doGetTemplateListData) or do you use Oracle Insurance tools for mapping and form triggering?

You can accomplish this business scenario using multiple steps — use the EWPS doCreateFolder web service to create the transaction and use another application, such as a policy administration system or iDocumaker/iPPS to complete the issuance process.

Using EWPS, you can accomplish the task of creating the transaction using the doCreateFolder web service one of two ways:

- Using Oracle Insurance tools to map the data — The carrier produces an Oracle Insurance standard XML or data extract file to be used as an import file.
  
  See Option 1: Mapping the Data using Oracle Insurance Tools on page 25 for more information.

- Mapping resolved prior to doCreateFolder request — The carrier can get schema for the quote form via the doGetTemplateListData web service and then provide a ComposeData structure populated with data.

  See Option 2: Resolving the Data Mapping Before the doPublish Request on page 27 for more information.
OPTION 1: MAPPING THE DATA USING ORACLE INSURANCE TOOLS

Option 1: Mapping/triggersing via Skywire Software tools

- Policy forms and rating data are triggered/mapped to the policy using Skywire Software tools.
- Rating engine generates a standardized extract file with data.
- Rating engine calls EWPS doCreateFolder Web service with extract.
- Agent/Underwriter completes the issuance process as desired (doPublish can be used for manual issuance).

Process flow diagram

Step 1 **MAPPING.** The rating data is mapped to the policy forms using Oracle Insurance tools. A layout or copy-book of the extract feed is provided as a reference point for mapping and triggering.

Step 2 **EXTRACT.** The rating engine generates a standardized extract file with data for the quote form.

Step 3 **DO PUBLISH.** The rating engine calls the EWPS doCreateFolder web service with the extract data. For example, here is a sample request:
Step 4  **ISSUANCE.** The agent or underwriter completes the issuance as desired via their policy administration system or by using an application such as iPPS.
OPTION 2: RESOLVING THE DATA MAPPING BEFORE THE doPUBLISH REQUEST

Option 2: Carrier handles mapping/triggering using internal tools

- Carrier determines how to trigger various policy forms and uses doGetTemplateListData Web service to obtain field-level data for mapping.
- Rating engine calls EWPS doCreateFolder Web service with ComposeData.
- Agent/underwriter completes the issuance process as desired (doPublish can be used for manual issuance).

Process flow diagram

Step 1  MAPPING. The carrier uses the doGetTemplateListData web service to get field-level information for mapping and triggering purposes. For example, here is a sample request:
<doGetTemplateListDataRequest>
  <AuthUser></AuthUser>
  <LibraryId>100AIC</LibraryId>
  <TemplateList>
    <Story StoryName="A100 03-1997" id="2870026170">
      <Key1 id="AMERGEN" package="INTERLINE" />
      <Key2 id="INTERLINE" />
      <Description>Common Policy Dec - DC</Description>
      <Props>
        <Prop name="OPTIONS">RX</Prop>
      </Props>
    </Story>
    <Story StoryName="A101 03-1997" id="4">
      <Key1 id="AMERGEN" package="INTERLINE" />
      <Key2 id="INTERLINE" />
      <Description>Minimum Earned Premium Endt</Description>
      <Props>
        <Prop name="OPTIONS">RM</Prop>
      </Props>
    </Story>
  </TemplateList>
</doGetTemplateListDataRequest>

...and the response:
Step 2  **doCreateFolder.** The rating engine calls the EWPS doCreateFolder web service with ComposeData. For example, here is a sample request:

```
<doSetTemplateListDataResponse 
xmlns="http://webservices.docucorp.com/ewps/schema/2005-12-01"> 
  <Result>Success</Result> 
  <ComposeData> 
    <Field name="POLICY"/> 
    <Story StoryName="A100 03-1997" id="2870026170"> 
      <Key1 id="AMERGEN" package="INTERLINE"/> 
      <Key2 id="INTERLINE"/> 
      <Description>Common Policy Dec - DC</Description> 
      <Props> 
        <Prop name="OPTIONS">RMX</Prop> 
      </Props> 
      <StoryFragments/>
    </Story>
    <Story StoryName="A101 03-1997" id="4"> 
      <Key1 id="AMERGEN" package="INTERLINE"/> 
      <Key2 id="INTERLINE"/> 
      <Description>Minimum Earned Premium Endt</Description> 
      <Props> 
        <Prop name="OPTIONS">RM</Prop> 
      </Props> 
      <StoryFragments> 
        <StoryFragment FragmentName="a101c"> 
          <Field name="ERNPRM"/> 
          <Field name="PERCENT"/> 
        </StoryFragment> 
      </StoryFragments> 
    </Story>
  </ComposeData> 
</doSetTemplateListDataResponse>
```
Step 3 **ISSUANCE.** The agent or underwriter completes the issuance as desired via their policy administration system or by using an application such as iPPS.
Chapter 3

Common Schema Types

Here are the common schema types used throughout Oracle Insurance’s Enterprise Web Processing Solution (EWPS).

These schema may be part of a message contract in one or more instances:

- LibraryList on page 32
- BusUnitsList on page 33
- TemplateList on page 34
- RecipientList on page 35
- ComposeData on page 36
- Props on page 37
- CoreProperties on page 38
- ImportFileType on page 39
- Errors on page 40
- ResponseGroup on page 41
- DistributionOptions on page 42
- DistributionResults on page 44
**Libra*ryList**

The LibraryList group provides information about libraries and their respective publishing services.

Here is a sample group:

```xml
<LibraryList>
    <Library id="Amergen">
        <Service type="Entry" name="Entry"/>
        <Service type="WIP" name="Work in Process"/>
        <Service type="Archive" name="Archive"/>
    </Library>
    <Library id="DOCC">
        <Service type="Entry" name="Entry"/>
    </Library>
</LibraryList>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>A library the provider supports.</td>
</tr>
</tbody>
</table>
The BusUnitsList group provides information about business units that you can use to refine and filter the document selection process. Additional business unit refinement appears as nested lists of Key criteria (Key1, Key2, Key3, and so on).

Here is a sample group:

```xml
<BusUnitsList>
  <Key1 id="AMERGEN PACKAGE" package="AMERGEN PACKAGE">
    <Key2 id="CRIME"/>
    <Key2 id="INLAND MARINE"/>
    <Key2 id="LIABILITY"/>
    <Key2 id="PROPERTY"/>
    <Key2 id="MOTOR TRUCK CARGO"/>
  </Key1>
  <Key1 id="AMERGEN GL" package="GENERAL LIABILITY">
    <Key2 id="LIABILITY"/>
  </Key1>
  <Key1 id="AMERGEN PROPERTY" package="COMM'L PROPERTY">
    <Key2 id="PROPERTY"/>
  </Key1>
  <Key1 id="AMERGEN IM">
    <Key2 id="INLAND MARINE"/>
  </Key1>
  <Key1 id="AMERGEN MTC">
    <Key2 id="MOTOR TRUCK CARGO"/>
  </Key1>
  <Key1 id="AMERGEN AUTO">
    <Key2 id="AUTO"/>
  </Key1>
  <Key1 id="AMERGEN IM">
    <Key2 id="INLAND MARINE">
      <Key3 id="PWC"/>
      <Key3 id="OTHER"/>
    </Key2>
  </Key1>
</BusUnitsList>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>A company available for the library ID provided in the request payload.</td>
</tr>
<tr>
<td>Key2</td>
<td>A line of business available for Key1</td>
</tr>
</tbody>
</table>
**TEMPLATELIST**

The TemplateList group provides information about candidate templates (story) returned from a query or when filtering requests for a library.

Here is a sample group:

```xml
<TemplateList>
  <Story StoryName="Letter" id="1" alias=""%">
    <Required>Yes</Required>
    <Description>Customer Letter</Description>
  </Story>
  <Story StoryName="Bill Letter" id="2" alias="">
    <Required>No</Required>
    <Description>Bill Letter</Description>
  </Story>
  <Story StoryName="Bill Letter" id="2.1" alias="">
    <Required>No</Required>
    <Description>Bill Letter Duplicate</Description>
  </Story>
</TemplateList>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Story</td>
<td>A provider form.</td>
</tr>
</tbody>
</table>

**NOTE:** For a Documaker implementation, a *story* can be limited in scope and be considered to be roughly equivalent to a *form*. As part of a broader schema for future growth and functionality, a story can extend across multiple pages, and several stories can share a single page.

A story can encompass the entire contents of a document package, or it may include an individual block of content. Additionally, a story could be quite dynamic; appearing in blocks throughout a document. such as the first part on page 1, the second part on page 5, and so on.
**RecipientList**

The RecipientList group provides a way to associate recipients with story templates. The RecipientList is exclusive to composition services such as doGetTemplateList and the folder-oriented services.

Here is a sample group:

```xml
<RecipientList>
  <Recipient name="AGENT">
    <Story StoryName="Letter" id="1" alias="" extracopies="1"/>
    <Story StoryName="Bill Letter" id="2" alias="" extracopies="0"/>
  </Recipient>
  <Recipient name="HOME OFFICE">
    <Story StoryName="Letter" id="1" alias="" extracopies="0"/>
    <Story StoryName="Bill Letter" id="2" alias="" extracopies="0"/>
  </Recipient>
  <Recipient name="INSURED">
    <Story StoryName="Letter" id="1" alias="" extracopies="0"/>
    <Story StoryName="Bill Letter" id="2" alias="" extracopies="0"/>
  </Recipient>
</RecipientList>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient</td>
<td>A recipient listing the Story elements that it will receive for a document.</td>
</tr>
</tbody>
</table>
**COMPOSEDATA**

The ComposeData group (requests only) provides information about the user-entered data on a particular page of a document composition to be saved for stateful requests.

**NOTE:** Schema for FIELD attributes are primarily driven by the type of View that is returned, mostly via the attributes found at the field (such as INPUT) level.

Here is a sample group:

```xml
<ComposeData>
  <Field name="GlobalField" type="" required="True">data</Field>
  <Story StoryName="Letter" id="1" alias="">
    <Field name="StoryFieldField" type="">data</Field>
    <StoryFragments>
      <StoryFragment FragmentName="CPADR">
        <Field name="StoryFragmentField1" type="">Bob</Field>
        <Field name="StoryFragmentField2" type="">Main</Field>
      </StoryFragment>
      <StoryFragment FragmentName="CPBODY">
        <Field name="StoryFragmentField3" required="True">text here</Field>
      </StoryFragment>
    </StoryFragments>
  </Story>
</ComposeData>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>A global, Story, or StoryFragment level field.</td>
</tr>
<tr>
<td>Story</td>
<td>A form containing one or more StoryFragment elements.</td>
</tr>
</tbody>
</table>

**NOTE:** For a Documaker implementation, a StoryFragment can be considered to be similar to a section or image.
The Props group provides a generic structure for extended properties that are not native to base schema objects.

For example, a recipient might have extended property information for distribution addresses or a folder could have extended property information with its CoreProperties to handle application-specific attributes.

The following schema objects can have extended properties:

- Story
- Recipient
- Folder (CoreProperties)

Here is a sample group:

```xml
<Props>
  <Prop name="propertyname1">propertyvalue1</Prop>
  <Prop name="propertyname2">propertyvalue2</Prop>
  <Prop name="propertyname3">propertyvalue3</Prop>
  <Prop name="propertyname4">propertyvalue4</Prop>
  ...
</Props>
```
COREPROPERTIES

The CoreProperties group provides information about the core properties of a document or folder.

Here is a sample group:

```xml
<CoreProperties>
  <Library id="DOCUCORP"/>
  <Description>Past Due Notification</Description>
  <DocumentId>90125</DocumentId>
  <DocumentType>NB</DocumentType>
  <StatusCode>N</StatusCode>
  <EffectiveDate>2005-12-01</EffectiveDate>
  <Key1 id="AMERGEN PACKAGE" package="AMERGEN PACKAGE">
    <Key2 id="CRIME"/>
    <Key2 id="INLAND MARINE"/>
    <Key2 id="LIABILITY"/>
    <Key2 id="PROPERTY"/>
    <Key2 id="MOTOR TRUCK CARGO"/>
  </Key1>
  <Props>
    <Prop name="RECNUM">66421AER7</Prop>
  </Props>
</CoreProperties>
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>The library for the document.</td>
</tr>
<tr>
<td>Description</td>
<td>The description for the document.</td>
</tr>
<tr>
<td>DocumentId</td>
<td>The Key ID for the document.</td>
</tr>
<tr>
<td>DocumentType</td>
<td>The transaction or document type for the document.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>The status code for the document, such as W for Work in Progress.</td>
</tr>
<tr>
<td>EffectiveDate</td>
<td>The effective date for the document.</td>
</tr>
<tr>
<td>Key1</td>
<td>The company value for the document.</td>
</tr>
<tr>
<td>Key2</td>
<td>The line of business for the document.</td>
</tr>
<tr>
<td>Props</td>
<td>A set of properties that correspond to the fields in the index for the document.</td>
</tr>
</tbody>
</table>
The ImportFileType provides a generic structure for passing a chunk (file) of opaque data to a service as a base64Binary element. Note that data can be referenced as in-line data (location="ATTACH") or via a URL (location="URL").

Here is a sample group:

```xml
<Import>
  <ImportFile location="ATTACH" p5:contentType="text/xml" xmlns:p5="http://www.w3.org/2005/05/xmlmime">PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGl0IHRvb2wgZm9ybG...<p>
  </ImportFile>
  <ImportFile location="URL" p5:contentType="text/xml" xmlns:p5="http://www.w3.org/2005/05/xmlmime"> file://1.1.1.1/38ED0A22842449A49D921B7542D09EC0.XML </ImportFile>
</Import>
```

Oracle Insurance supports different contentType definitions, which you can use to provide EWPS with information about the type of file being sent. Here are the supported contentTypes and their meaning:

<table>
<thead>
<tr>
<th>contentTypes</th>
<th>Tells the system to treat the referenced file as</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/vnd.docucorp+xml</td>
<td>Oracle Insurance XML format</td>
</tr>
<tr>
<td>application/vnd.docucorp+v2</td>
<td>Oracle Insurance V2 format (PPS import)</td>
</tr>
<tr>
<td>application/vnd.docucorp+extract</td>
<td>A raw extract file.</td>
</tr>
</tbody>
</table>
The Errors group provides information about any errors or problems that occurred during a request.

Here is a sample group:

```xml
<Errors>
  <Error>
    <ErrorCode>String</ErrorCode>
    <DetailedMessage>String</DetailedMessage>
    <ErrorSource>String</ErrorSource>
    <Severity>Warning</Severity>
    <Remedy>String</Remedy>
    <Trace>String</Trace>
  </Error>
  <Error>
    <ErrorCode>String</ErrorCode>
    <DetailedMessage>String</DetailedMessage>
    <ErrorSource>String</ErrorSource>
    <Severity>Warning</Severity>
    <Remedy>String</Remedy>
    <Trace>String</Trace>
  </Error>
</Errors>
```
The ResponseGroup group, only used for web service requests, provides a way to specify one or more optional response groups as part of an overall web service response.

**NOTE:** This group is reserved for future use.

Here is a sample group:

```
<ResponseGroup>
  <Response>Group1</Response>
  <Response>Group2</Response>
  <Response>Group3</Response>
</ResponseGroup>
```
DISTRIBUTION OPTIONS

The DistributionOptions group provides information about publishing channels and the recipient-specific distribution options contained therein. You can handle each recipient differently with specific options (sample A) or bundled together as a group (sample B).

Here is a discussion of the parameters you can use:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copies</td>
<td>(Optional) Determines the number of copies desired for each recipient type. The default is one copy.</td>
</tr>
<tr>
<td>DocucorpArchive</td>
<td>(Optional) Determines if the transaction should be archived into the Docucorp Smart Archive.</td>
</tr>
<tr>
<td>Distribution Source</td>
<td>(Optional) Determines the source of the information driving the distribution. You can choose from Ad Hoc or Predefined. The default is Ad Hoc.</td>
</tr>
<tr>
<td>Priority</td>
<td>(Optional) Determines the publishing priority of the document. DEFERRED means that a true batch system handles all distribution and publish type specifications. Basically, the system saves the transaction to WIP with a Batch status code for the nightly process to pick it up and print it. For DEFERRED, a generic DistributionResults is returned stating the status was Sent. REALTIME means the publishing system will be executed immediately and will tell you what happened during the print process. If REALTIME is used, a detailed listing of the output is returned via the DistributionResults complex type. The default is REALTIME.</td>
</tr>
<tr>
<td>PublishType</td>
<td>(Optional) Determines the type of document. XER (Metacode), AFP, PCL, PXL (PCL6), PDF, RTF, HTML, PST (PostScript), BPD (TIFF or other bitmap), TXT (line print), GDI (Windows print), XMP (XML output), VIPP (Xerox flavor of PostScript), V2 (Standard Export File). The default is PDF.</td>
</tr>
</tbody>
</table>
Here is sample group A — a simple example with Predefined source:

```xml
<DistributionOptions source="PREDEFINED">
  <Priority>REALTIME</Priority>
</DistributionOptions>
```

Here is sample group B — a simple example with Ad Hoc source:

```xml
<DistributionOptions source="ADHOC">
  <Channel>
    <Recipient name="ALLRECIPS"/>
  </Channel>
</DistributionOptions>
```

Here is sample group C — a simple example for Preview:

```xml
<DistributionOptions source="ADHOC">
  <Channel>
    <Recipient name="RECIP"/>
    <Preview>True</Preview>
  </Channel>
</DistributionOptions>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistributionType</td>
<td><em>(Optional)</em> Determines channel of distribution. The default is Immediate Print. Not valid if the Distribution Source is Predefined. • Immediate Print • Deferred Print (a scheduled or nightly batch)</td>
</tr>
<tr>
<td>Disposition</td>
<td><em>(Optional)</em> Determines how the document should be returned. You can choose from: • URL (for a file location) • ATTACH (for an attachment) See table below for defaults for each distribution type.</td>
</tr>
<tr>
<td>Preview</td>
<td><em>(Optional)</em> Determines if the document should be generated as a <em>Print</em> or <em>Template</em> preview.</td>
</tr>
<tr>
<td>Stories</td>
<td><em>(Optional)</em> Used to determine which story parts and extra copies are published for each recipient. Not valid with a distribution type of <em>Batch</em>. The default is all story parts.</td>
</tr>
<tr>
<td>Recipient</td>
<td><em>(Optional)</em> Describes the recipients requested for print. To let the form set/extract and publishing system handle all default recipients, do not specify any recipients in this structure. When you use a predefined distribution source, you can use a specific recipient to override the disposition generated from the predefined system.</td>
</tr>
</tbody>
</table>
The DistributionResults group provides read-only information about the recipient-specific published document results. You can handle each recipient differently with specific options (sample A), bundle them as a group (sample B), or group them in any combination.

This group includes these parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Story</td>
<td>This is the list of story templates that contains data (fields and StoryFragments) that comprise the forms.</td>
</tr>
<tr>
<td>PublishType</td>
<td>(Optional) Determines type of document. The default is PDF.</td>
</tr>
<tr>
<td>Documents</td>
<td>A listing of published documents as base64Binary elements or URL references to external files.</td>
</tr>
<tr>
<td>DocumentStatus</td>
<td>This indicates the status of the resulting document:</td>
</tr>
<tr>
<td></td>
<td>• Failure - The document had some failure and detailed information can be found in the Error object.</td>
</tr>
<tr>
<td></td>
<td>• Sent - Indicates the Document was sent to the DistributionType of Deferred Print, FAX, or Email.</td>
</tr>
<tr>
<td></td>
<td>• URL - A URL reference to the document.</td>
</tr>
<tr>
<td></td>
<td>• ATTACH - Indicates that the published document will be sent back as a 64-bit code encryption. These are the same encryptions found in Docupresentment send/receive message files. You must use a base-64 decoder to view the attachment.</td>
</tr>
</tbody>
</table>

Here is sample group A — a simple example with a pre-defined source:

```xml
<DistributionResults source="PREDEFINED">
  <Channel>
    <Recipient name="RECIP" id="1">
      <Props>
        <Prop name="RECIP_NAME1">Andy Jones</Prop>
      </Props>
    </Recipient>
    <PublishType>PDF</PublishType>
    <DistributionType>Immediate Print</DistributionType>
    <Documents>
      <Document status="URL">file://\myserver\documents\23480283423408098.pdf</Document>
    </Documents>
  </Channel>
  <Channel>
    <Recipient name="RECIP" id="2">
      <Props>
        <Prop name="RECIP_NAME1">Don Rogers</Prop>
      </Props>
    </Recipient>
    <PublishType>PDF</PublishType>
    <DistributionType>Immediate Print</DistributionType>
    <Documents>
```

Here is sample group B — a simple example with Ad Hoc source:

<DistributionResults source="ADHOC">
  <Channel>
    <Recipient name="ALLRECIPS"/>
    <PublishType>PDF</PublishType>
    <DistributionType>Immediate Print</DistributionType>
    <Documents>
      <Document status="ATTACH" p5:contentType="application/pdf" xmlns:p5="http://www.w3.org/2005/05/xmlmime">PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLT…</Document>
    </Documents>
  </Channel>
</DistributionResults>

Here is sample group C — a simple example for Preview:

<DistributionResults source="ADHOC">
  <Channel>
    <Recipient name="RECIP" id="1"/>
    <PublishType>PDF</PublishType>
    <Documents>
      <Document status="ATTACH" p5:contentType="application/pdf" xmlns:p5="http://www.w3.org/2005/05/xmlmime">PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLT…</Document>
    </Documents>
  </Channel>
</DistributionResults>
NOTE: Distribution options must be configured in the Documaker system to properly product the distribution channels provided in the request. See the Documaker System Reference for more information.
Chapter 4

Available Web Services

There are several web services available with EWPS. This chapter describes these services.

You can use the following composition services:

- doGetLibraries on page 48
- doGetBusUnits on page 49
- doGetTemplateList on page 50
- doGetTemplateListData on page 52
- doCreateFolder on page 53
- doGetFolderList on page 55
- doGetFolder on page 57
- doModifyFolder on page 58
- doDeleteFolder on page 60

You can use this service for composition or publishing:

- doCallAPI on page 61

You can use this service for publishing:

- doPublish on page 68

In addition, you can set up a web service to send Studio resource information, in the form of a WDF file, to the Documaker Add-In for Microsoft Word. For more information, see Accessing a Workspace Definition File via a Web Service on page 70.
**doGetLibraries**

Use this service to get a simple list of candidate document libraries available for publishing services.

This web service is non-stateful in nature and accepts optional user identification for library profiling purposes.

**Scenario**

<table>
<thead>
<tr>
<th>Message style</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
</table>

**Message types**

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>AuthUser</td>
<td>Optional user identification.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>• LibraryList</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Result</td>
<td>Returns <em>Success</em> or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td></td>
<td>LibraryList</td>
<td>A list of the available libraries.</td>
<td>LibraryList</td>
</tr>
<tr>
<td>Fault</td>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
**doGetBusUnits**

Use this service to get a list of candidate business unit (BU) selection criteria for a particular library that helps refine the document selection process.

This web service is non-stateful in nature and accepts optional user identification for library profiling purposes.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

**Operation/Message types**

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>AuthUser</td>
<td>Optional user identification.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>EffectiveDate</td>
<td>Optional date qualifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BusUnitsList</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Result</td>
<td>Returns Success or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td></td>
<td>BusUnitsList</td>
<td>A list of available business-unit groupings to refine transaction selection.</td>
<td>BusUnitsList</td>
</tr>
<tr>
<td>Fault</td>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
**doGetTemplateList**

Use this service recursively to get a list of the candidate forms available for publishing services. Use the Start and MaxResults parameters to specify where the template listing should start and how many records are returned. The NameQuery, DescQuery, and ContentQuery parameters filter the results by their form name, description, in-line contents, or any combination thereof.

This web service is non-stateful in nature and accepts optional user identification for library profiling purposes.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

**Operation/Message types**
The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>AuthUser</td>
<td>Optional user identification.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>BusUnitsList</td>
<td>A list of selected business-unit groupings to refine transaction selection.</td>
<td>BusUnitsList</td>
</tr>
<tr>
<td></td>
<td>EffectiveDate</td>
<td>Optional date qualifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>One-based index of the first desired result.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>MaxResults</td>
<td>Number of results desired per query.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>NameQuery</td>
<td>Use to refine the search by template name. You can use all or part of this parameter in the query.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>DescQuery</td>
<td>Use to refine the search by template description. You can use all or part of this parameter in the query.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ContentQuery</td>
<td>Search string for content matches within a template.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>SortBy</td>
<td>Optional parameter to specify how the list you retrieve is ordered. You can sort the list by: • NAME • DESCRIPTION</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including • TemplateList:</td>
<td>ResponseGroup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response</th>
<th>Result</th>
<th>Returns <em>Success</em> or an error message</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td></td>
<td>TemplateList</td>
<td>A list of the available story templates.</td>
<td>TemplateList</td>
</tr>
<tr>
<td></td>
<td>StartIndex</td>
<td>The index (1-based) of the first search result in TemplateList.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>EndIndex</td>
<td>The index (1-based) of the last search result in TemplateList</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>TotalResults</td>
<td>The total number of results that exist for the search request.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>SearchTime</td>
<td>The total amount of time the service took to complete the search in seconds.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td><strong>Fault</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
<td></td>
</tr>
<tr>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
<td></td>
</tr>
</tbody>
</table>
### doGetTemplateListData

Use this service to get schema for a given TemplateList, including details about the Story, StoryFragments, and Fields information for the TemplateList. This service is useful if you want to map field-level data to a document package.

The ComposeData type in the response contains a full aggregate of schema for each story in the TemplateList, which lets you interrogate any portion of the schema for varying types and scope of elements.

This web service is non-stateful in nature and accepts optional user identification for library profiling purposes.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>AuthUser</td>
<td>Optional user identification.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>TemplateList</td>
<td>A list of the available story templates.</td>
<td>TemplateList</td>
</tr>
<tr>
<td></td>
<td>EffectiveDate</td>
<td>Optional date qualifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ComposeData</td>
<td></td>
</tr>
</tbody>
</table>

| Response                         | Result        | Returns **Success** or an error message.                                   | String     |
|                                  | Errors        | A list of the errors returned if the request completed, but not 100% successfully. | ErrorList  |
|                                  | ComposeData   | A fragment of the schema for the selected list of templates – use to map data for downstream composition and publishing services. | ComposeData |

| Fault                            |               |                                                                            |            |
|                                  | BadRequest    | An exception because of a bad request or malformed parameters.             | Client     |
|                                  | ServiceException | An exception because of server problem or configuration.                   | Server     |
doCreateFolder

Use this service to create a remote folder of selected story templates you want to work on. A remote folder works somewhat like an e-Commerce shopping cart, except it is designed for managing an active document package with stateful composition.

The doCreateFolderRequest service is considered abstract in nature, which means it cannot be implemented. Instead, doCreateFolder supports the implementation of these underlying concrete types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Use this type if you...</th>
</tr>
</thead>
<tbody>
<tr>
<td>doCreateFolder_Import</td>
<td>Want Oracle Insurance rules to handle the dynamic triggering of Story and StoryFragment types and mapping of data to the document. (&lt;doCreateFolderRequest xsi:type=&quot;doCreateFolderReq_Import&quot;...))</td>
</tr>
<tr>
<td>doCreateFolder_ComposeData</td>
<td>Know which story templates you need and (optionally) want to map specific data elements to the document. (&lt;doCreateFolderRequest xsi:type=&quot;doCreateFolderReq_ComposeData&quot;...))</td>
</tr>
</tbody>
</table>

This web service is stateful in nature and returns a unique FolderId to be used in subsequent requests.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Request/Response using SOAP over HTTP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message types</th>
<th>The following operation/message types should be supported and follow the synchronous request/response scenario:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Abstract Request</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>doCreateFolderRequest</td>
<td>Owner</td>
<td>Identity of the document or folder owner, for identification purposes.</td>
<td>Owner</td>
</tr>
<tr>
<td>CoreProperties</td>
<td>CoreProperties</td>
<td>Core properties of the folder.</td>
<td>CoreProperties</td>
</tr>
<tr>
<td>ResponseGroup</td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including: • TemplateList • CoreProperties</td>
<td>ResponseGroup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Typed Request</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>doCreateFolderReq_Import</td>
<td>ImportFile</td>
<td>Attachment data for the import files that drives the publishing request</td>
<td>ImportFileList</td>
</tr>
<tr>
<td>TemplateList</td>
<td>TemplateList</td>
<td>Optional listing of selected story templates.</td>
<td>TemplateList</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Typed Request</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>doCreateFolderReq_ComposeData</td>
<td>ComposeData</td>
<td>A list of selected story templates with composition data to be merged with the active document for composition or publishing.</td>
<td>ComposeData</td>
</tr>
</tbody>
</table>

<p>| Response                       |                                       | |</p>
<table>
<thead>
<tr>
<th>doCreateFolderResponse</th>
<th>Result</th>
<th>Returns <em>Success</em> or an error message.</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td></td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td>FolderId</td>
<td></td>
<td>Unique folder identifier.</td>
<td>String</td>
</tr>
<tr>
<td>TemplateList</td>
<td></td>
<td>A list of all story templates currently in the remote folder.</td>
<td>TemplateList</td>
</tr>
<tr>
<td>CoreProperties</td>
<td></td>
<td>The core properties of the folder.</td>
<td>CoreProperties</td>
</tr>
<tr>
<td>Fault</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BadRequest</td>
<td></td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td>ServiceException</td>
<td></td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
doGetFolderList

Use this service to get a list of folders. This web service is stateful in nature and requires a FolderId to maintain the state of the request.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

**Operation/Message types**

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>doGetFolderListRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Owner</td>
<td>Identity of the document or folder owner, for identification purposes.</td>
<td>Owner</td>
</tr>
<tr>
<td></td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>One-based index of the first desired result.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>MaxResults</td>
<td>Number of results desired per query.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>DocumentIdQuery</td>
<td>Use to refine the search by DocumentId. You can use all or part of this parameter in the query.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>DescQuery</td>
<td>Use to refine the search by Description. You can use all or part of this parameter in the query.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>PropQuery</td>
<td>Use to refine the search by one or more custom properties. You can use all or part of this parameter in the query.</td>
<td>PropQueryInfo</td>
</tr>
<tr>
<td></td>
<td>SortBy</td>
<td>Optional parameter to specify how to order the list you retrieve. You can sort the list by:</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TemplateList</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Composedata</td>
<td></td>
</tr>
</tbody>
</table>

**Response**

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>doGetFolderListResponse</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Result</td>
<td>Returns <em>Success</em> or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td></td>
<td>FolderList</td>
<td>A list of folders returned from the search query.</td>
<td>FolderListType</td>
</tr>
<tr>
<td></td>
<td>StartIndex</td>
<td>The index (1-based) of the first search result in FolderList.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>EndIndex</td>
<td>The index (1-based) of the last search result in FolderList.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>TotalResults</td>
<td>The total number of results that exist for the search request.</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>SearchTime</td>
<td>The total amount of time the service took to complete the search in seconds.</td>
<td>String</td>
</tr>
</tbody>
</table>

**Fault**
<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
**doGetFolder**

Use this service to get the contents of a pre-existing folder.

This web service is stateful in nature and requires a FolderId to maintain the state of the request.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

**Operation/Message types** The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doGetFolderRequest</td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>FolderId</td>
<td>Unique folder identifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TemplateList</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CoreProperties</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Composedata</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doGetFolderResponse</td>
<td>Result</td>
<td>Returns Success or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td></td>
<td>Owner</td>
<td>Identity of the document or folder owner, for identification purposes.</td>
<td>Owner</td>
</tr>
<tr>
<td></td>
<td>FolderId</td>
<td>Unique folder identifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>TemplateList</td>
<td>A list of all story templates currently in the remote folder.</td>
<td>TemplateList</td>
</tr>
<tr>
<td></td>
<td>CoreProperties</td>
<td>The core properties of the folder.</td>
<td>CoreProperties</td>
</tr>
<tr>
<td>Fault</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
doModifyFolder

Use this service to add, remove, re-arrange, and generally modify the contents, XML data, and/or general information of a folder.

Here are the rules that apply to the use of doModifyFolder:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoreProperties</td>
<td>Any property item omitted means the pre-existing property in the folder remains intact. Additionally, any property passed as a blank/empty value indicates the pre-existing property in the folder should be cleared — subject to certain underlying publishing rules, wherein the clearing of a property would effectively invalidate the folder in the system.</td>
</tr>
<tr>
<td>ComposeData</td>
<td>If ComposeData is included in the request, it means there is an intention to update/modify the document itself. If ComposeData is included and there is a mismatch between Story items, ComposeData determines the new document packaging (overwrites the current document). Additionally, if a field item is included in the request, it means there is an intention to update/modify the same field in the folder. Conversely, omitting field items in the request indicates there is an intention to preserve the current content of the field in the folder.</td>
</tr>
</tbody>
</table>

This web service is stateful in nature and requires a FolderId to maintain the state of the request.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

The following operation/message types should be supported and the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>doModifyFolderRequest</td>
<td>Owner Identity of the document or folder owner, for identification purposes.</td>
<td>Owner</td>
</tr>
<tr>
<td></td>
<td>Owner</td>
<td></td>
<td>Owner</td>
</tr>
<tr>
<td></td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>FolderId</td>
<td>Unique folder identifier.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>ComposeData</td>
<td>Optional fragment of composition data to be injected/merged with the active document for composition or publishing</td>
<td>ComposeData</td>
</tr>
<tr>
<td></td>
<td>CoreProperties</td>
<td>Modified core properties of the folder.</td>
<td>CoreProperties</td>
</tr>
<tr>
<td>Response</td>
<td>ResponseGroup</td>
<td>Optionally return certain response groups, including:</td>
<td>ResponseGroup</td>
</tr>
<tr>
<td></td>
<td>Result</td>
<td>Returns SUCCESS or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Type</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>FolderId</td>
<td>Unique folder identifier.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>TemplateList</td>
<td>A list of all story templates currently in the remote folder.</td>
<td>TemplateList</td>
<td></td>
</tr>
<tr>
<td>CoreProperties</td>
<td>The core properties of the folder.</td>
<td>CoreProperties</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fault</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
doDeleteFolder

Use this service to delete a folder.

This web service is stateful in nature and requires a FolderId to make the request.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>LibraryId</td>
<td>Required library selection (ID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>FolderId</td>
<td>Unique folder identifier.</td>
<td>String</td>
</tr>
<tr>
<td>Response</td>
<td>Result</td>
<td>Returns Success or an error message.</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Errors</td>
<td>A list of the errors returned if the request completed, but not 100% successfully.</td>
<td>ErrorList</td>
</tr>
<tr>
<td>Fault</td>
<td>BadRequest</td>
<td>An exception because of a bad request or malformed parameters.</td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td>ServiceException</td>
<td>An exception because of server problem or configuration.</td>
<td>Server</td>
</tr>
</tbody>
</table>
doCallAPI

Use this service operation to submit any request type to a provider such as Docupresentment. This service operation allows more flexibility than the other service operations discussed in this document by providing more abstraction of a request payload that can be submitted to a provider. For instance, you can submit and return any number of name/value pairs, collections, and attachments. This also means that unlike the other service operations discussed in this document, doCallAPI does not lend itself well to the definition of a well-defined service. Only use this service when one of the other service operations does not provide the necessary functionality.

**NOTE:** This service operation is only supported in Java.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request payload</td>
<td>(doCallAPIRequest) elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>schemaVersion</td>
<td>The schema version to use. The default is 1.0, but version 1.1 can also be used.</td>
<td>schemaVersion</td>
<td>0..1</td>
</tr>
<tr>
<td>timeOut</td>
<td>The timeout value in seconds for the service operation to receive a response from the provider.</td>
<td>int</td>
<td>0..1</td>
</tr>
<tr>
<td>ProviderName</td>
<td>The provider name for the operation that should be invoked. The default is IDSProvider, which represents a Docupresentment server as a provider.</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>Operation</td>
<td>The provider operation that should be invoked. The default is processRequest but you can also provide a value of Discovery to ask the doCallAPI service operation to return a list of supported operations for the provider.</td>
<td>string</td>
<td>1</td>
</tr>
<tr>
<td>Props</td>
<td>A list of Prop, FileProp and Collection elements to send to the provider. See the definition of each for more details.</td>
<td>PropertyList</td>
<td>1</td>
</tr>
</tbody>
</table>
| Prop            | A name/value pair to send to the provider for a specific operation. You can submit a Prop name/value pair of name Discovery and value true to ask doCallAPI service operation to return additional information about the expected Prop elements for the Operation value provided. Here are the attributes:
  • name - The name of the name/value pair.
  • Text - The value of the name/value pair.
Here is an example:

  &lt;Prop name="Reqtype">SSS&lt;/Prop&gt; | PropertyInfo | 0..many      |
### Available Web Services

**FileProp**
A file attachment to send to the provider. The attachment can be inline base64 content or a valid HTTP or File URL to a file accessible by the doCallAPI service operation.

Here are the attributes:
- **name** - The name of the attachment.
- **location** - The location of the attachment. Use ATTACH if the content is provided as base64 inline text. Use URL if the content will be provided by an HTTP or File URL.
- **URLLocation** - Only present when the type is PropFile_URL and should contain the value of the HTTP or File URL.
- **contentType** - You can omit this value. It is only present because the base class used for PropFile_ATTACH and PropFile_URL types contains this attribute.
- **Text** - None (blank) when location is set equal to URL, otherwise, the base64 inline text content of the attachment when FileProp element is a child of Props element and none (blank) when it is a child of ResponseProps element.

Here is an example:
```xml
<FileProp xsi:type="contract:PropFile_URL" location="URL" URLLocation="file:///c:/test.xml" name="MyAttachment"/>
```

**Collection**
A collection of items to send to the provider. Each item can contain one or more columns with a name and value. You can also think of a collection as a row set with one or more rows, each row containing one or more name/value pairs.

Here are the attributes:
- **name** - The name of the collection
- **Text** - None (blank)

Here is an example:
```xml
<Collection name="My Collection">
  <Item name="My Item 1">
    <Column name="name1">value1</Column>
    <Column name="name2">value2</Column>
  </Item>
  <Item name="My Item 2">
    <Column name="name1">value1</Column>
    <Column name="name2">value2</Column>
  </Item>
</Collection>
```
### ResponseProps

An element that defines how attachments should be returned by the provider. As such, it contains one or more FileProp elements. See the definition of FileProp for more information.

Here is an example:

```xml
<ResponseProps>
  <FileProp xsi:type="contract:PropFile_ATTACH" location="ATTACH" name="ATC1"/>
  <FileProp xsi:type="contract:PropFile_URL" location="URL" URLLocation="file:///c:/test.xml" name="ATC2"/>
</ResponseProps>
```

### Response payload (doCallAPIResponse) elements

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Props</td>
<td>A list of Prop, FileProp and Collection elements returned by the provider.</td>
<td>PropertyList</td>
<td>1</td>
</tr>
<tr>
<td>Prop</td>
<td>A name/value pair returned by the provider.</td>
<td>PropertyInfo</td>
<td>0...many</td>
</tr>
<tr>
<td>FileProp</td>
<td>A file attachment returned by the provider. The attachment can be inline base64 content or a valid HTTP or file URL.</td>
<td>PropFile.Attach or PropFile.URL</td>
<td>0...many</td>
</tr>
</tbody>
</table>

### Attributes

- **name** - The name of the attachment.
- **location** - The location of the attachment. Will be ATTACH if the content is returned as base64 inline text, otherwise URL if the content is returned as an HTTP or file URL.
- **URLLocation** - Only present when the type is PropFile_URL and should contain the value of the HTTP or file URL.
- **contentType** - You can omit this value. It is only present because the base class used for PropFile_ATTACH and PropFile_URL types contains this attribute.
- **Text** - None (blank) when location is set equal to URL, otherwise, the base64 inline text content of the attachment returned by the provider.

Here is an example:

```xml
<FileProp xsi:type="contract:PropFile_URL" location="URL" URLLocation="file:///c:/test.xml" name="MyAttachment"/>
```
Here is an example request payload:

```xml
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <contract:doCallAPIRequest schemaVersion="1.0">
      <contract:timeOut>30</contract:timeOut>
      <contract:ProviderName>IDSProvider</contract:ProviderName>
      <contract:Operation>processRequest</contract:Operation>
      <contract:Props>
        <contract:Prop name="REQTYPE">ECH</contract:Prop>
        <contract:Prop name="Foo">Foo</contract:Prop>
        <contract:FileProp xsi:type="contract:PropFile_ATTACH" location="ATTACHMENT1"/>
      </contract:Props>
    </contract:doCallAPIRequest>
  </soap:Body>
</soap:Envelope>
```

Parameter | Description | Type | Occurrence
---|---|---|---
Collection | A collection of items returned by the provider. Each item can contain one or more columns with a name and value. You can also think of a collection as a row set with one or more rows, each row containing one or more name/value pairs.

Here are the attributes:

- name – The name of the collection
- Text – None (blank)

Here is an example:

```xml
<Collection name="My Collection">
  <Item name="My Item 1">
    <Column name="name1">value1</Column>
  </Item>
  <Item name="My Item 2">
    <Column name="name1">value1</Column>
    <Column name="name2">value2</Column>
  </Item>
</Collection>
```
Here is the corresponding response payload example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/Envelope/">
  <soapenv:Body>
    <doCallAPIResponse xmlns="http://webservices.docucorp.com/ewps/schema/2005-12-01">
      <Props>
        <Prop name="Foo">Foo</Prop>
        <Prop name="IDSGUID">ef87ae86d4d53eb5010b6791190894f1</Prop>
      </Props>
      <ResponseProps>
        <FileProp name="ATTACHMENT1" location="URL" URLLocation="file:///c:/temp/test1.xml"/>
        <FileProp name="ATTACHMENT2" location="ATTACH"/>
        <FileProp name="ATTACHMENT3" location="URL" URLLocation="file:///c:/temp/test2.xml"/>
      </ResponseProps>
    </doCallAPIResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
<soapenv:Body>
<doCallAPIResponse>
<Props>
<Prop name="IDSHOSTNAME">jrobertsnb1</Prop>
<Prop name="REQTYPE">ECH</Prop>
<Prop name="SERVERTIMESPENT">0.000</Prop>
<Prop name="SERVERTIMESPENTMS">0</Prop>
<Prop name="Timeout">30</Prop>
<Prop name="ServiceResults">SUCCESS</Prop>
<Prop name="ServiceTimeMillis">125</Prop>
</Props>
<URLLocation>alert://c:/temp/test1.xml</URLLocation/>
<FileProp name="ATTACHMENT1" location="URL"/>
<FileProp name="ATTACHMENT2" location="ATTACH">PPD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPza4NCjxtZXNzYWdlIl0KCTxkYXRhPg0KCSAgPHZhciBuYW1lPSJQRE5UZVNTU09SRCI+Rk9STUFhVTRU5UMSI+QzpcZG9jc2Vydlx0ZXN0ZmlsZXNcdGVzdC1jb25maWctZmlsZS54bWw8L3Zhcj4NCgk8L2RhdGE+DQoJPGF0dGFjaG1lbnRzPg0KCQk8ZmlsZSBuYW1lPSJBVFRBQ0hNRU5UMSI+QzpcZG9jc2Vydlx0ZXN0ZmlsZXNcdGVzdC1jb25maWctZmlsZS54bWw8L3Zhcj4NCgk8L2F0dGFjaG1lbnRzPgkNCjwvbWVzc2FnZT4NCg==</FileProp>
<FileProp name="ATTACHMENT3" location="URL"/>
<Collection name="Collection1">
<Item name="1">
<Column name="column1">value1</Column>
<Column name="column2">value2</Column>
</Item>
<Item name="2">
<Column name="name1">string1</Column>
<Column name="name2">string2</Column>
</Item>
</Collection>
<Collection name="Collection2">
<Item name="1">
<Column name="testname1">testvalue1</Column>
<Column name="testname2">testvalue2</Column>
</Item>
<Item name="2">
<Column name="myname1">stringvalue1</Column>
<Column name="myname2">stringvalue2</Column>
</Item>
</Collection>
<Props>
</Props>
</doCallAPIResponse>
</soapenv:Body>
</soapenv:Envelope>
Configuring the Provider

Each provider supported by the doCallAPI service operation contains a configuration section with the same name as that of ProviderName inside the ewps.config.xml configuration file. You can use this section to configure connection properties for each provider. For example, the IDSProvider section for Docupresentment contains the same configuration properties as those of the queue configuration properties for Docupresentment. Here is an example:

```xml
<IDSProvider>
  <entry name="marshaller.class">com.docucorp.messaging.data.marshaller.SOAPMIMEDSIMessageMarshaller</entry>
  <entry name="queuefactory.class">com.docucorp.messaging.http.DSIHTTPMessageQueueFactory</entry>
    <entry name="http.url">http://localhost:49152</entry>
    <entry name="http.reuse.ports">15</entry>
    <entry name="http.putmessage.tries">15</entry>
    <entry name="timeout">30</entry>
</IDSProvider>
```

The request payload can provide the configuration properties for the provider, overriding the properties defined in the ewps.config.xml configuration file. The configuration properties should be provided as a collection with the same name as that of ProviderName. Here is an example of a request payload that does that:

```xml
<?xml version="1.0" encoding="utf-8"?>
  <soap:Body>
    <contract:doCallAPIRequest schemaVersion="1.0">
      <contract:ProviderName>IDSProvider</contract:ProviderName>
      <contract:Operation>processRequest</contract:Operation>
      <contract:Props>
        <contract:Prop name="REQTYPE">SSS</contract:Prop>
        <contract:Prop name="CONFIG">DOCCDEMO</contract:Prop>
        <contract:Collection name="IDSProvider">
          <contract:Item name="Properties">
            <contract:Column name="marshaller.class">com.docucorp.messaging.data.marshaller.SOAPMIMEDSIMessageMarshaller</contract:Column>
            <contract:Column name="queuefactory.class">com.docucorp.messaging.http.DSIHTTPMessageQueueFactory</contract:Column>
            <contract:Column name="http.url">http://127.0.0.1:49152</contract:Column>
          </contract:Item>
        </contract:Collection>
      </contract:Props>
      <contract:ResponseProps>
        <contract:Column name="marshaller.class">com.docucorp.messaging.data.marshaller.SOAPMIMEDSIMessageMarshaller</contract:Column>
        <contract:Column name="queuefactory.class">com.docucorp.messaging.http.DSIHTTPMessageQueueFactory</contract:Column>
        <contract:Column name="http.url">http://127.0.0.1:49152</contract:Column>
      </contract:ResponseProps>
    </contract:doCallAPIRequest>
  </soap:Body>
</soap:Envelope>
```
doPublish

Use this service to publish a composed document, either from a stateful transaction or via an imported transaction.

A stateful request simply requires a FolderId, which is the identifier to handle a folder-based publishing request with pre-selected story templates in a ComposeData structure. Stateless requests can be driven by an import file (such as XML or a pre-defined extract) or the selection of desired story templates in a folder.

The doPublishRequest is considered abstract in nature, which means it cannot be implemented. Instead, doPublish supports the implementation of two underlying concrete types, as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Publishes a document from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>doPublish_Impo$$nt$$</td>
<td>An import file (doPublishRequest xsi:type=&quot;doPublishReq_Impo$$nt$$&quot;)</td>
</tr>
<tr>
<td>doPublish_Compose$$d$$</td>
<td>Composed data (doPublishRequest xsi:type=&quot;doPublishReq_Compose$$d$$Data&quot;)</td>
</tr>
<tr>
<td>doPublish_FolderId</td>
<td>A FolderId (doPublishRequest xsi:type=&quot;doPublishReq_FolderId&quot;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synchronous Request/Response using SOAP over HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>doc/literal</td>
</tr>
</tbody>
</table>

**Operation/Message types**

The following operation/message types should be supported and follow the synchronous request/response scenario:

<table>
<thead>
<tr>
<th>Message</th>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doPublishRequest</td>
<td>LibraryId</td>
<td>Optional library selection identifier</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(required for SourceType=IMPORT or FOLDERID).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DistributionOptions</td>
<td>Required grouping that specifies various publishing and distribution options.</td>
<td>DistributionOptions</td>
</tr>
<tr>
<td>Typed Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doPublishRequest_Import</td>
<td>SourceType</td>
<td>Fixed identifier for the type of publishing request (IMPORT).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Import</td>
<td>Attachment data for the import files that drives the publishing request.</td>
<td>ImportFileList</td>
</tr>
<tr>
<td>Typed Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doPublishRequest_FolderId</td>
<td>SourceType</td>
<td>Fixed identifier for the type of publishing request (FOLDERID).</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>FolderId</td>
<td>Unique identifier for the remote Folder</td>
<td>String</td>
</tr>
</tbody>
</table>
RETURNING A PDF FILE IN A DO_PUBLISH RESPONSE

Use the doPublishAttachment INI option in the CONFIG.INI file to enable a print stream produced by Documaker extract file processing to be returned in the EWPS doPublish response. With this option set to Yes, any input to doPublish, such as a Documaker standard XML file or an extract file, can be configured to return base64 attachments.

```xml
<IDSServer>
doPublishAttachment = Yes
</IDSServer>
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doPublishAttachment</td>
<td>Enter Yes to enable a print stream produced by Documaker extract file processing to be returned in the EWPS doPublish response. With this option set to Yes, any input to doPublish can be configured to return base64 attachments. The default is No.</td>
</tr>
</tbody>
</table>

**NOTE:** This INI option only affects doPublish processing with an import file, not ComposeData or FolderID. You must use this option with the EWPS doPublish disposition distribution option of ATTACH.
You can use a web service to provide a Documaker Workspace Definition file (WDF) to the Documaker Add-In for Microsoft Word. This file contains information about the workspace such as recipients, triggers, and fields which makes it easier for Add-In users to insert these objects into their documents.

The Documaker Add-in for Microsoft Word is pre-configured to get a WDF file with key information about the resource library. The Documaker Add-In for Microsoft Word User Guide explains how to set up the web service and make sure it is working properly.

These items must be in place to enable the web service to access the definition file:

- Docupresentment version 11.5 or higher must be installed and configured to use the workspace that will use the content created with the Add-in. Do this by adding the configuration name to the dap.xml file and creating a separate configuration file for the workspace, if it does not already exist.

- Put the GENDEFXML request type in the IDS configuration file (docserv.xml) to call the DPRGenerateDefinitionFile rule.

```xml
<section name="ReqType:GENDEFXML">
<entry name="function">atcw32->ATCLoadAttachment</entry>
<entry name="function">atcw32->ATCUnloadAttachment</entry>
<entry name="function">dprw32->DPRSetConfig</entry>
<entry name="function">dprw32->DPRGenerateDefinitionFile</entry>
</section>
```

**NOTE:** For more information on the DPRGenerateDefinitionFile rule, see Using the Documaker Bridge.

The request needs these input variables:

- **Config**
- **BDF Name**

Add-In users will enter these variables via the Documaker Add-in for Microsoft Word configuration option, see Downloading a Workspace Definition File in the Documaker Add-In for Microsoft Word User Guide for more information. To test, you can enter these variables via the DSICOTB test configuration tool. See the Internet Document Server SDK Reference for more information on this tool.

- Deploy EWPS to the application server via the ewps-axis2.war file. You can find this file at the root of the Docupresentment directory, docserv\webservices. Use the location of the deployed EWPS as the endpoint for configuring the web service connectivity.

Add-In users will enter these variables via the Documaker Add-in for Microsoft Word configuration option, see Downloading a Workspace Definition File in the Documaker Add-In for Microsoft Word User Guide for more information. To test, you can enter these variables via the DSICOTB test configuration tool. See the Internet Document Server SDK Reference for more information on this tool.
• Deploy EWPS to the application server via the ewps-axis2.war file. You can find this file at the root of the Docupresentment directory, docserv\webservices. Use the location of the deployed EWPS as the endpoint for configuring the web service connectivity.

• In the DocumakerWordAddin.dll.config file, change the following section from a setting of “None” to a “Transport,” as shown here:

  <security mode="Transport">
  
  The DocumakerWordAddin.dll.config file, is, by default, located in this directory:
  
  ..\Program Files\Oracle\DocumakerAddIn\n
• Verify your certificate with a trusted CA site. To verify the certificate, create a .csr or .cer file and import the file into the keystore file. Then submit the .csr or .cer file to a certification service, such as VeriSign. From that site you can verify or authenticate the certificate and then use that certificate to connect to the Add-In operating behind the Secure Sockets Layer (SSL).

**NOTE:** The Documaker Add-In does not accept self-signed certificates.
Chapter 5

Additional Resources

The following resources provide information about SOAP, JSON, and web services in general, as well as other useful topics:

- SOAP on page 74
- Web Services on page 75
- Web Services Description Language on page 77
- Using the XML Configuration File on page 78

The definitions within various sections of this document are taken from several of these resources.
SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols.

The framework has been designed to be independent of any particular programming model and other implementation specific semantics. SOAP supports message security, attachment, routing, reliability, and choreography.

W3C: [http://www.w3.org/TR/soap12-part1/](http://www.w3.org/TR/soap12-part1/)

SOAP Messaging Framework:

W3C: [http://www.w3.org/TR/SOAP](http://www.w3.org/TR/SOAP)

W3Schools SOAP Tutorial:

W3Schools: [http://www.w3schools.com/soap/default.asp](http://www.w3schools.com/soap/default.asp)
**WEB SERVICES**

Web services is a technology that lets applications communicate with each other in a platform- and programming language-independent manner.

A web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another web service.

Web services promise to increase interoperability and lower the costs of software integration and data-sharing with partners. As they are based on simple and non-proprietary standards, web services make it possible for computer programs to communicate directly with one another and exchange data regardless of location, operating systems, or languages.


**REFERENCES AND PROJECTS**

IBM developerWorks Web Services
IBM Corporation


O'Reilly Web Services
O'Reilly & Associates, Inc.

[http://webservices.oreilly.com](http://webservices.oreilly.com)

Microsoft Web Services
Microsoft Corporation

[http://msdn.microsoft.com/webservices](http://msdn.microsoft.com/webservices)

XML and Web Services
Microsoft Corporation


Java Technology and Web Services
Sun Microsystems, Inc.


Apache Web Services Project
The Apache Software Foundation

[http://ws.apache.org](http://ws.apache.org)

JSON
JSON.org

[http://www.json.org](http://www.json.org)
WEB SERVICES STANDARDS AND SPECIFICATIONS

Web Services Interoperability Organization
WS-I

http://www.ws-i.org

Web Services Activity
W3C

http://www.w3.org/2002/ws

OASIS

http://www.oasis-open.org/home/index.php

OTHER RESOURCES

Web Services Architect

http://www.webservicesarchitect.com

SOA World Magazine

http://webservices.sys-con.com

WebServices.org

http://www.webservices.org

Dr. Dobbs Journal – SOA, Web services, and XML

http://www.ddj.com/dept/webservices
Web Services Description Language (WSDL) is an XML-based service description on how to communicate using web services. The WSDL defines services as collections of network endpoints, or ports. WSDL specification provides an XML format for documents for this purpose.

The abstract definition of ports and messages is separated from their concrete use or instance. This allows the reuse of these definitions. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding, where the messages and operations are then bound to a concrete network protocol and message format. In this way, WSDL describes the public interface to the web service.

WSDL is often used with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.


Using WSDL in SOAP applications:


Understanding WSDL:


An overview of WSDL:


Apache Axis2 User Guide:

Apache: [http://ws.apache.org/axis2/1_1_1/userguide.html](http://ws.apache.org/axis2/1_1_1/userguide.html)
**Using the XML Configuration File**

EWPS uses a file to set up configuration options, including how it communicates with Docupresentment. The default behavior is to communicate with Docupresentment over HTTP, on port 49152, with EWPS and Docupresentment on the same machine.

There are several ways to have EWPS change where it looks for the configuration file. Based on your application server and clustering setup, you should select the approach that best fits your needs.

EWPS first looks for the following JVM system property:

```
ewps.config.url
```

If this is set, EWPS looks for the configuration file at the URL specified in the setting. If the configuration file is in a file on the local machine, use the `file://url` naming scheme. Here is an example:

```
file:///c:/configurations/ewps.config.xml
```

If the system property is not set or if there was an error retrieving the configuration file, EWPS searches its Java classpath for a file named:

```
ewps.config.xml
```

With some application servers, such as Tomcat, the file can be located in one of these EWPS directories...

- `/WEB-INF/classes` directory
- `/WEB-INF/lib` directory (if packaged in a JAR file)

Other application servers, such as WebSphere, have options to set up shared libraries that can be added to a web application's classpath but still remain external to the web application's deployment.

If none of these options find a configuration file, EWPS looks at the `init.file` context parameter in the web application's `web.xml` file. Some application servers allow access and the editing of this file directly after web application deployment, while others have administration consoles to allow the editing of parameters in the web application.

If the parameter is in the form of a URL, EWPS looks for the configuration file at the URL specified in the setting. If the parameter is not a URL, it is assumed to be a file in the context and file structure of the web application. The default value for the `init.file` context parameter is shown here:

```
/WEB-INF/xml/ewps.config.xml
```
The ewps.config.xml file contains the configuration for the message bus provider. This file is located in the ewps-axis2.war's WEB-INF/services/EWPSService.aar (Axis2 Archive – Java Archive format) file, under its root.

You can modify this file to define the different message bus providers that Docupresentment (IDS) is configured to listen on. The ewps.config.xml file contains configuration examples for the following:

- JMS/ActiveMQ
- WebSphere MQ
- MSMQ (Windows only)
- IDS HTTP

You can find these message bus configuration examples under the `<EWPS><Core><queuemanager>` section of the XML file. Here is an example of the ewps.config.xml file:

```xml
<EWPS>
  <Core>
    <queuemanager>
      <!-- MESSAGING and QUEUE nodes are used for setting communication to Docupresentment. Refer to Docupresentment documentation for possible values -->
      <entry name="queuefactory.class">com.docucorp.messaging.jms.DSIAJMSJNDIMessageQueueFactory</entry>
      <entry name="jms.inputqueue.connectstring">resultq</entry>
      <entry name="jms.outputqueue.connectstring">requestq</entry>
      <entry name="jms.qcf.name">queueConnectionFactory</entry>
      <entry name="jms.initial.context.factory">org.apache.activemq.jndi.ActiveMQInitialContextFactory</entry>
      <!-- ***Settings for IDS http connection*** -->
      <entry name="queuefactory.class">com.docucorp.messaging.http.DSIIHTTPMessageQueueFactory</entry>
      <entry name="http.url">http://localhost:49152</entry>
      <!-- ***Settings for WebSphereMQ connection*** -->
      <entry name="queuefactory.class">com.docucorp.messaging.mqseries.DSIMQMessageQueueFactory</entry>
      <entry name="mq.queue.manager">QM.server1</entry>
      <entry name="mq.inputqueue.name">resultq</entry>
      <entry name="mq.inputqueue.maxwaitseconds">5</entry>
      <entry name="mq.outputqueue.name">requestq</entry>
      <entry name="mq.tcpip.host">10.1.10.159</entry>
      <entry name="mq.tcpip.port">1415</entry>
      <entry name="mq.queue.channel">SCC1.server1</entry>
      <!-- ***Setings for MSMQ connection*** -->
      <entry name="queuefactory.class">com.docucorp.messaging.msmq.DSIMSMQMessageQueueFactory</entry>
      <entry name="msmq.server.name">localhost</entry>
      <entry name="msmq.inputqueue.name">DIRECT=TCP:10.1.10.178\private$\resultq</entry>
    </queuemanager>
  </Core>
</EWPS>
```
<entry name="msmq.outputqueue.name">DIRECT=TCP:10.1.10.178\private$\requestq</entry>
<entry name="msmq.timeout">30000</entry>
<entry name="msmq.expiry">1800000</entry>
<entry name="msmq.debuglevel">2</entry>
</queuemanager>
</Core>
</EWPS>
Appendix A

Using the Jmeter Test Script to Test EWPS

EWPS provides multiple service operations. Some of the service operations are abstract and encompass multiple implementations. Such a set of complex service operations can present a steep learning curve to new users of EWPS, but the EWPS Jmeter test script helps you get started by providing the following:

- A set of examples for each service request operation. These examples can help you more quickly implement your system.
- An interface which lets you inspect, modify, and expand test plans or create your own to establish EWPS functionality or test performance.
- A Docupresentment master resource library (MRL) that EWPS can use to invoke each service operation via the Jmeter test script.
- A way to do regression testing.

Also, the accompanying FSDMS2 library in Docupresentment shows a full working set by illustrating the library configuration on the Docupresentment side, which is used by EWPS.

NOTE: To use the Jmeter test script, you must have Docupresentment version 2.2, patch 08 or higher.
WHAT IS JMETRER

Jmeter is a user interface from Apache that you can use to test the performance and functionality of Web services as well as other endpoints. You can learn more about Jmeter at the following link:

http://jakarta.apache.org/jmeter/

A Jmeter script is a test module that contains one or more pre-recorded test cases. A script can contain assertions and validations for each test case and can be run in a single thread to test for functionality or in multiple threads to test performance.
**Using Jmeter**

To run the EWPS Jmeter test script, you must first download the latest version of Apache Jmeter from this site:

http://jakarta.apache.org/jmeter/

**NOTE:** Version 2.4 or greater is required to run the EWPS-Java-FSDMS2.jmx script.

The Docupresentment installation includes a Jmeter test script for testing the different EWPS service operations. You can find this script here:

Docupresentment_installation/jmeter-scripts/EWPS-Java-FSDMS2.jmx

Where `Docupresentment_installation` represents the location were Docupresentment was installed, such as:

- On Windows: `c:\docupresentment\docserv`
- On UNIX: `/home/docupresentment/docserv`

The EWPS Jmeter test script contains these test cases:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doGetLibraries</td>
<td>Tests the doGetLibraries service operation.</td>
</tr>
<tr>
<td>doGetBusUnits</td>
<td>Tests the doGetBusUnits service operation.</td>
</tr>
<tr>
<td>doGetTemplateList1</td>
<td>Tests the doGetTemplateList service operation. Returns a template list by matching the effective date.</td>
</tr>
<tr>
<td>doGetTemplateListData1</td>
<td>Tests the doGetTemplateListData service operation. Returns the template list data for the template list returned by the doGetTemplateList1 service request.</td>
</tr>
<tr>
<td>doGetTemplateList2</td>
<td>Tests the doGetTemplateList service operation. Returns a template list by matching the effective date and form name. The results are sorted by form name.</td>
</tr>
<tr>
<td>doGetTemplateListData2</td>
<td>Tests the doGetTemplateListData service operation. Returns the template list data for the template list returned by the doGetTemplateList2 service request.</td>
</tr>
<tr>
<td>doGetTemplateList3</td>
<td>Tests the doGetTemplateList service operation. Returns a template list by matching the effective date and form description. The results are sorted by form description.</td>
</tr>
<tr>
<td>doGetTemplateListData3</td>
<td>Tests the doGetTemplateListData service operation. Returns the template list data for the template list returned by the doGetTemplateList3 service request.</td>
</tr>
<tr>
<td>doCreateFolder1</td>
<td>Tests the doCreateFolder_Import service operation. Creates a folder (transaction) in WIP using a Documaker import file provided as a file URL.</td>
</tr>
<tr>
<td>Test Case</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>doGetFolder1</td>
<td>Tests the doGetFolder service operation. Retrieves the folder created by the doCreateFolder1 service request and checks, through assertions, the Key1, Key2, DocumentId, DocumentType, StatusCode, and Description core property values.</td>
</tr>
<tr>
<td>doDeleteFolder1</td>
<td>Tests the doDeleteFolder service operation. Deletes the folder created by the doCreateFolder1 service request.</td>
</tr>
<tr>
<td>doCreateFolder2</td>
<td>Tests the doCreateFolder_Import service operation. Creates a folder (transaction) in WIP using a Documaker import file provided as inline base64 content.</td>
</tr>
<tr>
<td>doGetFolder2</td>
<td>Tests the doGetFolder service operation. Retrieves the folder created by the doCreateFolder2 service request and checks the Key1, Key2, DocumentId, DocumentType, StatusCode, and Description core property values.</td>
</tr>
<tr>
<td>doDeleteFolder2</td>
<td>Tests the doDeleteFolder service operation. Deletes the folder created by the doCreateFolder2 service request.</td>
</tr>
<tr>
<td>doCreateFolder3</td>
<td>Tests the doCreateFolder_Import service operation. Creates a folder (transaction) in WIP using a Documaker import file provided as a file URL and a TemplateList element that was returned by the doGetTemplateList2 service request.</td>
</tr>
<tr>
<td>doGetFolder3</td>
<td>Tests the doGetFolder service operation. Retrieves the folder created by the doCreateFolder3 service request checks the Key1, Key2, DocumentId, DocumentType, StatusCode, and Description core property values.</td>
</tr>
<tr>
<td>doDeleteFolder3</td>
<td>Tests the doDeleteFolder service operation. Deletes the folder created by the doCreateFolder3 service request.</td>
</tr>
<tr>
<td>doCreateFolder4</td>
<td>Tests the doCreateFolder_ComposeData service operation. Creates a folder (transaction) in WIP using a ComposeData element returned by the doGetTemplateListData2 service request and a custom property named INS_PHONE.</td>
</tr>
<tr>
<td>doGetFolder4</td>
<td>Tests the doGetFolder service operation. Retrieves the folder created by the previous service request (doCreateFolder4) and checks the Key1, Key2, DocumentId, DocumentType, StatusCode, and Description core property values. Also checks the custom property INS_PHONE.</td>
</tr>
<tr>
<td>doPublish1</td>
<td>Tests the doPublish_FolderId service operation. Publishes an XML export file from a transaction (folder) ID in WIP using the ADHOC distribution option and returns the file as a file URL.</td>
</tr>
<tr>
<td>doPublish2</td>
<td>Tests the doPublish_FolderId service operation. Publishes an XML export file from a transaction (folder) ID in WIP using the ADHOC distribution option and returns the file as base64 inline content.</td>
</tr>
<tr>
<td>doPublish3</td>
<td>Tests the doPublish_ComposeData service operation. Publishes a PDF file from a ComposeData Element using the ADHOC distribution option and returns the file as base64 inline content.</td>
</tr>
<tr>
<td>Test Case</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>doPublish4</td>
<td>Tests the doPublish_Import service operation. Publishes a PDF file from an XML Import file using the PREDEFINED distribution option and returns the file as a URL.</td>
</tr>
<tr>
<td>doModifyFolder1</td>
<td>Tests the doModifyFolder service operation. Modifies the Key ID (DocumentId), the status code, the INS_PHONE custom Key, and the Description for a transaction (folder) in WIP.</td>
</tr>
<tr>
<td>doGetFolder5</td>
<td>Tests the doGetFolder service operation. Makes sure the values modified in the doModifyFolder1 service request present.</td>
</tr>
<tr>
<td>doModifyFolder2</td>
<td>Tests the doModifyFolder service operation. Modifies the field and form data for a transaction (folder) in WIP using a ComposeData element.</td>
</tr>
<tr>
<td>doGetFolder6</td>
<td>Tests the doGetFolder service operation. Makes sure the values modified in the doModifyFolder2 service request present.</td>
</tr>
<tr>
<td>doDeleteFolder4</td>
<td>Tests the doDeleteFolder service operation. Deletes the folder created by the doCreateFolder4 service request.</td>
</tr>
<tr>
<td>doGetFolderList</td>
<td>Tests the doGetFolderList service operation. Gets a folder list for the user, library, and other search criteria specified.</td>
</tr>
<tr>
<td>doCallAPI1</td>
<td>Tests the doCallAPI service operation. Sends an SSS request type to Docupresentment.</td>
</tr>
<tr>
<td>doCallAPI2</td>
<td>Tests the doCallAPI service operation. Sends an ECH request type with a 971,272 byte file attachment as base64 inline content to Docupresentment.</td>
</tr>
<tr>
<td>doCallAPI3</td>
<td>Tests the doCallAPI service operation. Sends an ECH request type with two file attachments. Demonstrates using the ResponseProps element to indicate how the response file attachments should be returned. One is returned as a file URL and the other is returned as inline base64 content.</td>
</tr>
</tbody>
</table>
RUNNING THE JMETER TEST SCRIPT

Follow these steps to run the Jmeter test script:

1. Enter this command from the bin directory to start Jmeter:
   ```
   jmeter
   ```

2. Once Jmeter starts, select File, Open and enter the location of the EWPS jmeter test script.

3. Select the top most node (EWPS Test Plan) on the left tree view to display the Test plan properties in the right pane.

You must configure these properties for your environment before you run the Jmeter test script:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>Specifies the IP address or server name of the application server hosting EWPS.</td>
</tr>
<tr>
<td>port</td>
<td>Specifies the port of the application server hosting EWPS.</td>
</tr>
</tbody>
</table>
4 Select EWPS Thread Group to display the thread group for the test plan. You can change the number of threads for the thread group to test performance or accept the default to test functionality.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libraryid</td>
<td>Specifies the library ID EWPS will use. This value corresponds to the library Docupresentment is configured to use in the DAP.INI file. This INI file points to the location of the library.</td>
</tr>
<tr>
<td>ids.host</td>
<td>Specifies the IP address or server name of the machine where Docupresentment is running the HTTP server (EWPS communicates with Docupresentment via HTTP with the default setup).</td>
</tr>
<tr>
<td>ids.port</td>
<td>Specifies the port number for the HTTP server running under Docupresentment. The port number is specified in the docserv.xml file for Docupresentment. The default is 49152.</td>
</tr>
<tr>
<td>file.input.URL</td>
<td>Specifies a file URL used by the doCallAPI service operation to generate a file attachment in the request payload.</td>
</tr>
<tr>
<td>file.output.URL</td>
<td>Specifies a file URL used by the doCallAPI service operation to generate a file from a file attachment in the response payload.</td>
</tr>
<tr>
<td>key1</td>
<td>Specifies the company value for the request payload.</td>
</tr>
<tr>
<td>key2</td>
<td>Specifies the line of business value for the request payload.</td>
</tr>
<tr>
<td>userid</td>
<td>Specifies the user account for the request payload.</td>
</tr>
<tr>
<td>effectiveDate</td>
<td>Specifies the effective date to use for forms and sections stored and retrieved via the different service operations.</td>
</tr>
</tbody>
</table>

5 To start the test, make sure EWPS and Docupresentment are running, then select Run, Start from the menu.

6 As the test runs, you can monitor the progress by looking at the summary report. To see this report, select the Summary Report node in the tree view on the left.
To view the results of a particular test case, expand the test case and select its View Results Tree. Then, select the test case node within the View Results Tree and select the Request or Response Data tabs to view the payloads.
Appendix A

Using the Jmeter Test Script to Test EWPS